# Hands-on Laboratory Exercises for Teaching Software Security

Xiaohong Yuan, Joaquin Hernandez, India Waddell, *North Carolina A&T State University*
Bill Chu, *University of North Carolina at Charlotte*
Huiming Yu, *North Carolina A&T State University*

***Abstract –To meet the growing demand for skilled professionals who can develop secure software, it is important to provide software security education to computer science students in colleges and universities. This paper describes a set of hands-on laboratory exercises we developed to teach software security. These laboratory exercises cover the following topics: code review with tools, web application vulnerability assessment, web spidering, exploiting hidden value, fuzz testing, and threat modeling. Our teaching experiences and related work are also discussed.***

**Index terms – Software Security, Curriculum, Laboratory Exercises**

## I. Introduction

Software security has become an increasingly important topic in recent years. Industries, government and organizations have created a number of initiatives to integrate security into software or product development lifecycle. Some of these initiatives include Microsoft's trustworthy computing initiative, the Building Security in Maturity Model (BSIMM) [1], the Software Assurance Maturity Model (SAMM) [2], the software assurance project in Software Engineering Institute, Carnegie Mellon [3], and many others.

To meet the growing demand for skilled professionals who can develop secure software, it is important for colleges and universities to provide software security education in their computer science or related programs. The Department of Computer Science at North Carolina A&T State University developed a Secure Software Engineering Track in its graduate program. For this track, two new graduate level courses that are required by this track have been developed. The two courses are "Secure Software Engineering" and "Software Security Testing". These two courses have been offered online in the recent a few years.

The "Secure Software Engineering" course uses McGraw's "Software Security" book [4], and the "Software Security Testing" course uses the books "The Art of Software Security Testing" [5] and "The Web Applications Hacker's Handbook" [6]. These two courses cover the following topics:

- The secure software development processes
- Risk management for software security
- Architectural risk analysis
- Attack patterns and abuse cases
- Common design and implementation vulnerabilities
- Taxonomy of coding errors
- Risk based security testing with threat modeling
- Code review with tools
- Security testing methods (white box testing, black box testing, gray box testing)
- Web application vulnerability assessment
- Fuzz testing

We have developed a set of hands-on laboratory exercises to help students learn some of these topics. These laboratory exercises cover the following topics: code review with tools, web application vulnerability assessment, web spidering, exploiting hidden value, fuzz testing, and threat modeling. These laboratory exercises are conducted in Virtual Machine environment. The software tools and applications required by the laboratory exercises are installed in a virtual machine. By providing students with virtual machines, the students can conduct these laboratories using their own computers at home, or using the computers for general purpose in the university. This makes it possible for offering the two new courses on software security as distance-learning courses.

The remainder of the paper is organized as follows: Section 2 describes the hands-on laboratory exercises and the topics of software security they cover. Section 3 discusses our experiences of using the laboratory exercises. Section 4 discusses related work, and Section 5 concludes the paper.

## II. The Hands-on Laboratory Exercises

We have developed a set of eight laboratory exercises to cover code review with tools, web application vulnerability assessment, and threat modeling. In the rest of this section, we will briefly describe each of the above topics, and the associated laboratory exercises.

ISBN 1-933510-95-1/$15.00  

### *A. Code Review with Tools*

Code review is an effective practice to find implementation vulnerabilities. Code review can be conducted by using security checklists, or using static analysis tools. Taylor et. al. [7, 8, 9] have developed security checklists for integer overflow, buffer overflow, and input validation for CS0, CS1, and CS2 courses. Static analysis tools examine the source code of a program without attempting to execute it. Static analysis tools use large rule sets to scan vulnerabilities in the source code. Some of the static analysis tools include RATS [10] for C/C++ and PHP code, Flawfinder [11] for C/C++ code, and Fortify [12] for C/C++, Java, and C# code.

Lab 1. Code review with RATS and Flawfinder
In this laboratory exercise, the students are asked to use RATS and Flawfinder to scan the security vulnerabilities for a C program. The students are asked to compare the results found from RATS and Flawfinder.

