AC 2012-4501: TEACHING SOFTWARE SECURITY: A MULTI-DISCIPLINARY APPROACH

Dr. Walter W. Schilling Jr., Milwaukee School of Engineering

Walter Schilling is an Assistant Professor in the Software Engineering program at the Milwaukee School of Engineering in Milwaukee, Wis. He received his B.S.E.E. from Ohio Northern University and M.S.E.S. and Ph.D. from the University of Toledo. He worked for Ford Motor Company and Visteon as an Embedded Software Engineer for several years prior to returning for doctoral work. He has spent time at NASA Glenn Research Center in Cleveland, Ohio, and consulted for multiple embedded systems companies in the Midwest. In addition to one U.S. patent, Schilling has numerous publications in refereed international conferences and other journals. He received the Ohio Space Grant Consortium Doctoral Fellowship, and has received awards from the IEEE Southeastern Michigan and IEEE Toledo Sections. He is a member of IEEE, IEEE Computer Society, and ASEE. At MSOE, he coordinates courses in Software Quality Assurance, Software Verication, Software Engineering Practices, Real Time Systems, and Operating Systems, as well as teaching Embedded Systems Software.

Dr. Eric Durant, Milwaukee School of Engineering

Eric Durant is an Associate Professor and Director of the computer engineering program in the EECS Department at Milwaukee School of Engineering. In addition to information security, he enjoys teaching many subjects, including digital logic and digital signal processing. He is active in hearing aid algorithm research, where he holds one U.S. patent and has three pending. His current focus is on beam-forming and noise reduction.

Teaching Software Security: A Multi-Disciplinary Approach

Abstract

As computing devices become more and more ubiquitous, the importance of software security cannot be overlooked. As such, many software engineering and computer science programs offer an elective course in software security. While the title of these courses is often similar, the content is often vastly different, reflecting the large domain of software security. Certain aspects of security appeal to practitioners, certain aspects appeal to Computer Scientists, and certain aspects apply MIS personnel.

In order to provide a holistic view of computer security, software engineering students need to have exposure to all three aspects. Thus, for software engineering students, a single course in security can be inadequate. To combat this problem, the Milwaukee School of Engineering has developed a three course sequence in software security targeting the multi-disciplinary problem of security. While each of the three courses addresses software security, each course targets a different aspect. An Introduction to Network Security offers students an understanding of the nature of network security. Secure Software Development focuses the design and construction of software systems in a manner in which security is built into the product from the beginning of development. Information Security offers students an understanding of the techniques used to ensure that data and other systemic information is protected using the most appropriate techniques. Since the development of this sequence, one or more of these courses has been taken by approximately 100 students in the software or computer engineering programs. This article provides an overview of the three courses offered at Milwaukee School of Engineering, the challenges of offering these courses as independent electives, and student impressions of the security course series.

Introduction and Course Overviews

Within the software and computer engineering fields, the understanding of security is paramount to the successful construction of modern systems. By definition, Software Security is "the practice of building software to be secure and to function properly under malicious attack."[1] To this end, the Milwaukee School of Engineering has developed a Computer Security application domain for software engineering students. An application domain is, by definition, a 9-credit sequence of courses focusing on one aspect of software engineering or the application of software engineering skills to a given application area. Graduates from MSOE are required to complete this sequence as well as successfully complete two technical electives to graduate.

The Computer Security application domain reflects a holistic approach computer security and includes instruction in all areas of security which have been deemed relevant and important for students in the software engineering program. These three pillars, shown in Figure 1, contribute to a complete and thorough exposure to the breath of Computer Security while still permitting future study by students.

