Introducing a Cybersecurity Mindset into Software Engineering Undergraduate Courses

Ingrid A. Buckley, Janusz Zalewski
Department of Software Engineering
Florida Gulf Coast University
Fort Myers, FL, USA

Peter J. Clarke
School of Computing and Information Sciences
College of Engineering and Computing
Florida International University
Miami, FL, USA

***Abstract*—Cybersecurity is a growing problem globally. Software helps to drive and optimize businesses in every aspect of modern life. Software systems have been under continued attacks by malicious entities, and in some cases, the consequences have been catastrophic. In order to tackle this pervasive problem, emphasis has been placed on educating software developers on how to develop secure systems. The majority of attacks on software systems have been largely due to negligence, lack of education, or incorrect application of cybersecurity defenses. As a result, there is a movement to increase cybersecurity education at all levels: novice, intermediate and expert. At the college level, students can be exposed to cybersecurity skills and principles that will better equip them as they transition into the workforce. A case study is presented which assesses the cybersecurity knowledge of juniors and seniors in a software engineering degree program taught over a one-semester period.**

Keywords*—Cybersecurity; security education, software testing; computer security; defect detection, software maintenance

I. Introduction

Software continues to impact all aspects of our lives, including the way we use our phones, computers, home appliances, medical devices, and cars, just to name a few. Cybersecurity has been essential in the development of software due to the continued attacks and exploitation techniques that are performed by malicious entities over the Internet. Due to the ubiquitous nature of software, there is a great demand for skilled software developers.

Cybersecurity is an important element of software development and is an essential process to help prevent or reduce defects and vulnerabilities that can be exploited. Software vulnerabilities and defects have caused significant losses and inconveniences when systems fail or are exploited by hackers across different domains such as health care, financial, government, telecommunications and transportation systems. In general, software developers, testers and programmers are not experts on security. They implement systems that are not equipped to defend against cyber-attacks as they tend to only focus on ensuring that requirements have been adequately implemented. From a business point of view, the cost of cyberattacks are high; they increase maintenance costs, negatively impact customer perception of a product and lead to loss in profits.

This material is based upon work supported by the National Science Foundation under Grant number DUE-1562773.

However, programmers are now expected to consider threats and vulnerabilities, and to implement applications and programs that cannot be easily attacked or exploited. This is especially true for students who are not yet experienced in software development, or in cybersecurity. This lack of cybersecurity knowledge is a major issue in software development. It has been proven over the years that, software defects account for huge losses [1]-[3] and rework when security is not considered or poorly implemented. At the course level, it is important to motivate students to take a responsible approach to software development by teaching them how to test with the basic goal of evaluating and identifying defects [4].

Due to our reliance on software, there is a great need to educate and equip students with effective cybersecurity skills and knowledge. In this paper, a study is conducted to find an effective approach to expose undergraduate students to security principles. The goal of this exercise is to determine how well students can evaluate control structures by determining the correct output and, identifying defects. The specific objective of this paper is to determine how to increase cybersecurity knowledge of novice software developers which include university juniors and seniors in programming focused courses. The rest of the paper proceeds as follows. Sections 2 presents related work. Section 3 presents the case study and an evaluation of the students' performance. Section 4 discusses future work and Section 5 concludes the paper.

II. Related Work

Due to the urgency to increase cybersecurity awareness, skills and knowledge worldwide, colleges and universities, in particular, have implemented a variety of efforts to teach students about cybersecurity in software development and programming. Chen [5] proposed a teaching tool, called SWEET (Secure Web Development Teaching), for undergraduate and graduate computing courses. SWEET features virtualized web servers and a platform that allows instructors to teach security issues in web application development within undergraduate and graduate courses. This project included a laboratory exercise where students learned how to create a self-signed web server certificate. The goal of this exercise is to guide students on how to create a public and private key pair, a Secure Socket Layer (SSL) certificate and a certificate signing request (CSR). In the security exercise given in this study, students are not developing or creating

something new. Instead, they focus on assessing existing code and identifying defects that may be already existing. This provides an alternative way of learning and considering security by assessing existing code.