Lab 2. Code review with Fortify
In this laboratory exercise, students will use Fortify Source Code Analyzer (SCA) and Audit Workbench to analyze a web application called "TuneStore". The "TuneStore" web application (developed by Bill Chu and his team in the Department of software and Information Systems, University of North Carolina at Charlotte) is an example web application used to demonstrate various vulnerabilities. The TuneStore web application is briefly described below:

The TuneStore web application is an online store selling tunes. A user needs to register and log-in to gain access to the store. The application provides the following options for legitimate user: add balance, friends, profile, CDs and logout. Add balance allows the user to add money to his account by entering his credit card number and the amount of money to be added. The friends function allows the user to add a friend to his list of friends. The CDs function allows the user to view the CDs he purchased. The profile function allows the user to change his password. User can comment on the CDs sold by TuneStore.

The students are asked to use Fortify to scan the source code of TuneStore, and generate the vulnerability report. Then the students are asked to fix the code to prevent SQL injection attacks using parameterized query. The students then run Fortify to scan the source code again to show the vulnerabilities are no longer reported. The students will also demonstrate SQL injection attacks are no longer successful after the code is fixed.

### *B. Web Application Vulnerability Assessment*

Many web applications today are insecure and have a variety of vulnerabilities. Common web application vulnerabilities include broken authentication, broken access control, SQL injection, cross-site scripting, information leakage, etc. [6]. WebScarab [13] is an open source tool that implements a basic intercepting proxy, a web site spider, and a rudimentary fuzzer.

Lab 3. Information gathering with WebScarab
The first step to assess the vulnerability of a web application is to gather and examine some key information about the application. In this lab, the students are asked to use WebScarab's spidering function to enumerate the content and functionality of the web application TuneStore. The students are asked to discuss the pros and cons of mapping web application manually and using a spidering tool.

Lab 4. Exploiting hidden value
A common web application vulnerability is to use hidden HTML form fields to transmit data via client. A hidden form field is not displayed on-screen, however, the field's name and value are stored within the form and sent back to the application when the user submits the form. In this lab, a mock ECommerce web application (Figure 1) is created, which uses hidden form field to store the price of the products. The ECommerce application illustrates the example described in [6]. The students are asked to use WebScarab's intercepting proxy to manipulate the hidden value, so they can purchase a product with cheaper price.

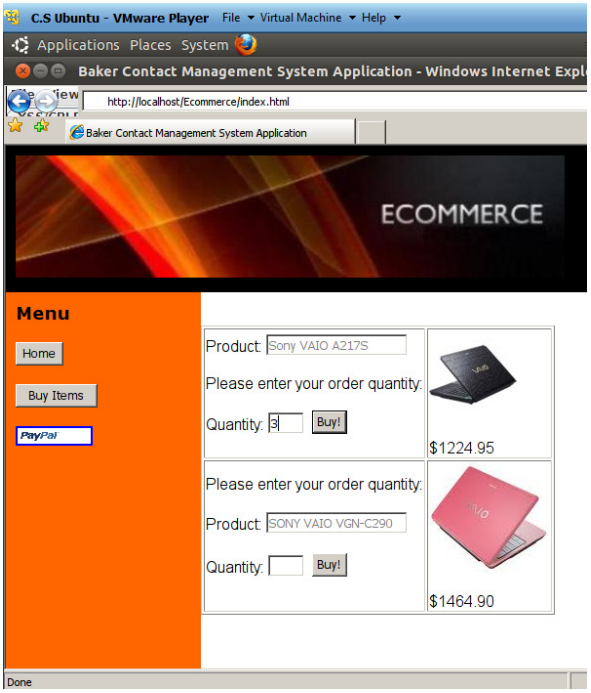

Figure 1. Manipulating hidden form fields on Ecommerce application

Lab 5. Vulnerability assessment of TuneStore and BOG
This lab asks the students to assess the vulnerabilities of the web applications TuneStore and BOG in terms of authentication, access control, SQL injection, and Cross-Site Scripting. BOG is an online blogging site that allows user to post or edit textual blogs. For authentication, students will examine such vulnerabilities as bad passwords, brute-forcible login, verbose failure message, and vulnerable transmission of credentials. The students will not only examine the main login form, but also such functions as registering new accounts, changing passwords, remembering passwords, recovering forgotten passwords, and impersonating other users. For access control, the students will examine vertical and horizontal privilege escalation.