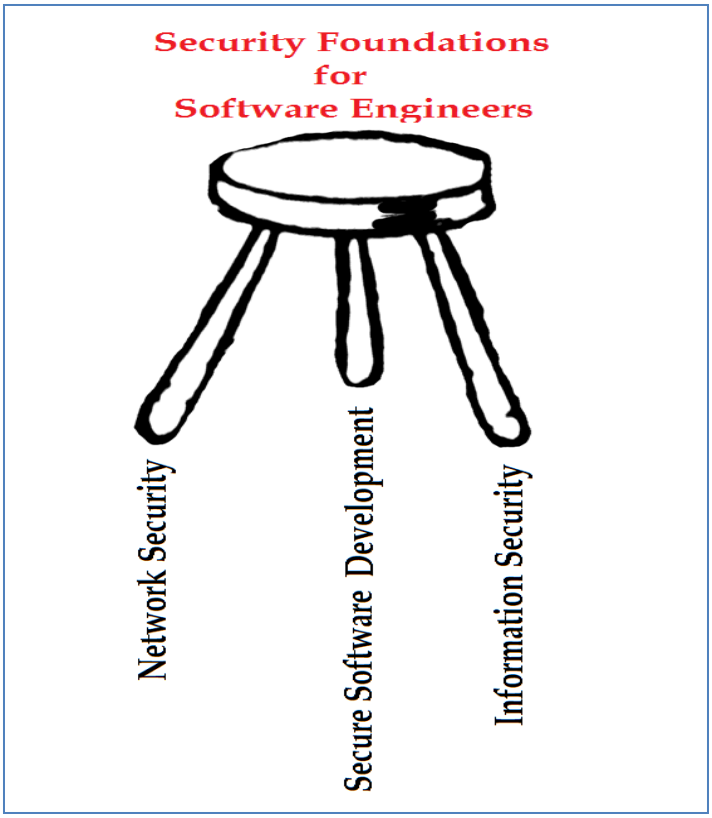

Figure 1 The three courses within the Software Security Application domain.

Network Security Tools and Practices

The network security tools and practices course is intended as an introduction to the foundational skills and background knowledge necessary to act as a network security professional. The lecture material for this course balances presentation of topics with demonstrations of tools used in the network security profession. As the course prerequisites do not require a background in networking, the early lectures cover the basics of TCP/IP networks and application protocols like HTTP and HTTPS.

After establishing the network knowledge required for the course, foundational computer security topics are explored. The principals of Confidentiality, Integrity and Availability are defined and examples are provided related to network computing environments. Threats against network assets are identified along with adversaries who would attack the network. Next, a malware taxonomy is defined to include the following types: virus, trojan, rootkit and worm. Real world examples of each type of malware are evaluated in class by the students and presented within the context of the taxonomy. Additional topics covered in lecture include network defenses (e.g. firewalls, spam filters, intrusion detection), network forensics and penetration testing.

Table 1 Course outcomes for each of the three security courses in the Secure Software Application Domain.

Course	Course Outcomes
Information Security	1. Discuss the business case and the need for an increased focus on computer security, including types of vulnerabilities (social engineering, insecure libraries, etc.) and how current vulnerabilities are disseminated by the software community 2. Analyze computing systems with an awareness of various timely legal issues related to security and privacy 3. Choose appropriate security implementation techniques based on secret and public key cryptography, the use of hashing, and other cryptographic principles 4. Appraise competing tools for common security practices, such as public key encryption, firewalling, and securing network traffic
Secure Software Development	1. Construct and document Software Abuse Cases. 2. Analyze the threats against a software system and determine mitigation actions for these threats. 3. Perform an architectural risk assessment for a software system. 4. Design a software system using secure software design techniques. 5. Use the design principles of software security to ensure that a system is designed in a secure fashion. 6. Assess a software package for security vulnerabilities using a commercial grade static analysis tool.
Network Security	1. Assess and evaluate network security tools for use in defending, attacking and testing computer networks. 2. Design a threat scenario and implement defenses to mitigate potential attacks. 3. Perform a penetration test of a live network and assess the results. 4. Discuss the legal and ethical issues involved with assessing and testing a network for vulnerabilities and weaknesses. 5. Discuss the roles and responsibilities of network security professionals.

A key component of the lecture material for this course is demonstrations of network security tools and in-class activities to promote active learning. The first time this course was offered there was considerable student feedback that supported an additional focus on demonstrations. Although this caused a reduction in the number of topics covered in lecture, feedback on the course was much more positive in its second offering.

