Proceedings of the 12th Colloquium for Information Systems Security Education
University of Texas, Dallas
Dallas, TX June 2 - 4, 2008

Course Modules for Software Security

Austin Frazier, Xiaohong Yuan, Yaohang Li, Stephan Hudson, *North Carolina A&T State University*

Abstract

Each year the reported number of security vulnerabilities increases as does the sophistication of attacks to exploit these vulnerabilities. Most security vulnerabilities are the result of insecure coding practices. There is a critical need to increase the security education of computer science students, particularly in software security. We are designing course modules, to be used at the undergraduate or graduate level, to integrate software system security into our computer science curriculum. The course modules we have developed, and are developing, include: operating system security, software security testing, code review, risk analysis, and database security. Each course module includes lecture materials, in-class demonstrations, and hands-on assignments. These course modules are designed to be integrated into existing courses. The software security testing and database security modules were taught at this university in the Fall 2007 semester and received positive feedback from student surveys and questionnaires. The other modules will be taught in the Spring 2008 semester. Future work will include the development of more topics in these modules and the creation of new modules in secure software development.

Index terms – Secure Software Engineering, Risk Analysis, Code Review, Operating System Security, Database Security, Secure Software Testing

I. Introduction

Security has become an increasingly important topic in computer science. Each day we learn of new vulnerabilities or hear of new attacks. Most of these vulnerabilities and attacks are, in fact, due to software defects. According to CERT/CC, between 1995 and 2006, the number of reported software vulnerabilities increased by fifty two percent each year.

In recent years, a number of colleges and universities have recognized the need for information assurance education and have developed information security courses in the computer science or related disciplines. However, most of these courses focus on network security. Courses and educational resources on software security are very few. Therefore, there is a significant demand for introducing software security into computer science or information assurance curriculum.

In this paper, we discuss the design and development of software system security course modules. These modules will be integrated into our existing computer science curriculum. We have developed, and are developing, course modules that can be integrated into four courses in the department. Each module consists of several lectures along with in-class demonstrations and laboratory assignments. Instructors can decide how much of a module they would use in their class. The course modules we have developed / are developing include:

(1) Operating system security module
(2) Software security testing module
(3) Code review module
(4) Risk analysis module
(5) Database security module

In the Fall 2007 semester, the database security module was taught in the senior-level undergraduate "Database Design" course and the software security testing module was taught in the graduate "Software Specification, Analysis and Design" course. This course traditionally covers software testing but does not include secure software testing. Both of these modules received positive feedback in surveys and questionnaires that were given to the students.

Currently, the operating system security module has been developed and will be taught in the senior-level undergraduate "Operating Systems" course in the Spring 2008 semester. We are developing the risk analysis and code review course modules, and they will be used in the graduate "Software System Design, Implementation, Verification and Validation" course.

McGraw [1] introduces a framework for secure software development (Figure 1). Lightweight software security best practices, called touchpoints, are applied to the normal steps that make up most software development lifecycles. Each touchpoint is ranked by order of effectiveness. Walden and Frank [2] proposed ten modules that correspond to some of McGraw's touchpoints. These ten modules were taught in a seminar course.