Similarly, Scheffler [6] designed two projects that use real world scenarios within public key infrastructure and web of trust modeling. They used several secure cryptographic algorithms that were assigned to students for implementation. The objective was to expose and teach students how to implement cryptography concepts in real world applications. Scheffler's work focuses on developing security based application from inception, whereas, the security exercises used in this paper focus on students' evaluation of existing code to uncover defects or defects in its logic.

Peltsverger [7] developed a bottleneck analysis lab with virtual network emulation environment. The lab consists of real work practical exercises using NetKit. The lab is designed to teach students how to set up a virtual network, capture traffic and analyze system performance. The lab exercises reinforced lectures and helped students to better understand computer network security concepts and challenges. Peltsverger's approach is similar to the security exercises utilized in this paper, in that it allowed students to analyze the outcome by reviewing the system performance. In this work, students analyze existing code manually and determine what the correct output should be given a specific input.

Chi et al. [8] implemented modules for teaching secure coding practices to STEM students. The modules were designed to provide fundamental secure programming skills to programmers and application developers. They used static-analysis tools to help with detecting vulnerabilities such as buffer overflows in code. Their aim was to increase security awareness by exposing a variety of students from different STEM disciplines to security principles, techniques and tools. The work by Chi et al. work is similar to the work completed in this study, except that they utilized tools to detect or uncover vulnerabilities in the code. In security exercises in this study, students analyze small code blocks manually to identify defects and determine the correct output given a set input.

Kumaraguru et al. [9] developed a system and game to teach users about phishing to help them make better trust decisions. They developed an email-based anti-phishing system called "PhishGuru", and an online game called "Anti-Phishing Phil", that teaches users how to use cues in uniform resource locators (URLs) to avoid falling for phishing attacks. The results from the PhishGuru studies suggest that the current practice of sending out security notices is ineffective. However, hands-on training can effectively teach people how to avoid phishing attacks. Similarly, the Anti-Phishing Phil exercise demonstrated that participants who played the game performed better at identifying phishing Web sites. Kumaraguru et al. used gamification to educate users about how to avoid phishing attacks. The security exercises in this study are geared towards students who will have to either develop, repurpose or maintain existing software. As a result, the exercise in this study focuses more on assessing existing code to determine defects that can be exploited by a hacker.

III. Security Case Study

In this section a description is given of the security case study completed in two software engineering courses consisting of university juniors and seniors. The primary objectives of this study are to assess (a) the overall cybersecurity knowledge of students, and their (b) ability to identify faults and defects and (c) aptitude to evaluate existing code.

A. Preliminary Work

Buckley [4] proposed a teaching strategy which leverages the use of basic data structures to teach the fundamentals of software testing principles. Software testing is an important phase in implementing secure code. In this approach, students must first understand the fundamental properties and constraints of various data structures and a recursive problem. The idea is to encourage students to fully understand the core properties and constrains of a system; this is analogous to understanding the security requirements of a system. This aspect is imperative in order to write effective test cases to uncover faults and defects. In this project, students are given the exercises to write test cases that ensure that each data structure's properties and constraints are upheld throughout implementation to avoid defects and faults that can be exploited in the future. The initial material which sparked the idea for this project is presented in Table I.

TABLE I. Data Collected from Software Testing Studens in Spring 2016

	Pre/Post-test correct responses											
	Ques. 1	Ques. 2	Ques. 3	Ques. 4	Ques. 5	Ques. 6	Ques. 7	Ques. 8	Ques. 9	Ques. 10	Average	**Std. Dev.**
Pre	63%	67%	86%	53%	55%	16%	84%	53%	61%	31%	**57%**	**0.354**
Post	79%	98%	94%	81%	65%	40%	88%	56%	94%	46%	**74%**	**0.144**

Forty nine (49) students completed the pre-test, while forty eight (48) completed the post-test. Overall, the results of the study showed a significant improvement in the post-test rate (74%) versus (57%) with standard errors of 14.4% and 35.4%, respectively. The average post-test results show a 30% improvement over the average pre-test results. The variation in the proportion of correctly answered questions decreased by 59%; i.e. from 35.4% to 14.4%.

