Pedagogical Assessment of Secure Coding in Student Programs

Dr. Saeed Al-Haj, Ohio Northern University

Dr. Saeed Al-Haj, PhD., is an Assistant Professor of Computer Science at Ohio Northern University, Ada, Ohio. He completed his Ph.D. in Computing and Informatics from the University of North Carolina Charlotte. His expertise and general interests include: Computer and Network Security; Security Analytics; Firewalls and Access Control Configuration Analytics; Computer Science Education and Cybersecurity Education. His teaching experiences include teaching Computer Science courses and labs, utilizing technology to maximize student learning process, developing curriculum and labs, and supervising undergraduate students projects.

Dr. Naeem Seliya Ph.D., Ohio Northern University

Dr. Naeem (Jim) Seliya, PhD., is an Associate Professor of Computer Science at Ohio Northern University, Ada, Ohio, USA. His key expertise and interests include Data Science (i.e., Machine Learning, Big Data Analytics, Data Mining, Deep Learning, Data Quality, Feature Engineering, etc.), Software Engineering and Systems Development, Computing Sciences Pedagogy, Assistive Technology for Persons with Disabilities and the Elderly, Cyber Security Analytics, and Interdisciplinary/Applied Data Analytics. He has published about 90 peer-reviewed technical articles in international conferences and journals. Dr. Seliya is proactive in scholastic work and computing sciences pedagogy, including grants, undergraduate research, and curriculum development. His prior professional endeavors include: Assistant (& Associate) Professor of Computer and Information Science at the University of Michigan-Dearborn; Adjunct Instructor of Computer Science and Technology at the State University of New York, Orange; and, President and Senior Software Engineer at Health Safety Technologies, LLC.

Mr. Collin Lee Kemner, Ohio Northern University

Mr. Collin Lee Kemner is a current student at Ohio Northern University. He is set to graduate with a B.S. in Computer Science in May 2019. His general expertise and interests include: IoT and Network Technologies, iOS application development, and Secure programming. He has recently published his first ASEE paper and presented at the ASEE NCS Section in March 2019 with his senior capstone team, SoT (Secure of Things).

Pedagogical Assessment of Secure Coding in Student Programs

Abstract

Students in introductory Computer Science (CS) courses are required to submit several programming assignments and/or projects. The submitted programs are largely assessed on their correctness to the given problem, and not against secure software coding practices. In our experience, student programs typically do not follow secure coding practices, making them susceptible to security problems. Given the general lack of strong emphasis on security concepts in introductory programming courses, students tend to neglect applying secure coding practices.

The goal of this "Work in Progress" is helping students to reduce vulnerabilities in their programs and eliminate coding errors. We will investigate how errors occur, how students interpret and correct these errors, develop metrics to measure how coding standards for security are used, and provide informative feedback and actionable guidelines to students and instructors. The analysis will emphasize security Knowledge Skills and Abilities (KSAs) identified within the National Initiative Cybersecurity Education (NICE) Framework [1]. A list of secure coding practices was compiled using two different resources: SEI CERT Coding Standard [2] and Open Web Application Security Project (OWASP) [3]. The selected coding practices are applicable to C++ and Java. Each secure coding practice is assigned a weight reflecting its importance and severity.

We consider a set of 43 students' programming assignments in C++ and Java, with all of them being anonymized for Personally Identifiable Information. Each assignment typically has different coding practices that are relevant, which is a result of the difference in requirements among assignments. The problem description of each assignment is analyzed to determine the applicable secure coding practices to each submitted assignment. Our quantitative analysis gives a score out of five to each secure coding practice based on the extent it was implemented: zero implies a rule is not being addressed, while five implies a rule is implemented effectively. Any score between zero and five is based on varying degrees of effectiveness. Subsequently, rules that consistently did not score high for the programs will be given to instructors as a recommended focus in relevant CS courses.

We are currently working on collecting additional student assignments and projects from different courses in different levels, e.g., CS I, CS II, Data Structures, and Software Design Patterns. The quantitative/qualitative analysis of our study have the following key outcomes: 1) Assist instructors in identifying shortcomings of expected good programming practices and secure coding practices in student programs, lending to customized lessons for the introductory programming courses in CS; 2) Bring awareness of secure coding to students in the early stages in their learning process. Many security problems are related to the lack of awareness of possible threats and vulnerabilities; and, 3) Provide feedback to students on their own program solution in terms of its structure and design. This allows students to identify problems and vulnerabilities in their coding design, and rectify the same as they move along in the CS curriculum.