Examples of in class demonstrations include: basic use of the Backtrack Linux distribution in a virtual machine environment, network discovery and scanning of the university's computing resources and packet sniffing of wireless network activity in class. These demonstrations were very well-received by students and typically were previews of the tools and techniques that were needed to complete an upcoming lab exercise.

Although most in-class activities were brief and involved testing or configuring software for use in lab, one example of a longer activity is when the students were asked to engineer, but not implement, a hypothetical distributed denial of service attack on the instructor. The students worked in teams to identify the attack vectors and how to make the attack difficult to detect and defend against. All results were reported back to the class throughout the lecture period. At the end the teams were asked to determine how the university could adequately defend itself against the attack. In this case the solution was to disable an account lockout policy that made the DDOS possible in the first place.

Another interesting in-class activity has students pair up and attempt to perform a man-in-the-middle attack on their partner using a combination of ARP poisoning and the sslstrip tool to view the contents of HTTPS requests in plaintext. Due to the complexity of this attack, only two students were successful in getting the full attack to work. This illustrates one of the core difficulties in teaching this course as the tools and network infrastructure often did not perform consistently or reliably.

Lab Coverage

The lab portion of this course focused on the application of network security tools available on the Backtrack 5 Linux distribution. Backtrack 5 is a Ubuntu Linux-based distribution customized for performing network penetration testing.

The first lab involves getting students up to speed with essential UNIX commands and has them explore some of the many tools available on the BackTrack distribution. The first part of the lab requires the students to set up a BackTrack 5 virtual machine (VM) environment or install BackTrack directly onto their laptop. This can be done with the Wubi installer which will allow BackTrack to be booted alongside Windows. Most students chose to use VMs. This was by far the most flexible approach. However, some students experienced network connectivity problems throughout the course on the university's wireless network which could not be easily reproduced or fixed.

The second lab focused on network reconnaissance, information gathering and scanning for hosts and vulnerabilities. It further expanded the networking and Linux knowledge of the students while exposing them to some of the most basic tools in a network security practitioner's toolset including tcpdump and wireshark for packet analysis and nmap and Nessus for scanning.
The third lab had students enumerate and sniff packets on open wireless networks using the aircrack-ng toolset included with BackTrack. Students captured packets from the network without actually joining the network using passive surveillance techniques. An important lesson learned by the students was how important it is to secure wireless networks with some form of encryption. Many students commented that they were unaware how easy it is to monitor traffic on open wireless networks.

The fourth lab had students working together to implement an intrusion response system that was capable of detecting simple port scans through the use of scanlogd and ARP poisoning attempts through the use of arpwatch. When scans or poisoning attempts were detected, the target system responded to the threat by blocking the attacking IP address with iptables.

The fifth lab was a quick introduction to the Metasploit framework bundled with BackTrack. This lab coincided with the midterm exam so it was intentionally straightforward. Students followed several tutorials of their choosing from a provided list, all of which would be useful for the upcoming final project.

The final project involved four teams attempting to defend their own live network from the other teams during two two-hour lab sessions. Each team had to deploy instances of the Metasploitable VM and the OWASP Broken Web Applications VM. These VMs are vulnerable to a variety of attacks and relatively easy to break into in their default configurations. The teams were allowed make some modifications to the systems to lock them down, for example changing the trivial, easy to guess passwords, however they still had to allow access to the vulnerable systems to some degree. This gave the other teams a reasonable chance to exploit some service on one of the six potential target machines on the lab network.

Detecting attacks was a large outcome of this lab and most teams were able to successfully detect and block various attacks against their systems. Only one team had their system fully compromised. Given additional time and resources it is likely that all 8 vulnerable VMs would have been hacked.

The biggest challenge in running these labs was the stability of the tools and the lack of adequate network infrastructure to support these types of activities. Due to physical constrains, the university is unable to provide dedicated lab facilities for this type of course, and the labs are actually conducted in a standard classroom.

Secure Software Development

The secure software development course is intended to teach students a coherent and appropriate approach for determining and implementing security within a software system. This course provides an overview of the various techniques to construct secure software. The course touches on all phases of the software development lifecycle, from requirements through deployment.