B. Student Background and Aptitude

This case study includes university juniors and seniors who are completing a software engineering degree program. The juniors were enrolled in a data structures and algorithm course which is offered in the spring semester of their junior year. The seniors were enrolled in a software testing course taken in the final semester of their degree program. All the students involved in this study completed programming courses using Java, C and/or C++ in prior semesters. In the data structures and algorithms course, the students are taught different data structures (stacks, queues, binary trees, linked list, etc.) and how to determine the efficiency of algorithms (Big O notation).

In software testing, the students are taught various testing techniques including unit, integration, systems, regression and acceptance testing. They are taught blackbox and whitebox testing techniques, and utilize statement and branch coverage tools. All seniors in this study had already completed data structures and algorithms the previous spring. Additionally, most of the seniors had completed at least one internship experience that involves programming, testing or some other aspect of software development.

C. Cybersecuity Pretest and Posttest

The pretest and posttest were designed to assess students' knowledge of inspecting and evaluating small blocks of code. This exercise challenges students to carefully evaluate code to find faults and defects by determining what the expected output should be. The objective of this knowledge area is to increase software quality by discovering and correcting faults and defects that can be exploited via cyberattacks. Additionally, it illustrates to students how bad programming habits or confusing code can lead to vulnerabilities and defects that are exploitable.

The questions on the pretest and posttest are based on scenarios that provide some hands-on relatable examples that will challenge students to carefully examine basic code that may have defects infused in inconspicuous areas of the code. Each scenario is accompanied by a flow chart to further illustrate the logic as shown in Fig. 1 and 2. The pretest and posttest consist of 11 questions based on two different scenarios. The students were given the pretest at the beginning of the course; they were also given the same test at the end of each respective course. The problem scenarios are summarized below:

- Scenario1 - test a method that takes input as a decimal number and returns a string of "pass" or "fail". Assume that a grade of 70 or higher leads to a "pass", and a grade below 70 leads to "fail"'. All valid grades fall into the range of [0, 100]; otherwise, a grade leads to "fail".
- Scenario2 - test a method that takes input as a decimal number and returns a string of a letter grade based on the grade scale in Table II and Fig. 2.

TABLE II. GRADING SCALE

Grades	Return
90-100	"A"
80-below 90	"B"
70-below 80	"C"
60-below 70	"D"
0-below 60	"F"

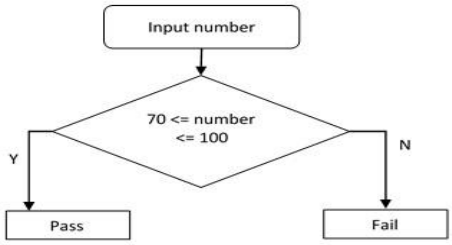

Fig. 1. Scenario 1 flow chart.

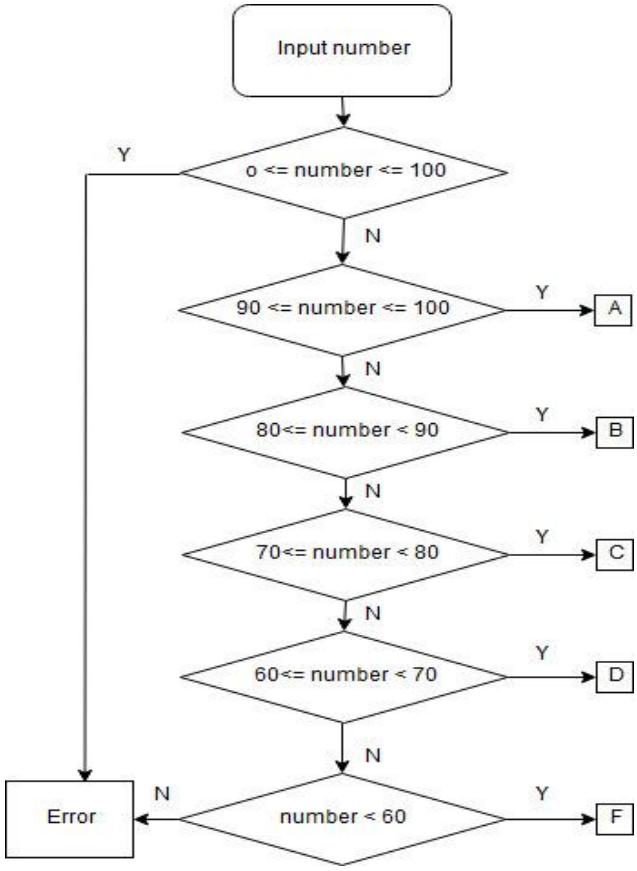

Fig. 2. Scenario 2 flow chart.

