Proceedings of the 10th Colloquium for Information Systems Security Education
University of Maryland, University College
Adelphi, MD June 5-8, 2006

Software Security: Integrating Secure Software Engineering in Graduate Computer Science Curriculum

Stephen S. Yau, *Fellow, IEEE*, and Zhaoji Chen
Arizona State University, Tempe, AZ 85287-8809
{yau, zhaoji.chen@asu.edu}

Abstract – In addition to enable students to understand the theories and various analysis and design techniques, an effective way of improving students' capabilities of developing secure software is to develop their capabilities of using these theories, techniques and effective tools in the security software development process. In this paper, the development and delivery of a graduate-level course on secure software engineering with the above objective at Arizona State University are presented. The developing process, stimulating techniques and tools used in this course, as well as lessons learned from this effort, are discussed.

Index terms – Information assurance, software security, secure software engineering, graduate curriculum, course, theory, techniques, tools, course project, and lessons learned.

I. INTRODUCTION

In this *information era*, information systems and networks often consist of software systems running on many interconnected computers with various capabilities, such as servers, desktops, laptops, PDAs, and even cell phones. In these systems, connectivity has become more important than ever before [1]. The pervasive connectivity has greatly enhanced our ability to fast share information and computing resources, but it has also greatly increased the chances for attackers to launch malicious attacks. The increasing complexity and extensibility of software systems further complicate the situation as they introduce more security breaches and make the information systems more vulnerable to failures and attacks. According to CERT Coordination Center, there were more than ten new security vulnerabilities reported per day in 2004 and the number almost doubled in 2005 [2]. Failures caused by attacks exploiting these vulnerabilities are very costly. According to a NIST report [3], the U.S. economy spends $59.5 billion in breakdowns and repairs cost caused by software errors.

Software security, which is the idea of engineering software such that it can function correctly and continuously under malicious attacks [4], has attracted much attention recently due to the fact that reactive network-based security approaches, like firewalls and signature-based anti-spyware, have been shown ineffective to achieve secure software. Furthermore, fixing software after release is very costly. The later the security is addressed in the development cycle, the costlier it becomes: one dollar required to resolve an issue during the design phase grows into 60 to 100 dollars to resolve the same issue after the software is shipped [5].

It is obvious that a better way to achieve secure software is to incorporate security in the software starting from the beginning of the development process. However, because software developers tend to focus the cost and time on meeting well-specified functional requirements and leave security issues for maintenance in the infamous *penetrate and patch* manner [6], a large amount of unnecessary effort is put in fixing security defects through patches, service packs, or generating new versions after these defects have been exploited and caused problems.

An effective way of developing secure software is to educate and train software developers on critical software security issues. Industry has already taken steps in this direction. For examples, Microsoft has launched its Trustworthy Computing Initiative, and IBM has started its SPADE (Security and Privacy Aware Development Environment) project. Beyond awareness, software developers should gain more software security knowledge and know how to follow the best practices of developing secure software through various educational and training programs.

The knowledge of software security is multifaceted and applicable in a diverse way [7], involving security requirement engineering, design principles and guidelines, implementation risks, analysis techniques, and security testing. With proper course and laboratory material, universities should enable students to understand the theories and techniques as well as use effective tools for secure software development.

Before this year, at Arizona State University (ASU), like many other universities, the Computer Science and

ISBN 1-933510-98-6/$15.00

Engineering curriculum did not have a course to address the overall secure software development, although some of the issues or specific techniques are covered in courses, such as the graduate-level courses *Applied Cryptography* and *Computer and Network Security*. Thus, as part of our effort to meet the NSTISSI-4011 and CNSSI-4012 standards and establish a National Center of Academic Excellence in Information Assurance Education [8], we have developed a new graduate-level course, *Software Security*, to focus on the basic concepts, various analysis and design techniques, as well as the latest research results to achieve secure software development. This course is intended to change students' behavior in developing secure reliable software, improving public awareness of this subject, as well as promote related education in local communities. It is being offered for the first time at ASU during this Spring semester.

In this paper, we will present how this course was conceived, developed, and offered. Although this semester will end in early May, we have gained sufficient experience to discuss the lessons we have learned. Curriculum artifacts are available at our course website [9].