Lecture material for the Secure Software Development course parallels the material covered on the Certified Secure Software Lifecycle Professional Certification examination[2] as well as the concepts embodied in the SEI/CERT TSP-Secure[3] software development process.

Lecture material begins with a discussion of the basic concepts embodied in secure software development. Outcomes for this introductory material include a quantitative understanding of the security problem, basic terminology related to secure software, and the core security concepts. Students are shown the difference between bugs and flaws as well as given overviews of proactive and reactive security principles. Students are also given an overview of the software security touchpoints in order to provide a roadmap for the course[1] .

From this introductory material, the students are then introduced to the concept of risk management and how secure software is an attempt to mitigate software security risks. This material is based upon the ideas put forth by McGraw[1]. Students are shown through mini case studies how two different organizations may have vastly different security goals and objectives based upon their relative risk.

From risk management, the course moves into the area of requirements analysis. A taxonomy of security requirements is presented to students after which detailed examples are provided for each type. Certain areas of requirements received more in-depth treatment, as they are deemed more important for the projects the students will be working on in lab. For example, confidentiality requirements and implementation mechanisms are discussed, even though this will overlap slightly with the information security course. Based on software use cases, students are given techniques for identifying the software assets as well as mechanisms for ensuring that the assets are properly protected. Any discussion of secure software development requirements, however, would be incomplete without including a discussion of abuse / misuse cases. Students are shown how misuse case scenarios and misuse diagrams can be drawn to help refine the needs in the area of security.

After requirements, the course moves into the area of design and architecture. Students are taught the core secure design principles and shown how they can be employed in developing a secure software system. A case study on internet browser design is given, comparing from a design standpoint the initial release of the Google Chrome browser with other web browsers. When talking about design, software architecture is introduced to the students in the context of an architectural risk analysis. The architectural risk analysis leads to the concept of threat modeling.

While design is important, many good designs have been undermined by improper implementation. To this end, the course discusses the major mistakes made during implementation. Students are exposed to the Common Weakness Enumeration and OWASP Top 10 ranking for implementation mistakes. Static analysis is introduced as well, including an in-depth discussion of its capabilities and weaknesses. In class demonstrations of common implementation mistakes are also given, including buffer overflows, cross site scripting, SQL injection, and broken authentication.

Security testing is also covered as a topic. Included in this area is a lecture material on mapping abuse cases and abuse case scenarios to security test scenarios. Fuzz testing is introduced as a mechanism to perform input validation on certain systems and an extensive in class demonstration of the development of a fuzzer and fuzz testing occurs. Limited demonstrations of penetration testing conclude the testing segment.

The last topic to be addressed in the course is software deployment. Students are taught about the importance of appropriate data logging and configuration management, especially when related to patches. Students are also taught the importance of incident response and incident management through a software case study.

Lab Topics

The Secure Software Development course provides two different aspects of lecture reinforcement. In some lab sessions, students are involved in basic tutorials, learning how assorted tools work

In the first lab, students are tasked with brainstorming about a system which is under construction and asked to identify software assets and threats against the system. This is done in small groups, and at the end of the lab session, the teams each given a brief presentation on their observations. This format allows teams to compare their analysis and offer their insight to other teams. While early in the course, this lab serves to jump start student thinking about security as well as allow for introductory discussion on topics that will be covered in more depth later in the class.

After the first lab, ongoing projects are assigned to teams. Each course has a number of projects assigned based upon course enrollment. Sample projects are given in Table 2. Each project has been carefully chosen to have appropriate complexity as well as security ramifications. Thus, the second and third weeks are spent doing requirements related activities on these systems. Teams will be tasked with developing use cases, abuse cases, and developing security requirements.

The fourth week of the course is spent developing architecture for the system and developing a preliminary high level design. In this activity, students use the secure design criteria to develop a preliminary design. In a cyclical development process, this design would encompass the deliverables completed for phase one. Another manner of thinking about this design is that this is a design for the prototype proof of concept which verifies the viability of the project.