ISBN 1-933510-96-8

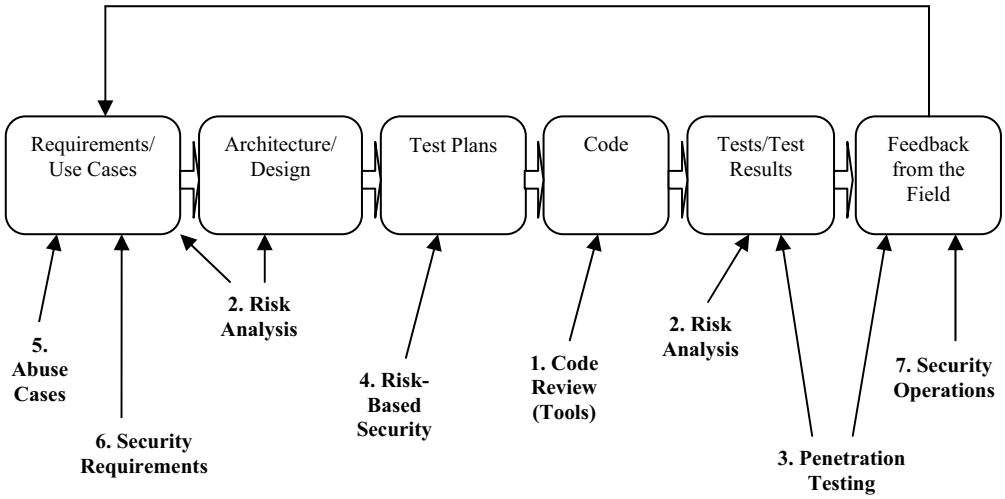

Figure 1: McGraw's secure software development life cycle

The two course modules that we are designing, code review and risk analysis, correspond to the two most effective touchpoints of McGraw's secure software development cycle. The software security testing module corresponds to the penetration testing touchpoint. These modules are similar to those by Walden and Frank [2] in that they cover similar topics; however our modules include more tools. Several different static code analysis tools are used in code review. Also, in code review we added security checklists similar to those created by Taylor and Azadegan [3]. In risk analysis, McGraw's process is presented in addition to Microsoft's STRIDE/DREAD model. In the future, we will develop course modules to correspond to the rest of the touchpoints in the secure software development cycle.

Idaho State's NIATEC [4] has developed eight teaching modules in Information Assurance (IA) that cover a broad range of IA topics. These topics include information protection, PC/Workstation security, security fundamentals, information security law and legislation, system and communications security, corporate security management, etc. Our course modules focus on software system security and can be integrated into a typical computer science curriculum.

The rest of the paper is organized as follows: Sections 2 to 6 describe the course modules we have developed and are developing, Section 7 describes the laboratory environment, and Section 8 concludes the paper.

II. Operating System Security Module

Most operating system courses introduce some security topics such as security threats, access-control, malicious software, etc. The operating system security module we developed builds on that material and provides students with hands-on experiences. The operating system security module includes the following topics:

(1) Buffer overflow and format string attacks [5]. Since most operating system vulnerabilities are caused by insecure coding practices, it is important for the students to learn about dangerous programming errors. The students will learn what buffer overflow and format string attacks are, why they are dangerous, and how to mitigate them.
Buffer overflows are an important topic in operating system security because they are the most common way for an outside attacker to gain access to a system [6]. This course module will illustrate how buffer overflows can be used at the kernel level to gain access to the operating system. An example of this would be exploiting a buffer overflow on a network daemon to gain a root shell. Operating-system level mitigation techniques, such as using a non-executable stack, will also be discussed.
A format string vulnerability was found to affect core APIs in most UNIX versions [5] in CVE-2000-0844 [7]. According to CVE: "Some functions that implement the locale subsystem on UNIX do not properly cleanse user-injected format strings, which allows local attackers to execute arbitrary commands via functions such as gettext and catopen." Samples of these vulnerabilities in C code are introduced along with in-class demonstrations.

(2) The dangers of using weak or default passwords. The students will learn how to create secure passwords. The students will use a password

strength checking website to test the strength of various passwords. In the laboratory assignment, the students will use the open source tool Cain and Abel [8] to do dictionary, brute-force, and cryptanalysis password attacks.

(3) Open network ports, rootkits, and the vulnerabilities of Windows and UNIX services. Students will learn why these are dangerous to operating system security and how to test for them [5] [9]. The students will use the open source tool Nmap [10] to identify unnecessary or rogue services/applications running on the machine. Students will also observe an in-class demonstration that was created for the scanning and detection of rootkits [11].

(4) Operating system vulnerability analysis. The students will use Microsoft Baseline Security Analyzer (MBSA) [12] and Nessus [13] to find the vulnerabilities of a Windows 2000 virtual machine. MBSA is a free tool to determine the security state of the Windows operating system in accordance to Microsoft and to offer recommendations. MBSA reports the status of firewalls, missing updates/patches, unsecured user accounts, shares, unnecessary services, and other security vulnerabilities. Nessus is a vulnerability scanner that uses a vast database to scan and report vulnerabilities for specific operating systems. Students will get exposure to an improperly secured operating system and the reports generated by the two vulnerability scanners. In the case of MBSA, students will learn how to fix the problems found in the report at the system administration level.

The development of this module is ongoing and will include operating system security problems caused by race conditions in the future.

III. SOFTWARE SECURITY TESTING MODULE

Traditional software testing is primarily focused on determining whether software meets requirements and specifications. It mainly looks for bugs that don't follow specifications. However, symptoms of security vulnerabilities are very different from those of traditional bugs. Security bugs are found by looking at additional software behavior, their side effects, and how the software interacts with its environment [9]. Software engineering courses usually only discuss traditional software testing. Software security testing is not covered. Our course module introduces software security testing in addition to traditional software testing.