II. Course Development

A. Goals

The primary purpose of this course is to encourage students to adopt best practices in secure software engineering and stimulate students' interest in research to advance the state of the art in this area. We have established five specific goals for the course: the students should be able to do the following after taking this course:

1. Understand secure software engineering principles and best practices.
2. Apply these principles and practices in their software development process.
3. Recount well-known software vulnerabilities.
4. Demonstrate design and coding skills to avoid these vulnerabilities.
5. Describe and apply various software analysis, testing and verification techniques and tools.

These goals are accomplished through lectures, research paper presentations and discussions, and a software development project throughout the whole semester.

B. Course Scope and Content

The course content consists of a general introduction and some specific information items defined in the NSTISSI-4011 standard [8]. Since the course has only thirty 75-min classes for one semester, its specific topics are carefully chosen to provide students with the necessary background as well as the latest research results regarding software security. Specifically, the course covers the following major topics:

- Goals and technical trends of software security.
- Requirement engineering and system design
- Software vulnerabilities
- Software analysis and verification
- Software security testing

Considering the distribution of information items, course schedule, the amount of effort required for students to digest the material, we mix the formal lectures given by the instructor with research paper presentations with extensive discussions by students, and the course project presentation with demonstration by students. The order of these topics was arranged to facilitate a good learning experience. Because the topics follow the normal software development process and proceed in a similar pace as the progress of the student course project, students can easily apply the knowledge they have learned in the class to their own course project.

C. Target Students

The course has attracted both master students and Ph.D. students at ASU. The students in the class have different background. Some are veterans who have been working in industry with more than 10 years with software development and maintenance experiences. Some are fresh graduates without any software development experiences besides the course projects they had during their undergraduate studies. They also have quite different reasons to take the graduate study. Some just need an advanced degree to improve their career track. Some like to do research and exploit the unknown and plan to do MS theses or Ph.D. dissertations on this subject. Some are mainly interested in this subject and want to broaden their knowledge.

Thus, even though all students registered for the course are interested in the topics, they have quite different expectations for what they want to learn from this course. One of the challenges the instructor faces is to acknowledge the differences and coordinate activities during classes to stimulate all students' interests and meet their individual anticipations.

Based on the educational experience of one of the authors, we think that students are more motivated and devoted when they are doing what they like and they are capable of generating good results. Hence, in order to meet these challenges, we have designed the course to be relatively

flexible. Although we still cover a broad range of topics through formal lectures given by the instructor, a list of research papers is provided to students, and each student selects a paper to present based on the student's interest, ability as well as schedule. Each student is required to study his/her selected paper thoroughly and present the results followed by extensive discussions.

D. Course artifacts

As shown in Figure 1, unlike some other courses that are built around the instructor, and disseminate the information through lectures in mostly a one-way communication manner, this course was built up around students. A great amount of time and effort is focused on discussing the latest research results on this subject through paper presentations and building up the hands-on experience through a carefully prepared software development project. We will discuss each of these course artifacts in detail in the following subsections.

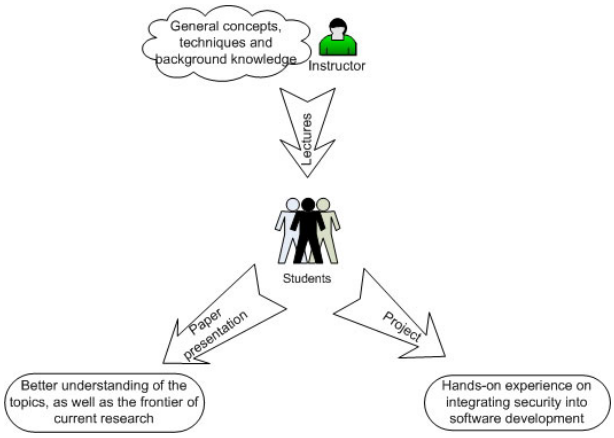

Figure 1. Basic Course Structure

1. Topic Outline

The course begins with a brief review and background introduction of the goals and technical trends in secure software engineering. Other related information assurance concepts and mechanisms are also introduced in the first two weeks. Then, various aspects and techniques as well as supporting tools related to software security are organized in sections, which are taught in an order according to normal software development process: starting from requirement engineering, design, followed by implementation, and testing and analysis come at last. Specific topics are covered according to the following outline and each topic is mapped back to one or two learning goals listed in Subsection *A. Goals*. The first goal is achieved mainly by instructor's lectures. The second and fourth goals are achieved through the software development process in the course project. The third and fifth goals are achieved by both instructor's lectures and student paper presentations.