The next lab session focuses on Threat modeling. Using the Microsoft STRIDE[4] model from Microsoft and the Microsoft SDL Threat Modeling tool, students create a threat model for their application. This serves to reinforce the design principles embodied in lecture as well as show them in a practical manner the issues that their designs have created.

When the design is completed, students are to begin proof of concept implementation on their systems. In short, they are to pick a subset of the requirements for the system and implement a proof of concept implementation. This is a multi-week assignment covering multiple labs.

In parallel with this implementation, students are provided a fully implemented system and an assignment to look for bugs and flaws within the system using static analysis. To perform this task, students are provided with access to the Fortify Static Analysis tool. As a team, students go through the practice of performing static analyses on multiple projects. The first project used is a very small embedded web server written in Java of approximately 500 LOC. It is small enough that the students can readily understand its implementation and do a complete analysis on the code. Following this, students are provided with a link to an open source project that needs to be analyzed, and the same activity is performed except on a larger scale project.

While the students are implementing code, another lab session focuses on the Hackme Casino. Students are provided with a tutorial on how the casino can be hacked into to reinforce in an active environment the vulnerabilities that may be present in a given software implementation.

During the final lab session of the course, students are to make an oral presentation on their project, discussing all aspects of security in a design walkthrough fashion. Other students are encouraged to challenge their design assumptions and implementation decisions. The presentation ends with a demonstration of the projects.

Table 2 Sample Secure Software Development Projects

Name	Short Project Description
A Lecture Markup System	With modern tablet machines becoming readily available and wireless internet also becoming available, there is a need for a more advanced interactive solution. Thus, the reason for the Lecture markup system. With the lecture markup system, students will be able to follow a lecture online on their PC, interacting with the professor through multiple choice questions, drawing pictures, and free form typing. In an interactive situation, the professor may view the student's submission and choose to dynamically incorporate a solution to a problem into one of his / her lectures. A good lecture system will also allow the student to annotate the professor's lectures as they attend the lecture. The system would automatically be synced to a recording of the lecture.
A Distributed UML Diagram Tool	The software tools that we have for software design often are not conducive to operating in a distributed environment. For this project, a client desires to have a UML system developed which will allow a distributed team to draw in a pseudo-whiteboard environment UML class diagrams and sequence diagrams. Diagrams will be kept under version control so that a person can revert to a previous version if necessary. Additionally, diagrams will be kept in a manner to allow a person to show the progression between one version and another version as well as highlight the differences between two different versions.
Assignment Assessor	Each quarter, students submit assignments. This typically occurs through Blackboard for many courses. Blackboard, however, has many undesirable features. To alleviate this problem, an assignment system is being proposed which would be online. It would allow students to submit artifacts in either word or pdf format and would automatically convert them into a pdf format. The professor will then be able to markup the assignments with hand written comments as well as score the assignment based upon a multi-dimensional scoring rubric. When completed, both the marked up assignment and the grading rubric will be visible to the student. The professor will be able to run assessment reports on the submitted assignments. For example, "how many students achieved a 3 or 4 on a given skill set on a given assignment" or "How much improvement did student X show in his / her writing skill from the first submission to the last submission?" The professor may also need to export the grading rubrics into a non-online format for archival.
Mobile Phone Weather Warning System	Dangerous weather occurs all the time. For the frequent traveler, this can be problematic, as in many cases, one is traveling in an area of the country that they are not familiar with. Thus the need for a Mobile Phone Early Warning System. The Mobile Phone Early Warning System will use GPS tracking to keep track of where a traveler actually is at in the country. While doing this, it will also monitor the National Weather service pages to determine if a watch or warning is active for the area in which the phone is located. If a watch occurs, the user will be informed of the watch as well as given updates. If a warning occurs, the system will immediately warn the user of the danger as well as inform them of appropriate safety activities. When the warning is cleared, the system will remind the user to check in with the system to indicate that they are unharmed. If they fail to check in, a notice will be provided giving their last known GPS coordinate. Family members may check in on the status of others who are traveling. If a spouse is traveling in a warning area, the system will automatically send an OK message to others when the user of the system checks in to indicate they are unharmed.
Professor Box	Professors often have multiple machines, and many of us use the commercial service DropBox to synchronize files. This works OK, but is not perfect. DropBox has issues with synchronizing files that do not need to be synchronized, failing to synchronize files, and other issues. For this project, a software system similar to DropBox is to be developed which allows files to be automatically synchronized on multiple machines. However, unlike DropBox, prior to automatically synchronizing new files, the tool must notify the user that a new file has been detected and ask if synchronization is requested. The system must protect against the propagation of viruses from one machine to another and in completed form should be portable across operating systems. Additionally, since the system is to be used in an Academic setting, it must be FERPA compliant in its implementation.
Project Op Score	Each fall, MSOE hosts the Op competition for high school students. In this competition, students compete to solve programming problems. Each year, however, scoring is very difficult. Currently, scoring is handled using a 15 year old UNIX application and file submission is handled using a set of shared drives. For this project, a system is to be developed which ideally integrates as a simple plug in into Eclipse and allows a student to submit a solution to a problem. This will then place that solution into a designated location and notify the judges that the solution is complete. The judges will then be able to pull that solution into their edition of eclipse, run it, and report back the score (passed / failed, and comments.) The system must be protected so that teams must log in and judges must log in as well. The system must only allow submissions of .java and .c /cpp files, and must protect against the propagation of viruses from the competitors machines to the judges machines.