Lab 6. Fuzz testing with WebScarab
Fuzz testing is an automatic software testing method [14]. In fuzz testing, tester supplies malformed and unexpected input data to a program in order to find defects. It is often used to test the security of a program [15]. This lab asks students to use WebScarab's Fuzz plugin to test TuneStore against SQL injection attacks and Cross Site Scripting attacks. The students will generate fuzz strings for SQL injection and Cross Site Scripting attacks, and discuss the testing results.

### *C. Threat Analysis and Modeling*

Threat modeling is a technique that can be used to prioritize security testing [16]. It is a key component of Microsoft's security development lifecycle (SDL). It hypothesizes potential security threats, evaluates the threats based on an understanding of the application's design, and suggests mitigation strategies based on the ranking of the threats. Threat modeling process includes the following stages: identify assets; create an architecture overview; decompose the application; identify the threats, document the threats, and rate the threats. Threats are categorized using the STRIDE (spoofing, tampering, repudiation, information disclosure, and elevation of privilege) method. Threats are then rated according to the DREAD (damage potential, reproducibility, exploitability, affected users, and discoverability) method [17]. Microsoft Threat Analysis and Modeling Tool allows non-security subject matter experts to produce threat model using business requirements and application architecture information [18]. Microsoft Security Development Lifecycle Threat Modeling Tool [19] is a free tool that includes the STRIDE Framework. It helps software developers and architects to identify and mitigate potential security issues during the design phase of SDL.

Lab 7. Threat analysis with Microsoft Threat Analysis and Modeling (TAM) tool
In this laboratory exercise, the students will learn how to use Microsoft Threat Analysis and Modeling tool to create threat models by following the tutorial "Using Microsoft's Threat Analysis and Modeling Tool" [20]. Then the students are given a description of a web application, i.e., a web shop which sells books and related items (videos, cassettes, CDs, etc.). The students are asked to:

(1) Create use cases, access control matrix, and architectural design (the main components/building blocks of the system) for the web shop application;
(2) Generate the threat model for the WebShop application using TAM. Show the screenshot of the threat model.
(3) Generate the threat tree for the WebShop. Show the screenshot of the threat tree.
(4) Generate the comprehensive report for theWebShop.
(5) Evaluate whether the threat model is accurate and complete

Lab 8. Threat modeling with Microsoft SDL Threat Modeling tool
There are four steps for using the Microsoft SDL Threat Modeling Tool: (1) Draw data flow Diagram; (2) Analyze Model; (3) Describe Environment; (4) Generate Reports. In this lab, the students will learn how to use Microsoft SDL Threat Modeling tool to identify possible security threats and possible mitigation methods for a web application [21]. The students will follow the above steps to conduct threat modeling for a selected web application or Tunestore, and discuss the generated reports. The students will discuss the "Recommended Fuzzing" report generated by the tool, and use the report to guide fuzz testing.

## III. TEACHING EXPERIENCES

Most of the labs described above have been used in our Secure Software Engineering or Software Security Testing classes once or several times. We plan to use Lab 7 and Lab 8 this semester. Table 1 lists the software required and the operating systems (OS) environment needed for the above labs.

For the convenience of the students, we created an ubuntu virtual machine on which TuneStore, BOG, E-Commerce, WebScarab, RATs, and Flawfinder are installed. This virtual machine is given to the students to conduct Labs 1, 3, 4, 5, and 6. A windows virtual machine can be created that has Fortify, TAM, and Microsoft SDL Threat Modeling Tool on it. This can be used by students to conduct Labs 2, 7 and 8. Using virtual machines save students time to install required software, and avoid the difficulties of platform incompatibility.