Part 1: Overview (Weeks 1 ~ 2)

- Review goals and technical trends of software security.
- Discuss information assurance issues and concepts related to software security.

Student course project starting point (Week 2)

- Project requirements and timetable are discussed

Part 2: Secure software engineering (Weeks 3 ~ 5)

- Lectures: review software design principles and best practices for building reliable, robust software systems.
- Lectures: aspect-oriented software design modeling related to various QoS, including security.
- Lectures: integrate security in software life cycle.

Student course project milestone 1 (Weeks 6-7):

- Design document presentation

Part 3: Software vulnerabilities (Weeks 8 ~ 9)

- Lectures: introduce the concept of both language dependent vulnerabilities, like buffer overflow and format string, and language-independent vulnerabilities, like race conditions and privilege control, and review related mitigation techniques, practices or supporting tools.
- Paper discussion on statically detecting buffer overflow vulnerabilities.
- Paper discussion on dynamically detecting buffer overflow vulnerabilities.

Midterm exam (Week 9—one class in Week 9)

Part 4: Software analysis and verification (Weeks 10 ~ 11)

- Lectures: review both static analysis techniques, like model checking, type qualifier and logic proof, and runtime analysis techniques, like sandboxing and intrusion detection. Various analysis and verification tools are also introduced.
- Paper discussion on modeling and analysis of security system architecture.
- Paper discussion on formal logic proof of security requirements.

Part 5: Software security testing (Week 12)

- Lecture: review various testing approaches and testing tools to address security issues, including specification-based testing, property-based testing, fault injection testing, etc.
- Paper discussion on testing by fault injection.

Student course project milestone 2 (Weeks 13 ~ 15)

- Presentation of the complete development process and discussion of lessons learned
- System demonstration
- Trial use of the system by other students

Each part starts with a lecture by the instructor. This lecture covers the overview of this section, the necessary concepts and various available techniques and tools in general. After the students have the general understanding of the topics covered in this section, the research papers related to the topics are presented and discussed by students in detail, especially the specific approaches or techniques presented in these papers.

The course project is announced in Week 2 and spans throughout the whole semester with two major milestones. The first milestone is in Week 6, when the students are required to present their software design. The instructor gives individual students feedback on their design as well as some general comments on the improvements most students should make so that possible design flaws can be corrected before getting into the implementation phase. The second milestone is at the end of the semester, where each student should finish the development and present the complete results as well as discuss his/her experience about integrating security consideration in software development. After the demonstration of each student's system for the course project, other students were encouraged or appointed to use the system and detect security vulnerabilities in the system.

2. Research Paper Presentations and Discussions

Since this course is a graduate-level course, one of the goals is to stimulate students' interest in research to advance the state of the art in this area. Besides the basic concepts, techniques and design principles, we would like to expose our students with the current state of the art in this area. A list of recent research papers is selected to accompany the formal lectures [9]. Each student was required to select a paper in the list, did their study and presented the paper in class. Students might also suggest different research papers, but they must be relevant to software security and contain sufficient technical information.

One obvious benefit of this arrangement is that the students are more involved. The students used the basic concepts and principles they have learned from the lectures, studied the papers themselves to understand specific techniques and approaches presented in the papers. This study process not only introduces more advanced topics to students, it also helps them clarify and reinforce those basic concepts and principles by requiring them applying the knowledge they have already learned in the study. The students have also learned from the authors of these papers how to conduct research and solve new problems. We believe that this experience will be very helpful when the students start their own research.

Due to the limited time each student could afford to devote to one course, each student was required to study only one paper thoroughly, make a presentation on the approach and other related research work covered by the paper, as well as discuss the strength and weakness of the approach. However, all students were required to read the paper before the presentation and actively participate in the discussions followed by the presentation.

3. Course Project

Besides introducing techniques needed to develop secure, reliable software to our students, the most important result we want to achieve is to convince students the importance of secure software engineering and truly change their approach to developing software after taking this course. To achieve this, nothing does it better than by creating a close to real-world task and letting students experience the difference and benefits they can have by applying the techniques they have learned on the software development process.