Independent Learning

A core part of secure software development is independent learning by students. In a typical class, independent learning is accomplished by requiring students to complete a research paper on a specific topic. A research paper involves obtaining significant depth in an area as well as a significant investment of time by the students.

For secure software development, students are instead expected to complete a number of article summaries. In an article summary, students are expected to read an article and provide a brief, 1-2 page summary of its content and what they specifically learned. To prevent duplication, only one student is permitted to summarize any given article. While students are free to choose their specific article, they are required to complete one summary from the IEEE Security and Privacy Magazine, one summary from a major IEEE computer security conference, and one summary from the Silver Bullet Security Podcast[5] . This format allows students to be exposed to different aspects of software security as well as different communication formats. It also helps to instill the importance of independent learning and continuous learning within the enrolled students.

Information Security

The information security course is taught as a lecture only course, with no associated lab. Three hours of lecture are given per week.

The course begins with introductory material related to information security, including information about the principles of information security and the principles for security mechanisms. This then leads into a brief discussion on the practical usage of security concepts in software systems. This then leads into some of the legal aspects of information security. In depth coverage is given through a case study of the HIPAA legislation.

With this background provided to students, the course then address cryptography in depth. Extensive coverage is provided on the basics of cryptography, secret key cryptography, hashing, and public key cryptography. These topics are supplemented with discussions of the practical usage of cryptography, including UNIX authentication, PGP, and Kerberos.

Independent Learning

In addition to the required course textbook, there is a second textbook addressing the ethical and social aspects of computer security. During each lecture, students are given a reading assignment for the following class. During the following class, a set of students are tasked with leading a 15 – 20 minute discussion on the ethical and societal aspects of the reading.

Students are also required to perform their own independent research on a topic of interest with a partner. The topic is up to the students to choose, though in most cases, the material comes from one or two primary sources. Student selected topics generally fall into the categories of the application of security to specific systems, best practices for security, and other associated items.

Challenges in offering the three courses

Being a smaller institution with severe faculty limitations, there are many challenges to offering such a course sequence in a consistent manner.

The first challenge to offering these courses deals with pre-requisite knowledge. While the goal of these courses was to offer them as a three course sequence, the courses also had to be designed to stand independently. This is due to the need for students to complete a significant number of technical electives outside of the application domain sequence. Thus, for any given offering of one of these courses, more than half of the students may be taking that course as a technical elective without any plan of taking the other two courses in the sequence. This problem is magnified even more by the fact that students of both the software engineering and computer engineering programs routinely take these courses as technical electives. This makes it more challenging to offer these courses in a neutral manner.