Table 1. The software and OS environment required for the hands-on labs

| Lab | Software Required | OS Environment |
|---|---|---|
| Lab 1. | RATS, Flawfinder | Linux |
| Lab 2. | Fortify | Windows |
| Lab 3. | WebScarab<br>Tunestore | Linux/ Windows |
| Lab 4. | WebScarab<br>Ecommerce | Linux/ Windows |
| Lab 5. | Tunestore<br>BOG | Linux/<br>Windows |
| Lab 6. | WebScarab<br>TuneStore | Linux/<br>Windows |
| Lab 7. | Microsoft Threat Analysis and Modeling Tool (v3.0) | Windows |
| Lab 8. | Microsoft's SDL Threat Modeling Tool v3.1.8 | Windows |

The students in these classes were asked to rate the degree of their agreement on the following statements where 5 indicates strongly agree and 1 indicates strongly disagree.

(1) I know better about putting the technologies or concepts/theories being taught in practice after finishing the related lab exercises/projects.

(2) The lab exercises/projects stimulate my further interests in learning the technology and theories/concepts behind software security.

Sixteen students answered the survey. The average rating of the first statement is 4.25, and the average rating for the second statement is 4.44. The students commented that the "real world" aspects of the assignments made the material more interesting, and they liked using virtual machines for these labs. These labs worked very well in delivering these courses online.

## IV. Related Work

The OWASP WebGoat Project [22] provides a set of web application security lessons using the deliberately vulnerable application WebGoat. The "Exploit Hidden Field" exercise in the Parameter Tampering lesson is similar to our "Lab 4 Exploiting hidden value". Our lab shows a realistic application while WebGoat shows different web pages or context for different lessons. The WebGoat lessons can complement our labs. Students can be asked to complete the related WebGoat lessons before they conduct "Lab 5 Vulnerability assessment of TuneStore and BOG", so the students will understand what techniques to use to conduct attacks on TuneStore and BOG.

Some of the labs we used are similar to those in "The Secure Web Development Teaching Modules (SWEET)" [23, 24]. The "web server security testing" module in SWEET includes labs that use Paros to crawl web pages and hidden web directories, and intercept and tamper requests for privilege escalation. It uses an example web application Badstore.net. These labs are similar to our "Lab 3 Information gathering with WebScara" and "Lab 4 Exploiting hidden value". Our labs use WebScarab instead of Paros, and use different web applications. Lab 4 corresponds to the examples illustrated by Wysopal, *et al.* [5] and by Stuttard *et al.* [6].

The "Web Application Threat Assessment" module in SWEET uses OWASP WebGoat Server to introduce SQL injection, Cross Site Scripting and poor authentication. In our "Lab 5 Vulnerability assessment of TuneStore and BOG", students are asked to conduct vulnerability assessment against TuneStore and BOG from aspects of authentication, access control, SQL injection, and Cross-Site Scripting. This lab requires students to apply the guidelines [6] they learned to realistic applications. No step-by-step instructions are provided to the students, and the solution to the lab is open-ended. Students are asked to discover as many vulnerabilities as possible, or use as many ways to attack the applications as possible. The "Vulnerability Management" teaching module in SWEET guides students to investigate and to modify the Perl CGI Script of a web server that has both SQL injection and cross site scripting vulnerabilities. Our "Lab 2 Code review with Fortify" asks students to modify the java source code of Tunestore to prevent SQL injection attack and other attacks.

The Security Injection Project at Towson University [8, 9] includes a series of modules that can be injected into existing undergraduate courses. These security injections include lab exercises and security checklists which help with teaching security concepts across the curriculum. These injections teach vulnerabilities such as integer errors, input validation, buffer overflow, cross-site

scripting, SQL injection attack, etc. Some of the hands-on exercises on SQL injection and cross-site scripting attacks are based on the OWASP WebGoat application.

The "SEED: A Suite of Instructional Laboratories for Computer Security Education" [25] developed at Syracuse University includes a cross site scripting lab that demonstrate what attackers can do by exploiting cross site scripting vulnerabilities. A vulnerable web-based message board using phpBB was used in the lab. The students were asked to progressively complete a sequence of tasks: post a malicious message, steal cookies, impersonate the victim using the stolen cookies, and write self-propagating XSS worm. This lab requires students to have Javascript and AJAX programming skills. The students could implement some of these attacks on the Tunestore and/BOG applications in "Lab 5 Vulnerability assessment of TuneStore and BOG".