We have prepared a secure software development project that has sufficient requirements and complexity to expose students with possible problems they may face later in their career, and yet still be relatively simple so that it can be completed within one semester. This project is to develop a *web-based system for Collaborative Software Development*, which facilitates the members in different teams to collaborate efficiently during software development. The system is required to manage all the documents, code and test plans for the project. A user should be able to use this system wherever Internet access and web browser are available. The functional requirements we specify for the project are relatively easy because our main purpose for the project is to let students go through the whole development process and spend more effort on how to meet the security requirements. The major security requirement for such a system is that "collaborators' access to the system should be restricted to only those parts required for the collaboration".

The students were required to start from requirement analysis and develop their design and present the design to the whole class. Since all the students are doing the same project, they all have thought through the details of this project while preparing their own system design. Thus, problems in one student's design were likely to be caught by other students. A student might also get some hints from other students' different choices to address certain issues. We would also like to have students experience as being both a "defender" and a "critique" by introducing peer-evaluation in this project. A student has a chance to test and try to break other students' software after the students complete their projects toward the end of the semester. To stimulate students to put more effort in reducing vulnerabilities in their own software and find more vulnerabilities in other people's work, we have decided to give extra credit for this peer-evaluation process as follows: A student would get 1 extra point for each vulnerability he/she found in other fellow student's work. In the mean time, the student responsible for the vulnerability would have 0.5 point deduction. Since this was only one criterion to judge students' performance, the maximum number of points a student could earn or lose would be limited to 5 points.

4. Tools

Various tools are available to detect implementation vulnerabilities and support secure software development. We believe that students should be able to utilize these resources when they develop secure software. In this course, we introduced various tools throughout lectures and study of research papers, including

- BOON for static buffer overflow detection
- PScan for static detection of format-string vulnerabilities in printf()
- CQual for static detection of inconsistent usage of value in C code
- MemWatch for dynamic detection of memory allocation and free misuse

Besides introducing these tools and the techniques they used, we encouraged students to use open-source tools in their own course project development to detect the corresponding vulnerabilities in their programs. This allowed students to focus on fixing the detected vulnerabilities.

5. Student Background Survey

As mentioned in Section *C*, students taking this course have different backgrounds and different motivations. In order to know the students in the class better, we conducted a background survey during the first class. We ask the following questions:

- Course background: What relevant courses has the student taken before or during the same semester?
- Educational background: What degree(s) has the student received and what degree program he/she is in now?
- Software design and development experience: What kinds of systems has the student designed and/or developed before?
- Experience: What system platforms, developing environments, tools and techniques has the student had experienced or used?
- Working experience in industry
- Research experience

From the responses we received, we could have very good ideas about a student's experience, skill and specialty, which were very helpful when we assigned or recommend certain research papers to each student.

6. Textbooks

Our course materials were developed based on various resources: books, research papers, online articles, and related presentations. Instead of specifying any specific textbook, we adopted the following two books as reference books, where students could find complementary examples and in-depth explanations.

- M. Howard and D. LeBlanc, *Writing Secure Code*, Microsoft Press, 2001. [1]
- G. Hoglund and G. McGraw, *Exploiting Software: How to break code?*, Addison-Wesley, 2004. [10]

III. Expected Course Outcomes

The course is expected to achieve the following outcomes

EO1). Establish a solid graduate-level course on software security in the Computer Science and Engineering Department at ASU as part of our program in Information Assurance Education.

EO2). Change of the ways of the students in developing secure software after taking this course.

EO3). Improve public awareness of software security among all students at ASU by publicizing the availability of this course.

EO4). Incorporate some appropriate materials generated in this course in our undergraduate software engineering capstone courses.

EO5). Promote incorporation of software security in various educational programs of local communities transferring appropriate results of this course to local community colleges.

Since the current semester is the first semester this course is offered, we do not have the complete results yet because we still have three weeks to complete this semester. The following evaluation will be conducted after students have completed the course.

1. The course project was designed to incorporate all the learning goals. Thus, how the students perform in developing the course project will indicate how students are faring in meeting those goals.