Scheduling is also an issue with these courses. Because of the small size of the programs, these courses are only offered every other year, resulting in a mixture of both junior and senior students taking the courses. This results in significant differences in the level of knowledge. For example, junior students taking secure software development are concurrently taking software architecture while seniors have already had this course material. These situations lead to vastly different understandings of the concepts of software architecture and in the performance of architecture risk analysis.

Network security also faces similar problems. Currently the software engineering curriculum does not require software engineering students to take a computer networking course. Students who wish to take a networking course can take it as a technical elective in the spring quarter of their senior year. However, this prerequisite material cannot be used in network security, offered in the fall quarter, requiring a significant amount of time to be spent on introductory topics.

Facilities needs are also challenging for these courses. As a group, these courses do not have dedicated laboratory space for their offerings. This is especially problematic for network security, as it is not possible for students to experiment in a “safe” networking environment which is protected from outside entities. Network connectivity also poses issues. The lab space typically used for these courses is not equipped with wired network connections, instead relying on 802.11g wireless networking. This poses significant connectivity issues when all students in a class are attempting to install a large binary for a commercial grade security tool.

Software tool support is also a challenge for these courses. At this institution, students are provided a laptop through a technology fee. They are issued new laptops in their freshman year and a replacement laptop in their junior year. However, the machines and the images installed upon them are not necessarily consistent from year to year or student machine to student machine. In previous offerings of the Secure Software Development course, for example, the instructor was required to support 4 different Windows configurations (2 different releases from Microsoft in both 32 and 64 bit flavors) for all software packages used in the course. Furthermore, because of the sporadic nature of these courses being offered, the supporting tools often undergo significant revision between offerings, requiring significant modification to tools

tutorials and lab instructions. The network security course has avoided some of these issues by using virtual machines for tools. However, that is not a viable option for all courses.

Student feedback and assessment

Since the first offering of the secure software courses, students have been very enthusiastic about enrolling in these courses. Each year the junior class is surveyed as to their desired electives, and each year, these courses have been high on their list of desirable elective courses. Overall, aside from gaming and embedded systems, the Software Security Application domain has also been one of the most popular application domain sequences for students to enroll.

Student comments have generally been favorable to all three of the courses, and all three courses have often been overenrolled due to student demand. Sample student comments are provided in Figure 3.

Quantitative assessments have also been collected for the offerings of the secure software development course, and it also ranks favorably. This information is shown in Table 4.

Conclusion

This article has provided information into the specifics of the Software Security Application domain offered at the Milwaukee School of Engineering. Students have viewed these courses very favorably, and the faculty members have enjoyed teaching these courses, even though there are significant constrains to teaching these courses in a quality fashion.

Bibliography:

1. Gary McGraw. 2006. *Software Security: Building Security in*. Addison-Wesley Professional.

2. 2010. *2010 Certified Secure Software Lifecycle Professional Candidate Information Bulletin*. International Information Systems Security Certification Consortium, Inc., (ISC)²

3. Noopur Davis. Secure Software Development Life Cycle Processes: A Technology Scouting Report, Software Engineering Institute, December 2005, CMU/SEI-2005-TN-024.

4. Shawn Hernan and Scott Lambert and Tomasz Ostwald and Adam Shostack. Uncover Security Design Flaws Using The STRIDE Approach MSDN Magazine, November, 2006.

5. Silver Bullet Security Podcast, Sponsored by IEEE Security and Privacy and Cigital, http://www.cigital.com/silver-bullet/.

Table 3 Sample student comments.

Course	Comment
Information Security	• I enjoyed this class but I was a little disappointed that it focused so much attention on cryptography algorithms rather than implementations and uses. I felt that more focus on the techniques of how to write secure applications would have been more appropriate. No big deal, just something that I remembered and figured it might help. • Sometimes hard to keep up with pace of lectures, especially when extra details about lecture being talked about. • I liked all of the discussion on RSA and the examples. It's always nice to see those. The cookies were awesome. • I still believe too much time is taken on discussing the Schnier book. Rather, I feel we are too crunched to learn the actual course content. • The reading by Scheier added a broader view of security, more than just encryption.