1. Introduction

Cybersecurity education aims to bring the awareness of the importance of security and privacy issues to students. This will help students change how they think when they develop computer applications to consider security problems while they design and test their code. The National Initiative Cybersecurity Education (NICE) Framework [1] identifies security Knowledge Skills and Abilities (KSAs) needed in future taskforces. These KSAs provides guidelines and indicators for cybersecurity educators to be used when offering cybersecurity courses.

In today' networked world, we depend heavily on the Internet and usage of computer and mobile applications, lending to an ever-increasing need for excellent computer programmers, especially those with good secure software development skills. Therefore, graduating programmers who have proper cybersecurity instruction becomes a necessity. This can be achieved by incorporating modern security analysis tools and proper usage of secure coding practices into introductory courses. The ultimate goal is to have more applications with fewer exploits and vulnerabilities.

Pedagogy curriculums in Computer Science (CS) typically begin with a sequence of introductory courses in Computer Programming. Subsequently, students apply the freshly learned programming knowledge and skills as they further advance in the program's curriculum with courses such as Data Structures, Algorithms, Software Engineering, and Capstone Project. Part of students' assessment in such courses is to submit programming assignments and programming projects. The assessment of students' work is largely evaluated based on the correctness, and in most of the cases does pay attention to secure coding practices and security checks that students may have violated in their work.

To get more details about how students interpret and correct security related errors in their code that are result from neglecting secure coding practices, we analyzed students' programming solutions that were submitted as assignment and projects. In our work, we have collected submissions from different courses and levels. The submissions cover wide range of secure coding practices that students may have missed. The submitted programs were tested against specific set of input values and scenarios that could exploit the vulnerabilities in the tested code.

Our approach provides quantitative and qualitative analysis that can be framed to analyze a single student, a single submission or a single secure coding practice. The resulting reports provides feedbacks to be used in enhancing CS curriculum.

The rest of the paper is structured as follows: Section 2 presents a summary of key related works; Section 3 elaborates our approach to data collection, analyzing programs and scoring them based on applicable secure coding practices; Section 4 discuses and analyzes the results of this study; and, Section 5 provides conclusion and future work directions related to this paper.

2. Related Work

Cybersecurity education has gotten wider attention than before due to the fact that usage of Internet and automated solutions in never decreasing. There are several works that address the need of incorporating cybersecurity into CS curriculums.

Qian et. al. [4] addressed the needs of authentic and active pedagogical learning materials and challenges of building Secure Software Development. The presented learning approach teach security principles through hands-on companion labs based on the Open Web Application Security Project (OWASP) recommendations.

Agama et.al. [5] developed a framework that introduces security concepts and practices to STEM students through the web portal. The framework equips students with the required skills and knowledge to guard against programming errors that lead to vulnerabilities in programs and uses static-analysis for detecting weaknesses in mobile devices applications.

Williams et. al. [6] recommended secure coding topics that can be mapped to IAS Knowledge Areas, and discussed the unique challenges of teaching secure coding to beginning programmers. They also pointed out some behaviors of beginning programmers leading to insecure programs that may need the instructor's attention.

Taylor et. al. [7] presented The Summit on Education in Secure Software recommendations and the new curricular guidelines and discuss the importance and challenges of teaching secure coding. Haywood et. al. [8] discussed the importance of certain topics such as access control, error handling, and a number of tools that can be used with Java that can create security keys and policies. They developed modules that were used in programming courses to teach students security features and safe programming practices.

Our work is different as it focuses on the pedagogical assessment of students programming solutions security level. The presented approach in this work analyzes how students use secure coding practices in their assignments and projects. The quantitative and qualitative analysis provides actionable outcomes to improve cybersecurity education in CS curriculum.

3. Approach

3.1 Data Collection

To collect the data needed for this work (i.e. student programming assignments and programming projects), emails were sent to current and past students of selected courses. The emails asked for voluntary submissions of past programming assignments from classes they have already taken, such as: programming courses in both languages C++ and Java. Submissions from software design courses and data structures and algorithms were also accepted. Students who accepted the invitation were given a form of consent which stated that the data must remain confidential and that details of the programs may not be discussed outside of the project. For

clarification reasons, submissions referring to a specific assignment will be referred to as programs, submissions, or applications.