D. Evaluation of Results

A total of sixty five (65) students completed the pretest and posttest, which included twenty nine (29) juniors and thirty six (36) seniors. The juniors and seniors were enrolled in programming centric courses, namely data structures and algorithms course and software testing respectively. Overall, the results in Table III illustrate that there is a 21% increase in the mean score of the juniors versus 13% for seniors. There is a 20% increase in the median score of the juniors versus 14% for the seniors. There is a 30% increase in the standard deviation score of the juniors versus a 1.34% decrease for seniors. Both groups obtained the same posttest score which is roughly (8/11), even though the juniors had lower pretest scores.

A more detailed statistical analysis of the results show that there is significant difference between the pretest and posttest for both the juniors and seniors in Table III. Using a paired sample t-test the results for the juniors are $t(26) = 5.51$, $p < 0.01$, which shows significance, and for the seniors the results are $t(35) = 4.12$, $p < 0.01$. Note that there was no significant difference on the pretest between juniors and seniors based on the equality test of variances.

E. Discussion

Overall, both juniors and seniors scored comparably on the posttest. However, juniors achieved a higher percentage of improvement between the pretest and posttest; that is, seniors achieved a lower percentage improvement. Since seniors

typically have more development experience and knowledge than juniors, they performed slightly better on the pretest. Despite having two different proficiency levels, both groups showed an improvement in their abilities to detect defects, faults and to determine correct output.

TABLE III. COMPARISON PERFORMANCE BETWEEN JUNIORS AND SENIORS GROUP

	Juniors			Seniors		
	Pretest	**Posttest**	**% Change**	**Pretest**	**Posttest**	**% Change**
Mean:	6.68	8.07	20.81	7.02	7.91	12.68
Median:	6.67	8.0	19.94	7.0	8.0	14.29
Std. Dev:	0.88	1.15	30.47	0.97	0.96	-1.52
The maximum score on this exercise is 11.						

Even though the problem scenarios used were basic familiar exercises, the majority of students were not able to answer all of the questions correctly. The exercises were designed to test each student's ability to thoroughly understand the code. Only three (2 seniors and 1 junior) of the sixty five (65) students who completed the exercise answered 90% of the questions correctly on the posttest. Even though the majority of students' scores improved between pretest and posttest, only 4.6% were able to identify the correct output and the majority of the faults.

Threats to Validity: One of the main threats to validity is the different educational levels of the students, it is expected that students in their senior year would have been exposed to the type of problems in the pretest more often than the juniors. This fact is shown in the better performance by the seniors in the pretest. Given that the sample was not randomly selected from the entire student population, it would be difficult to make a generalization based on the students' performance. In addition, it may be a stretch to claim that the sample questions used in the pre and posttest is reflective of the total skill set associated with cybersecurity concepts.

IV. FUTURE WORK

The focus of this study is to encourage students to evaluate existing code with the aim of identifying faults, defects, and assessing their understanding of existing code. This exercise is important, primarily because testing and maintenance are crucial aspects in the software development life cycle; it teaches students to refine their skills on how to approach testing and modification of existing code. Given the results of this preliminary study, another study will be undertaken which considers the aptitude level, grade point average (GPA), programming skill level, knowledge, and experience of each student participating in the study. This additional data provide a benchmark of where students are in their knowledge and skill level. It will also allow for a richer evaluation of their performance, knowledge gain and challenges or obstacles that impact their skill set and knowledge. Additionally, this data will help in identifying what factors and prerequisite knowledge contribute most in preparing or aiding students to better understand existing code with the aim of identify faults and defects.

We also plan to perform additional studies using the Software Engineering and Programming Cyberlearning Environment (SEP-CyLE) [10] that contains cybersecurity learning content. The learning content is in the form of digital learning objects (LOs) and tool tutorials. A learning object (LO) is a module of content that usually requires 2 to 15 minutes for completion, is self-contained, interactive, reusable and can be aggregated [11]. SEP-CyLE also supports embedded learning and engagement strategies that motivate students to interact with SEP-CyLE and access the learning content. The learning and engagement strategies include: collaborative learning, gamification, and social interaction [11].