The software security testing module first introduces a fault model as a basis for security testing. This fault model is made up of four "users" that an application will interface with: the kernel, APIs, the file system(s), and the user interface. Students will learn that based on this fault model, an attack plan can be created which includes attacks that fall into four categories: software-dependency attacks, user-interface attacks, attacks against the application's design, and attacks against the implementation of that design. This course module introduces attacks in two categories: user-interface attacks and attacks against the application's design. These two categories of attacks were chosen initially because software engineering courses typically cover the topics of software design and user interface testing. These two categories of attacks can be easily integrated with existing software engineering topics to introduce software security. The development of this module is on-going and we plan to develop topics for the other two categories of attacks and integrate them to a software engineering course in the future.

The following types of attacks are introduced to the students, grouped in their respective attack category by Whittaker and Thompson [9]:

User-interface attacks:

- Buffer overflows
- Enabling common switches and options through the user interface
- Cross site scripting (XSS)
- Format string attacks

Attacks against the application's design:

- Failing to handle errors
- Forcing the system to reset values
- Creating loop conditions
- Data flow attacks

Each attack is explained in the following format: when the attack is applied, why the attack is successful, how to determine whether security has been compromised, and how to conduct the attack [9]. Code examples, both bad and corrected, and in-class demonstrations are introduced for each of these attacks.

The software security testing module includes three hands-on laboratory assignments. Two of these hands-on laboratory assignments are based on the OWASP LiveCD Education Project [14]. In the first laboratory assignment, students will use WebGoat [15] to perform reflected cross site scripting and stored cross site scripting attacks. WebGoat is a deliberately insecure J2EE web application

designed for teaching software security. In the second laboratory assignment, students will learn how to exploit hidden fields and use WebScarab [16] as a proxy tool to intercept and modify parameters. WebScarab is a security testing tool that is able to intercept both HTTP and HTTPS communications. In the third laboratory assignment, the students will use the Acunetix web vulnerability scanner [17] to scan for XSS vulnerabilities from several Acunetix test web sites.

This course module may be integrated into a junior/senior/graduate level software engineering course that covers traditional software testing.

IV. CODE REVIEW MODULE

Code review was listed as the most important phase in McGraw's secure software development life cycle (SDLC). According to McGraw [1], fifty percent of the security bugs can be eliminated in this phase. It is important to test for security as early as possible in the SDLC. Doing so reduces costs and improves efficiency. It is much more time consuming and costly to search through and patch an application after it has been deployed.

In the code review module, the students will conduct code review using security checklists. This module will be integrated into the graduate level course "Software Design, Implementation, Verification and Validation" in which Personal Software Process (PSP) is taught and practiced. Code review is an important software development phase that the students learn and practice in that course. Security checklists for buffer overflows, integer overflows, input validation, and cross-site scripting have been developed and will be used in this course. Each checklist presents between four to seven conditions that can be checked to indicate the presence of vulnerabilities. The security checklists were developed based on the security checklists presented by Taylor and Azadegan [3] and from other related literature [18] [19] [20] [21] [22] [23] [24] [25]. Extensive sources were used in an effort to make the checklists as complete as possible. However, passing a checklist does not guarantee that the vulnerability has been completely eliminated.

In addition to introducing all of the security checklists, this course module also: presents the pros/cons of using a security checklist, discusses the rationale behind how each of the checklists were created, and gives a code example demonstrating how each checklist is to be used.

Besides using a checklist to look for traditional software bugs, the students will also use the security checklist to look for security bugs during the code review process. Next, the students will learn to use static analysis tools and compare the results they get from their manual code reviews with those found by the static analysis tools. The static analysis tools that will be used include:

- Flawfinder [26] for C/C++ code samples
- RATS [27] for C/C++ and PHP code samples
- OWASP's LAPSE [28] for Java web applications
- Microsoft's XSSDetect [29] for XSS vulnerabilities in C# code samples
- Fortify [30] for C/C++, Java, and C# code samples (Note: This version of Fortify is for demonstration only. It only scans for buffer overflow and SQL injection vulnerabilities.)

Static analysis tools use large rule sets to scan for vulnerabilities and are a much faster way to find vulnerabilities. These tools will be used against code samples, student homework assignments, or a semester long project. Students will perform several hands-on laboratory exercises where they learn to use each tool and analyze the reports, or results, generated by each tool. Students will evaluate the results and check for false positives. They will also be able to compare the results of several tools on the same piece of code. The commercial static analysis tool, Fortify, represents a huge leap over earlier tools because it incorporates compiler technology. As a result, Fortify can track control and data flow compared to only keyword searches from earlier tools.