Secure Software Development	• This class was very enjoyable and interesting.
	• This course was very interesting ad I wouldn't change anything.
	• Labs were fun and educational.
	• Labs were interesting and varied.
	• Effective labs! They helped a lot in understanding security vulnerabilities and how to find them.
	• The static analysis tool used in this class is one of the best tools used in any of our software engineering courses. I learned a lot from it usage and found it to be very easy to learn and use.
	• Hackme casino fun and good method for learning the material
	• Did care for the Microsoft tools lab. Too broad of scope for the course.
	• Need to watch the number of lab specific tools. Laptops very full and didn't have space to install them.
	• Virtual machine with all tools might be nice.
	• I really liked the exposure to commercial software given by the Fortify tool.
	  ○ Overall, course was easier relative to other courses and other electives.
	  ○ Course was balanced, but overall would have preferred a bit tougher course with more in depth coverage.
	  ○ Course must be balanced based on a pp domain such that all three courses cover required material.

Network Security Tools and Practices	• Lots if background material and a wide variety of interesting types • Good overall course, maybe could be expanded into 4 credits and go more in depth. • Good class - liked the hands on aspects (ARP demo was cool) • Interesting demos and labs. Made everything understandable. • Liked the demos • This course was very enjoyable and I feel like I learned a lot. • Fun course, learned a lot, wish we could see more classes like this. Final group project was a lot of fun. • Lecture and labs were both very interesting, lots of good stuff. • The professor showed enthusiasm for this class. It was a very interesting course, and I learned a lot about different security tools that can be used to find exploits in a network in order to fix them. The professor got students involved through class activities where we would work in teams and work on the topic for the day. The labs were very hands on, and we learned a lot from them. The final lab was very cool. Building our own little networks, and trying to defend them was a lot of fun, and we learned a lot about the capabilities of different tools we can use to detect attacks. • Tool of the Week was very helpful in learning about different tools each week. • Demos in class were interesting and useful.

Table 4 Quantitative assessment for SE4930 Developing Secure Software.

Each statement was assessed on a 5 point Likert scale, with "Strongly Agree" representing a 5, "Agree" representing a 4, "Ambivalent" representing a 3, "Disagree" representing a 2, and "Strongly Disagree" representing a 1.

		2009-2010				2011-2012			
		Average	Median	Stdev	4/5 percent	Average	Median	Stdev	4/5 percent
General Course	I felt that the paper reviews helped me to independently learn new material that I would not have learned otherwise.	3.64	4.00	1.12	63.6%	3.81	4.00	1.03	66.7%
	I would highly recommend this course for future students should it be offered again.	4.18	4.00	0.40	100.0%	3.76	4.00	0.83	61.9%
Lab Projects	This lab was an excellent tool for teaching the concepts of this course. I learned more by doing this type of project than if each lab had been a separate entity.	4.00	4.00	0.63	81.8%	3.81	4.00	0.75	81.0%
	I learned a lot from this lab sequence.	4.00	4.00	0.89	81.8%	3.67	4.00	0.73	71.4%
	The project based approach should be continued for future students.	3.73	4.00	0.90	63.6%	4.05	4.00	1.02	81.0%
Hackme Casino / Bookshop Lab	This lab was an excellent tool for teaching the security aspects covered by this lab.	4.36	4.00	0.50	100.0%	4.24	4.00	0.83	85.7%
	I learned a lot from this lab.	4.18	4.00	0.75	81.8%	3.95	4.00	0.92	76.2%
	This lab should be continued for future students.	4.45	4.00	0.52	100.0%	4.05	4.00	0.92	81.0%
Fortify Static Analysis Lab	This lab was an excellent tool for teaching about static analysis and its capabilities.	3.91	4.00	0.83	81.8%	4.24	4.00	0.70	85.7%
	I learned a lot from this lab.	3.55	4.00	0.52	54.5%	4.00	4.00	0.77	81.0%
	This lab should be continued for future students.	4.00	4.00	0.89	81.8%	4.24	4.00	0.62	90.5%