With the use of SEP-CyLE, a comprehensive assessment of a student's cybersecurity knowledge and expertise can be designed. In that, students will complete a variety of cybersecurity focused learning objects (LOs) and the following data can be collected about a student's learning tendencies such as the (i) time taken to complete a LO, (ii) number of LOs attempted, (iii) number of LOs passed, (iv) number of LOs failed, and (v) total number of virtual points gained. SEP-CyLE has been adopted and used in various studies [11] as an effective supplemental tool and resource that supports students learning and instruction.

V. CONCLUSION

The study presented in this paper concentrated primarily on detection and evaluation, which are fundamental in achieving a secure system. The ability to detect and correct faults and defects is an important skillset that is essential for software developers and testers to acquire. In light of this fact, the exercises were deliberately given to seniors and juniors who were enrolled in software development focused courses. The results showed that juniors achieved a higher percentage improvement between the pretest and posttest, while seniors showed a lower percentage improvement. Both juniors and seniors scored comparably on the posttest and showed improvement in their abilities to detect bugs, faults and determine correct output. Additionally, only 4.6% of students answered 90% of the questions correctly. Although the exercises are simple, the results show that there is value in integrating security knowledge and practical skills in select courses. This exercise shows that a student's knowledge of security can influence the quality of the programs and systems they develop.

ACKNOWLEDGMENT

This work was supported in part by the 2016 Cybersecurity Summer Research and Training for College Faculty, led by Dr. Nasir Memon at New York University, Tandon School of Engineering Computer Science and Engineering Department.

REFERENCES

[1] W. Du, A.P. Mathur, "Categorization of Software Errors that led to Security Breaches", In Proceedings of 21st NIST-NCSC National Information Systems Security Conference, Arlington, Virginia, October 5-8, 1998, pp. 392-407.

[2] R. Telang and S. Wattal, "An Empirical Analysis of the Impact of Software Vulnerability Announcements on Firm Stock Price", In

Proceedings of IEEE Transaction on Software Engineering, Vol.. 33, No. 8, pp. 544-557, August 2007.

[3] Symantec Corporaton, "Internet Security Threat Report", Vol. 21, Mountain View, Calif., April 2016. Last Accessed: June 26, 2018, https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf

[4] I. A. Buckley and W. S. Buckley, "Teaching Software Testing using Data Structures", International Journal of Advanced Computer Science and Applications (IJACSA), Vol 8, No. 3; 2017.

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

[6] P. Scheffler, M. Hylkema, A. Temkin, "Putting It All Together: Theory and Practice in Courses on Cryptography", In Proceedings of the 14th Colloquium for Information Systems Security Education, Baltimore, Maryland June 7 - 9, 2010.

[7] S. Peltsverger, C. Zhang, " Bottleneck analysis with NetKit: teaching information security with hands-on labs", In *Proceedings of the 15th Annual Conference on Information technology education* (SIGITE '14). ACM, New York, NY, USA, 45-50, 2014.

[8] H Chi, E. L. Jones, and J Brown, "Teaching Secure Coding Practices to STEM Students" In *Proceedings of the 2013 on InfoSecCD '13: Information Security Curriculum Development Conference* (InfoSecCD '13). ACM, New York, NY, USA.

[9] P. Kumaraguru, S. Sheng, A. Acquisti, L. F. Cranor, and J. Hong, "Teaching Johnny not to fall for phish", *ACM Trans. Internet Technol.* 10, 2, Article 7, June 2010.

[10] R. Chang-lau and P. J. Clarke. Software engineering and programming cyberlearning environment (SEP-CyLE), 2018. Last Accessed: June 28, 2018, https://stem-cyle.cis.fiu.edu/

[11] I. A. Buckley, P. J. Clarke, "An approach to Teaching Software Testing Supported by Two Different Online Content Delivery Methods", In proceedings of 16th LACCEI International Multi-Conference for Engineering, Education, and Technology, "Innovation in Education and Inclusion" Lima, Peru July 18 – 20, 2018 (to appear).