After the data was collected, the data was sanitized and anonymized; in other words, the programs were analyzed in terms of the written code and any mention of the person who submitted the program was erased and anonymized with a code name. The same was done with the program names so there is no possible way that an observer could guess who wrote the program. The submission names were not organized in a certain way; they were listed randomly so that the anonymization was as effective as possible.

3.2 Secure Coding Practices

In the world outside of programming, there are many practical ways to keep security-sensitive things from being maliciously affected. This includes security cameras, movement-sensitive lasers, and even passwords. Without a measure of security, important aspects of everyday lives could be left vulnerable. Similarly, this is why developers use secure coding practices. In software development, these practices are what help keeping malicious line of codes out and strengthening the overall reliability of the software. Such general security practices include checking for invalid input, preventing malicious commands from being executed, and encryption practices among others.

These types of practices are what this research looked at for the purpose of evaluating student submissions. The practices were compiled on a checklist so that there was easy reference to what practices/rules the project staff value the most. One source of secure coding practices we used in our analysis is the CMU-Software Engineering Institute (SEI CERT) Coding Standards for C++ and Java [2]. A selected number of practices from the website were chosen to be used for analyzing data.

The same was done for practices that were listed by the Open Web Application Security Project (OWASP) [3]. OWASP created a comprehensive list of secure coding practices related to programming in general; there is no list for specifically C++ or Java. After reviewing the OWASP list and compiling selected rules from OWASP and CERT, the stage became set for data collection before analysis could begin.

The complied list contains secure coding practices that could be applied to wide range of courses in CS curriculum such as: programming and advanced programming, data structures, algorithms, networks, operating systems, software engineering, software design, etc. In this work, we have compiled about 125 secure coding practices. Table 1 shows a subset of secure coding practices we used in this work.

Table 1: A subset of secure coding practices used in our assessment.

No.	Description
1	Validate for expected datatypes, range, and length.
2	Validate all input against a whitelist of allowed characters.
3	Normalize strings before validating them.
4	Perform any string modifications before validation.
5	Exclude un-sanitized user input from (format) strings.
6	Sanitize all output of untrusted data to operating system commands.
7	Establish and utilize standard, tested, authentication services whenever possible.
8	Enforce password complexity requirements established by policy or regulation.
9	All authentication controls should fail securely.
10	Never hard code sensitive information.
11	Limit the number of transactions a single user or device can perform in a given period of time.
12	Minimize the accessibility of classes and their members.
13	Ensure that security-sensitive methods are called with validated arguments.
14	Use error handlers that do not display debugging or stack trace information.
15	Implement generic error messages and use custom error pages.
16	Do not abruptly terminate the program.
17	Handle all exceptions thrown before main() begins executing.
18	Detect and handle memory allocation errors.
19	Detect errors when converting a string to a number.
20	Implement least privilege, restrict users to only the functionality, data, and system information that is required to perform their tasks.
21	Do not store passwords, connection strings, or other sensitive information in clear text or in any non-cryptographically secure manner on the client side.
22	Remove all unnecessary functionality and files. Also, remove test code or any functionality not intended for production, prior to deployment.
23	Explicitly initialize all your variables and other data stores, either during declaration or just before the first usage.
24	Restrict users from generating new code or altering existing code.
25	Do not use floating point variables as loop counters.

3.3 Data Analysis and Security Metrics

Each different assignment typically has different secure coding practices that are applicable relevant, so one assignment may have a different set of rules than the other. Therefore, every

submission description was studied carefully to determine the secure coding practices that will be assessed in this submission. After marking the applicable set of coding practices for a submission, the submitted program would be run and tested for any vulnerabilities. Once errors are found by running the program, the source of the errors in the code would be analyzed and any piece of code that relates to relevant coding practices will be checked.

To quantitatively analyze students' submissions, each coding practice would then be given a score out of five based on how well/the extent that the rules were implemented in the tested program; zero being equivalent to the rule not being addressed and five being equivalent to the rule being implemented effectively. Any score in between is the rule being implemented at varying degrees of effectiveness, but may leave one or two features out that prevents it from getting a score of five. This process is repeated for all secure coding practices that could be applied on a single submission.