## V. Conclusion

This paper describes some hands-on laboratory exercises we developed and used to teach software security. These laboratory exercises cover such topics as code review with tools, web application vulnerability assessment, web spidering, exploiting hidden value, fuzz testing, threat analysis and threat modeling. We also discussed related work and resources for teaching software security which are alternatives to these labs or complement these labs. Our teaching experiences show that students like the "real world" aspects of these exercises, and these hands-on exercises made the material more interesting to the students. By using virtual machine environment, these labs worked well in online courses. Our future work includes continuing evaluating the effectiveness of these labs, and developing more resources for teaching software security.

## VI. Acknowledgement

This work is partially supported by NSF under grants DUE-0737304, and by Department of Education under grant P120A090049. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation and Department of Education.

## VII. References

[1] The Building Security In Maturity Model, retrieved from http://bsimm.com/

[2] Software Assurance Maturity Model, retrieved from http://www.opensamm.org/

[3] Software Engineering Institute, Carnegie Mellon. Software Assurance, retrieved from http://www.cert.org/work/software_assurance.html

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

[5] Wysopal, C., Nelson, L., Zovi, D., and Dustin, E. *The Art of Software Security Testing*, Addison Wesley, 2007.

[6] Stuttard, D. and Pinto, M. *The Web Applications Hacker's Handbook*, Wiley Publishing Inc, 2008.

[7] 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.

[8] Kaza, S., Taylor, B., Hochheiser, H., Azadegan, S., O'Leary, M., and Turner, C. F. Injecting Security in the Curriculum – Experiences in Effective Dissemination and Assessment Design, In *Proceedings of the 14th Colloquium for Information Systems Security Education,* Baltimore, Maryland, June 7-9, 2010.

[9] Secure Injections, retrieved from http://triton.towson.edu/~cssecinj/secinj/

[10]Fortify Software Inc., Welcome to RATS-Rough Auditing Tool for Security, retrieved from https://www.fortify.com/ssa-elements/threat-intelligence/rats.html

[11] Flawfinder, retrieved from http://www.dwheeler.com/flawfinder/

[12] Fortify -- an HP Company, retrieved from https://www.fortify.com/

[13] Category: OWASP WebScarab Project, retrieved from http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

[14] Takanen, Ari. Fuzzing: the Past, the Present, and the Future. Codenomicon Ltd. 2008. Retrieved from http://actes.sstic.org/SSTIC09/Fuzzing-the_Past-the_Present_and_the_Future/SSTIC09-article-A-Takanen-Fuzzing-the_Past-the_Present_and_the_Future.pdf

[15] "OWSASP: Fuzzing with WebScarab", retrieved from http://www.owasp.org/index.php/Fuzzing_with_WebScarab

[16] Chatper 3 Threat Modeling, retrieved from http://msdn.microsoft.com/en-us/library/ff648644.aspx

[17] Uncover Security Design Flaws Using The STRIDE Approach, retrieved from http://msdn.microsoft.com/en-us/magazine/cc163519.aspx#S3

[18] Microsoft Threat Analysis and Modeling v2.1.2, retrieved from http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=14719

[19] SDL Threat Modeling Tool 3.1.8, retrieved from http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=2955

[20] Meneely, A., Smith, B. and Williams, L. Using Microsoft's Threat Analysis and Modeling Tool, retrieved from http://agile.csc.ncsu.edu/SEMaterials/tutorials/msthreatmodeling/#section0_0

[21] Hernan, S., Lambert, S., Ostwald, T., and Shostack, A. Getting Started With The SDL Threat Modeling Tool, retrieved from http://msdn.microsoft.com/en-us/magazine/dd347831.aspx

[22]Category: OWASP WebGoat Project, retrieved from https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project

[23]Chen, L. C., Tao, L., Li, C. "A Tool for Teaching Web Application Security," *Proceedings of the 14th Colloquium for Information Systems Security Education*, Baltimore, Maryland, June 7-9, 2010.

[24] Secure Web Development Teaching Modules, http://csis.pace.edu/~lchen/sweet/

[25]SEED: Developing Instructional Laboratories for Computer SEcurity Education, http://www.cis.syr.edu/~wedu/seed/