V. RISK ANALYSIS MODULE

McGraw states that fifty percent of software security problems are caused by design flaws (architectural and design-level problems) [1].Therefore, it is important to identify potential threats, determine the level of security risks posed by these threats, and determine which threats require mitigation. To achieve this goal, the risk analysis module will consist of Microsoft's Threat Modeling framework [31] and McGraw's architectural risk analysis [1].

A threat model is defined as "a security-based analysis that helps people determine the highest level security risks posed to the product and how attacks can manifest themselves" [31]. Threat models also have the added benefit of helping developers understand what the application, as a whole, is actually doing. The students will first learn the threat modeling and risk analysis methodology. They will construct data flow diagrams at different levels to decompose their semester project or selected examples. These diagrams list the data flows and assets of the application. The students will then determine threat targets. The STRIDE (spoofing, tampering, repudiation, information disclosure, and elevation of privilege) method will be used to categorize threats

against these targets. After building one or more threat trees, the DREAD (damage potential, reproducibility, exploitability, affected users, and discoverability) method will be used to calculate the security risk for each threat tree. The students will learn different ways to respond to and mitigate threats. The students will also use Microsoft's Threat Analysis and Modeling Tool [32] to perform risk analysis on their semester project or a selected example. Video demonstrations will be used to help the students learn how to use the tool.

McGraw's architectural risk analysis is made up of three steps: (1) Attack resistance analysis, (2) Ambiguity analysis, and (3) Weakness analysis. Attack resistance analysis uses checklists to identify known threats. Ambiguity analysis is the process of discovering new risks that would be otherwise missed by just looking for known attacks. Weakness analysis involves understanding how flaws in external dependencies affect applications. Students will perform a brainstorming activity in their project groups to identify unknown or creative attacks that might occur against their project and identify any external dependencies for security flaws using security websites such as BugTraq [33] or Security Tracker [34].

VI. DATABASE SECURITY MODULE

Currently at this University, database classes do not address the issue of security. However, due to the numerous and frequent attacks to database servers, there is a strong need to introduce the issue of security into database courses. One particularly popular and harmful type of attack is SQL Injection. Using a SQL Injection attack, a hacker can steal user names and passwords, steal all the tables in a database, or simply delete them.

The database security course module includes a lecture, a hands-on laboratory assignment, and a homework assignment. The lecture discusses how SQL Injection attacks occur and how to prevent SQL Injection attacks. In the laboratory assignment, the students use a SQL Injection attack to gain access to a web site. The students guess usernames and passwords stored in a database. The homework assignment includes questions based on the information discussed in the lecture and in the laboratory assignment.

VII. LABORATORY ENVIRONMENT

The laboratory environment for both the software security testing and operating system security modules is a Windows 2000 virtual machine (VM) that was created using VMWare. In the future, this VM will be upgraded to Windows XP Professional. A VM allows for the creation of one operating system image to be loaded on as many computers in the laboratory as there are students in the class. The VM provides a safe environment for the students to use security tools without harming the environment of the computers in the laboratory. The VM can also be reloaded in the event that it becomes corrupted. VMs save time and provide a uniform environment for all the students during the laboratory exercises. The VM's operating system and software may be updated for use in future semesters, if desired. All in-class demos and laboratory assignments for all the software security course modules are included on a single VM. All the software used was either free, open-source, demonstration, or already licensed by the school. An existing computer laboratory, for computer science students only, will be used to host the software.

VIII. CONCLUSION

We are developing software system security course modules and integrating them into four classes in our computer science curriculum. Each module consists of lecture material along with in-class demonstrations and laboratory assignments. Two of these modules have been taught in the Fall 2007 semester. The other three course modules will be taught in the Spring 2008 semester.

Three of the course modules, i.e., software security testing module, code review module, and risk analysis module correspond to three of the touchpoints of McGraw's secure software development life cycle. In the future, we plan to develop course modules that correspond to the rest of the touchpoints in McGraw's secure software development life cycle. We plan to add the topic of race conditions to the operating system security module, and add the topics of software-dependency and implementation attacks to the software security testing module. We also plan to create a website to disseminate the course modules we developed to the information assurance education community.