We used three security metric categories to assess security levels of the submitted solutions:

- *Student Based Metric*: This metric evaluates a single student.
  - It calculates the aggregate score of all applicable secure coding practices for all assignments submitted by a single student.
  - This metric helps in providing a personalized feedback for every student about his programming style and points out to her weaknesses.

- *Assignment Based Metric*: This metric evaluates a single assignment or project.
  - It calculates the score of all applicable secure coding practices and for all students who submitted this assignment.
  - This metric helps the course instructor by providing a detailed feedback on secure coding practices that students did not implement (partially or fully) in their solutions. This will help the instructor to revise their teaching materials and the assignment description.

- *Secure Coding Practice Based Metric*: This metric evaluates a single secure coding practice.
  - It calculates the score for all students and for all submissions in which this practice is applicable.
  - This metric helps in assessing specific topics that are highly relevant to a specific programming assignment.

All applicable secure coding practices in a single submission are weighted based on their importance and severity. All weights are normalized and the total sum for all weights in a single assignment is 100.

The scoring of a sample program from the initial data collected to-date is provided in Table 2. The problem of the Java program is a guessing game where a user guesses a number from 1 to 100 and the program will tell the user if their guess is too high or too low. The game ends when the number is correctly guessed and indicates how many guesses it took to win the game. Table 2 shows, for the sample program, the secure coding practices' description, whether the practice was implemented or not, and the score provided to the program on a scale of 0 to 5, as mentioned

previously. The “Practice Implemented?” has three possible options: Yes, implying the practice was fully implemented; No, implying the practice was not implemented; and Partial, implying the practice was attempted but not fully implemented. The average score for the sample program is 4.24. We note that all the secure coding practices are equally weighted in this work; however, our future work will provide weights to the different secure coding practices. Among the secure coding practices analyzed in this sample program, 73.68% were fully implemented, 10.53% were not implemented, and 15.79% were attempted but not fully implemented.

4. Results and Discussion

The aggregated scores for submissions (several solutions from different student for the same assignment) show that there are some secure coding practices that need to be focused on. These practices include:

- Input validation (checking for input errors) includes checking for integer and buffer overflow.
- Input sanitization.
- Pseudorandom number generating algorithms.
- Effective access control (of members, fields, and form objects).
- Proper variable instantiation and data (user input) retrieval.
- Error and exception handling.
- More reliable comments that do not expose the underlying design of the program.

Some of these practices were partially implemented in programs while others were consistently not present.

One finding was that input validation was not implemented often in general. In many of the programs analyzed, there was an overall lack of checking for incorrect user input, such as integer/buffer overflow and entering a wrong data type (such as entering a string into an integer field). As a result, those programs could jeopardize the system without much effort from hackers.

Related to input validation is the practice of input sanitization, which is the act of removing unwanted and/or harmful characters from user input. Input sanitization can prevent such harmful attacks such as XML and SQL injection from occurring. This practice is not seen in any of the programs analyzed, regardless of the language being used. The reason behind why sanitization is not present is because it is not a part of the curriculum being taught in introductory courses, though knowing this skill could help students in the future as it can be viewed as another input validation method.

Another interesting find relates to error/exception handling and access control. Utilization of access control modifiers and exception handling seem to be much more present in student’s Java submissions than C++. This could be a result of these practices being potentially too advanced for the first programming course as it is typically a student’s first exposure to a college programming.