   The students' performance on the course project can be judged based on how they use the knowledge they have gained in their design and implementation, as well as their ability to find possible problems in other students' deliverables.

2. A questionnaire will be distributed to students at the end of semester, asking students about what they consider they have be benefited from this course and the changes happened to their ways of developing software.

   We also plan to conduct another similar student survey 6-12 months later to find out the long term impact on students and whether the changes we find in the previous survey will last beyond the classroom.

3. Another possible evaluation we planned to conduct is to use the course project with the same requirements in our general software engineering courses and compare the difference in performance of students who have not taken this course and the students have taken this course in terms of security vulnerabilities in their software systems.

IV. Lessons Learned

Based on our experience in developing and offering this course, we have learned the following lessons:

1. While delivering the course, we noticed that, instructor's guide and suggestions are critical during the discussions in class. After a student gave a presentation, if the instructor did not raise many questions, other students usually said they do not have questions or comments. They actually had some ideas or confusions about the presentation, but they just did not know how to ask questions or express their thoughts in a clear and organized manner. After the instructor offered some suggestive questions, more students would join in the discussion and some of them could offer very good ideas or insights.
2. We specify the basic software engineering course (*CSE360 Software Engineering*) as the perquisite for this course. But, based on the design documents of the course project generated by students, we found that some of the students did not have any software engineering practices even though they had taken the basic software engineering course or its equivalent. They did not have enough training on how to generate good software documentations. We plan to add a tutorial lecture on general software engineering practices the next time we offer this course.
3. During the final presentation phase of the course project, we are very pleased to notice that students were very passionate in finding vulnerabilities in other students' programs by trying various attacks. When a student's program was hacked right away during his presentation, all other students in the class learned the problems of the program, their attack techniques and the vulnerabilities being exploited.
4. The students seem to be more interested in attacking other students' programs than doing rigorous testing to protect their own programs. This phenomenon may be due to the extra points of credit the students will get for the number of errors or security vulnerabilities they can find in others' programs. Another explanation may be that students treat a successful hacking as a proof to show they are better than the student who developed the software. In order to lead the students in the right direction, it is necessary to emphasize that attacker's job is much easier than defender's. Students should be more proud of their capability of defending their software from various attacks.

V. Conclusions and Future Work

We have been so far quite pleased with the experience in offering this course. The course will be an important part of the information assurance concentration in our M.S. and Ph.D programs we are trying to establish.

This is the first course in software security offered at ASU. In the future, we plan to establish a computer and network security laboratory for this course for the students to experiment how their systems perform with various attacks. We also plan to incorporate appropriate content developed in this course in our undergraduate capstone software engineering courses, *Software Engineering Project I and II* so that when students take

this graduate-level course, they will be better prepared. In addition, interesting topics in software security may be covered in more depth by adding appropriate new general content in our existing courses, like *Software Requirements and Specification* and *Software Verification, Validation, and Testing*.

VI. References

[1] M. Howard and D. LeBlanc, "Writing Secure Code", Microsoft Press, 2001.

[2] CERT Coordination Center, CERT/CC statistics 1988-2005. Available at: http://www.cert.org/stats/cert_stats.html

[3] National Institute of Standards and Technology, "Software Errors Cost U.S. Economy $59.5 Billion Annually" (NIST2002-10). Available at: http://www.nist.gov/public_affairs/releases/n02-10.htm

[4] Gary McGraw, "Software Security", *IEEE Security & Privacy*, vol. 2(2), 2004, pp. 80-83.

[5] K.S. Hoo, A.W. Sudbury, and A.R. Jaquith, "Tangible ROI Through Secure Software Engineering", *Secure Business Quarterly*, vol.1(2), 2001.

[6] J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way?,* Addison-Wesley, 2001

[7] S. Barnum and G. McGraw, "Knowledge for Software Security", *IEEE Security & Privacy*, vol. 3(2), 2005, pp. 74-78.

[8] NSA, "National IA Education & Training Program", Available at: http://www.nsa.gov/ia/academia/cnsstesstandards.cfm

[9] CSE591 Software Security at: http://enpub.fulton.asu.edu/iacdev/courses/CSE591s/home.html

[10] G. Hoglund and G. McGraw, "Exploiting Software: How to break code", Addison-Wesley, 2004