ACKNOWLEDGEMENT

Project sponsored by the National Security Agency under Grant Number H98230-07-1-0130. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notation herein.

REFERENCES

[1] McGraw, G., *Software Security*, Addison-Wesley, 2006

[2] Walden, J., and Frank, C., Secure Software Engineering Teaching Modules. In *InfoSecCD Conference '06*, Kennesaw, GA, 2006

[3] Taylor, B., and Azadegan, S., Using Security Checklists and Scorecards in CS Curriculum. In *Proceedings of the 11th Colloquium for Information Systems Security Education*, Boston, MA, 2007.

[4] Idaho State NIATEC site, http://niatec.info/index.aspx?page=105

[5] Howard, M., LeBlanc, D., and Viega, J., *19 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*, McGraw-Hill/Osborne, 2005.

[6] Silberschatz, A., Galvin, P., and Gagne, G., *Operating Systems Concepts*, Seventh Edition, Wiley, 2005

[7] CVE-2000-0844 site, Common Vulnerabilities and Exposures, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0844

[8] Oxid.it Cain and Abel site, http://www.oxid.it/cain.html

[9] Whittaker, J., and Thompson, H., *How to Break Software Security: Effective Techniques for Security Testing*, Pearson/Addison-Wesley, 2004.

[10] Nmap Security Scanner, Insecure.org, http://nmap.org/

[11] F-Secure Blacklight Rootkit Elimination Technology, http://www.f-secure.com/blacklight/

[12] Microsoft Baseline Security Analyzer (MBSA) site, http://www.microsoft.com/technet/security/tools/mbsahome.mspx

[13] Tenable Network Security Nessus site, http://www.nessus.org/nessus/

[14] The Open Web Application Security Project (OWASP) LiveCD Education Project site, http://www.owasp.org/index.php/Category:OWASP_LiveCD_Education_Project

[15] The Open Web Application Security Project (OWASP) WebGoat site, http://www.owasp.org/index.php/OWASP_WebGoat_Project

[16] The Open Web Application Security Project (OWASP) WebScarab site, http://www.owasp.org/index.php/OWASP_WebScarab_Project

[17] Web Vulnerability Scanner, Acunetix, http://www.acunetix.com/vulnerability-scanner/

[18] Wheeler, D., "Secure programmer: Validating input", IBM, http://www.ibm.com/developerworks/linux/library/l-sp2.html

[19] Buffer Overflow Security Module, Embry-Riddle Aeronautical University, http://nsfsecurity.pr.erau.edu/bom/index.html

[20] Stocker, C., "XSS Prevention", http://wiki.flux-cms.org/display/BLOG/XSS+Prevention

[21] Cross-Site Scripting, OWASP, http://www.owasp.org/index.php/Cross_Site_Scripting

[22] Cross-Site Scripting Explained, http://64.233.169.104/search?q=cache:zxpfiCL2cmkJ:crypto.stanford.edu/cs155/CSS.pdf+cross+site+scripting+examples&hl=en&ct=clnk&cd=3&gl=us&client=firefox-a

[23] Ollman, G., "HTML Code Injection and Cross Site scripting", http://www.technicalinfo.net/papers/CSS.html

[24] Viega, J., "Avoiding malloc()/new-related integer overflows", SecureProgramming.com, http://www.secureprogramming.com/?action=view&feature=recipes&recipeid=14

[25] blexim, "Basic Integer Overflows", Phrack Inc., http://www.phrack.org/archives/60/p60-0x0a.txt

[26] Flawfinder site, http://www.dwheeler.com/flawfinder/

[27] Fortify Software RATS (Rough Auditing Tool for Security) site, http://www.fortifysoftware.com/security-resources/rats.jsp

[28] The Open Web Application Security Project (OWASP) LAPSE (Lightweight Analysis for Program Security in Eclipse) site, http://www.owasp.org/index.php/Category:OWASP_LAPSE_Project

[29] Microsoft XSSDetect site, http://blogs.msdn.com/ace_team/archive/2007/10/24/xssdetect-analyzing-large-applications.aspx

[30] Fortify site, http://www.fortify.com/products/sca/

[31] Howard, M. and LeBlanc, D., *Writing Secure Code*, 2nd edition, Microsoft Press, 2003

[32] Microsoft Threat Analysis and Modeling Tool site, http://msdn2.microsoft.com/en-us/security/aa570411.aspx

[33] BugTraq site, http://www.securityfocus.com/archive/1

[34] SecurityTracker site, http://www.securitytracker.com/