Table 2: Secure coding practices and scoring for sample program

Description	Practice Implemented?	Score
Validate for expected datatypes, range, and length.	Yes	5
Validate all input against a whitelist of allowed characters.	No	0
All validation failures should result in input rejection.	Yes	5
Never hard code sensitive information.	Partial	3.5
Limit the number of transactions a single user or device can perform in a given period of time.	Yes	5
Minimize the accessibility of classes and their members.	Yes	5
Ensure that security-sensitive methods are called with validated arguments.	Partial	3
All random numbers, random file names, random GUIDs, and random strings should be generated using the cryptographic module's approved random number generator when these random values are intended so they cannot be guessed.	Yes	5
Implement generic error messages and use custom error pages.	Yes	5
Handle all exceptions.	Partial	4
Do not leak resources when handling exceptions.	Yes	5
Do not allow exceptions to expose sensitive information.	Yes	5
Remove comments in user accessible production code that may reveal backend system or other sensitive information (this does not mean no comments period).	No	0
When exceptions occur, fail securely.	Yes	5
Remove all unnecessary functionality and files. Also, remove test code or any functionality not intended for production, prior to deployment.	Yes	5
Explicitly initialize all your variables and other data stores, either during declaration or just before the first usage.	Yes	5
Restrict users from generating new code or altering existing code.	Yes	5
Generate strong random/pseudorandom numbers. Also ensure that the number generator is properly seeded.	Yes	5
Check for integer and buffer overflow as well as invalid user input.	Yes	5
	Avg. Score	**4.24**

Pseudorandom number generation was also found to be typically using insecure methods, such as *srand* () in C++ and the *Random* import class in Java. Even though these random number generating methods are seen as insecure and unreliable, they still seem to have great influence in the classroom while there are other, more reliable and secure methods available.

Reliable comments, though not necessarily pertinent to security, can help others working with the program source code understand what each function is doing inside the program. Otherwise, others may feel lost as to what the code does. The tested programs given by the volunteers in one of the selected programming courses seemed to have an adequate number of comments near the beginning of the course. As the course progressed, the number and quality of comments would diminish.

The quantitative and qualitative analysis and results of our study have the following key outcomes:

- Assisting the programming instructor in identifying shortcomings of expected good programming practices and secure coding practices in student program solutions. This will lead to more customized lessons for the introductory programming courses in the CS curriculum, which typically are critical in setting a student's approach to programming solutions. In addition, this will help in designing and creating security modules that supplement programming courses' curriculum.

- Bringing the awareness of secure coding to students in the relatively early stages in their learning process. Many of security problems are related to the lack of awareness of possible threats and vulnerabilities.

- Providing feedbacks to students on their own program solutions in terms of its structure and design, as qualified and quantified by software measurements used to establish a student programming style profile. This can allow students to identify problems and vulnerabilities in their coding design, and rectify the same as they move along the CS curriculum courses.

5. Conclusion

This "Work in Progress" investigates how security related errors occur in students' programs and how students interpret and correct these errors. We analyzed students' programs to assess how secure coding practices are implemented and used in their work. Three security metrics were developed: student based, assignment based and secure coding practice based. These metrics provide quantitative and qualitative analysis that could lead to actionable outcomes to help instructors to revise their teaching material and to help students by providing a personalized feedback about their work. We will continue this work and collect more students' programs from different levels and courses hoping to cover more secure coding practices that students can learn during their study and get the needed knowledge before working in the field.

References

[1] NIST National Initiative for Cybersecurity Education (NICE). https://www.nist.gov/itl/applied-cybersecurity/nice

[2] SEI CERT Coding Standards. https://wiki.sei.cmu.edu/confluence/display/seccode

[3] Open Web Application Security Project (OWASP). https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide

[4] Kai Qian, Dan Lo, Reza Parizi, Fan Wu, Emmanuel Agu, and Bei-Tseng Chu. "Authentic learning secure software development (SSD) in computing education". *Proceedings of the 2018 IEEE Frontiers in Education Conference* (FIE), pp. 1–9, San Jose, California, October 2018.

[5] Edward Agama and Hongmei Chi, "A framework for teaching secure coding practices to STEM students with mobile devices." *Proceeding of the 2014 ACM Southeast Regional Conference*, no. 39, pp. 1–4, Kennesaw, Georgia, 2014.

[6] Kenneth A. Williams, Xiaohong Yuan, Huiming Yu, and Kelvin Bryant, "Teaching secure coding for beginning programmers", *Journal of Computing Sciences in Colleges*, vol. 29 no.5, pp. 91-99, May 2014.

[7] Blair Taylor, Matt Bishop, Elizabeth K. Hawthorne, and Kara L. Nance, "Teaching secure coding: the myths and the realities," *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (SIGCSE'13), pp. 281–282, Denver, Colorado, March 2013.

[8] A. Haywood, H. Yu and X. Yuan, "Teaching java security to enhance cybersecurity education," *2013 Proceedings of IEEE SoutheastCon*, pp. 1-6, Jacksonville, Florida, April 2013.