The Design and Implementation of Secure Program Design Education Tool

Qinrui Yu, Jinsheng Xu, Huiming Yu and Xiaohong Yuan
Department of Computer Science, North Carolina A&T State University, Greensboro, U.S.A.

Keywords: Secure Program Design, Education Tool.

Abstract: Designing and implementing secure software is becoming very critical because various types of attacks take advantage of software vulnerabilities. In order to effectively teach students how to develop secure applications, we developed a Secure Program Design Education Tool to assist students in learning how to design and implement secure programs. This tool consists of three sections to help students understand what input flaws and buffer overflow are and how to prevent them. This tool is designed and implemented with a user friendly environment and uses examples to demonstrate the results of unsecure code and the purpose of secure program design. This tool has been used for an undergraduate Data Structures class fall 2019 in the Department of Computer Science at North Carolina A&T State University. Our experience exhibits that using this tool helped students not only better understand the concepts of input flaws and buffer overflow, but also gained significant skills to develop secure software. Students' survey and feedback reflected this tool is very valuable in their education experience. This tool could also be used in other sophomore level related courses.

1 INTRODUCTION

Software applications are widely used everywhere in today's modern society. Companies have information assets that they need to keep protected. It is critical to keep these assets safe from threats in order to avoid potential financial loss, trade secrets from being stolen, possible loss of customers, and gaining an overall negative reputation. How to design and implement secure software becomes a critical issue. In the Software Development Life Cycle, the design phase is the segment of development where the resources needed for hardware and software are recognized and the logical methods that will be used are determined. Software engineers must keep secure practices throughout the entire development (Yu, Jones, 2014). A good example of a secure design practice includes that everyone is thoroughly familiar with the design, its ins and outs, and captures possible threats identified by others (Williams, Yuan, Yu, Bryant, 2014, Yuan, Yang, Jones, Yu, Chu, 2015). Since malicious software can attack unsecure programs computer science students must be trained to keep security in mind when designing and implementing software to anticipate all possible threats and prevent attacks (Frej, 2002, IEEE, 2017).

Education is a very powerful tool in promoting secure program design. By understanding software vulnerabilities and possible threats, developers can save valuable time and create secure applications. Many education tools have been developed to aid students learning related topics. Martin Quinson and Gerald Oster of the University of Lorraine developed an education tool titled "A Teaching System to Learn Programming: the Programmer's Learning Machine" (Quinson, Gerald, 2015). The Programmer's Learning Machine is an interactive exerciser for students with little to no experience to learn programming and algorithms. This tool assists students on their program skills by proving a random programming exercise that is explained and demoed simultaneously for the student's understanding. A. Abuzaid, H. Yu, X. Yuan and B. Chu have developed a cryptographic education tool to help students understand cryptographic theory and implementation techniques (Abuzaid, Yu, Yuan, Chu, 2011).

In this paper, we present a new Secure Program Design Education Tool, and discuss the teaching experience of using this tool. The rest of the paper is organized as follows. In the next section, Data Structures course related information will be discussed. Objective and goals will be discussed in section III. In section IV design considerations of this tool will be presented. In section V implementation

Yu, Q., Xu, J., Yu, H. and Yuan, X.
The Design and Implementation of Secure Program Design Education Tool.
DOI: 10.5220/0009184003190324
In *Proceedings of the 12th International Conference on Computer Supported Education (CSEDU 2020) - Volume 1*, pages 319-324
ISBN: 978-989-758-417-6

of the Secure Program Design Education Tool will be discussed. Experimental results will be given in section VI and conclusions will be presented in section VII.

2 COMP 280 DATA STRUCTURES

Comp 280 Data Structures is a required course for undergraduate students who’s major is computer science. This course takes students that have completed, as a prerequisite, a foundational programming class to the next level of learning data structures. The course examines essential data structures (linked lists, stacks, queues, trees, balanced search trees, hash tables, and binary heap). It analyzes and implements techniques such as sorting, searching, and use of STL data structures (set, map, priority queue, stack, queue, vector, list) to solve general problems. The emphasis of the course is on building computer programs that implement essential data structures, and more importantly learn how and when to use them. The students journey through the thought process of programming efficiency and effectiveness in order to handle problems like the speed of data retrieval, storage and management. Further, they gain experience with performing the presentation and discussion of their design logic; and handle questions and feedback from their peers. In order to enhance a students’ knowledge in secure programming developement, we taught the Introduction to Secure Program Design module in COMP 280 Data Structures class. We also developed a Secure Program Design Education Tool to help students understand related topics.

3 OBJECTIVE AND GOALS

Designing and implementing secure software is a very critical issue in today’s society. This results in many new requirements for software developers. In order to prepare our students for the expectations of the Cybersecurity workforce and for them to effectively learn secure software design, we developed a Secure Program Design Education Tool for the Data Structures computer science class to achieve this goal. Objectively this tool will provide students a visual interactive tutorial, step by step demonstrations on how input flaws and buffer overflow occur, and how to prevent them, to help students grasp an improves understanding of related topics and developing secure software. This tool was designed for undergraduate Data Structures course and can be used by another courses or lectures involved with security topics. After utilizing this tool, students should understand how input flaws and buffer overflow occur, and how to prevent them by writing secure code.

4 DESIGN CONSIDERATIONS

The design considerations of Secure Program Design Education Tool are visually simplistic, user friendly and interactive, consistent and platform independence. It is implemented with JavaFX.

4.1 Visually Simplistic

Visualization is an effective technique to help students understand a subject matter of importance. Visually simplistic is necessary for an education tool because it can provide an easy way to help students understand the effects of these vulnerabilities. In this tool Java swing GUI widget toolkit is used to allow students to input data and display results. In this way, students can immediately see the results of inputting invalid data or writing data over a boundary, and learn input flaws and buffer overflow.

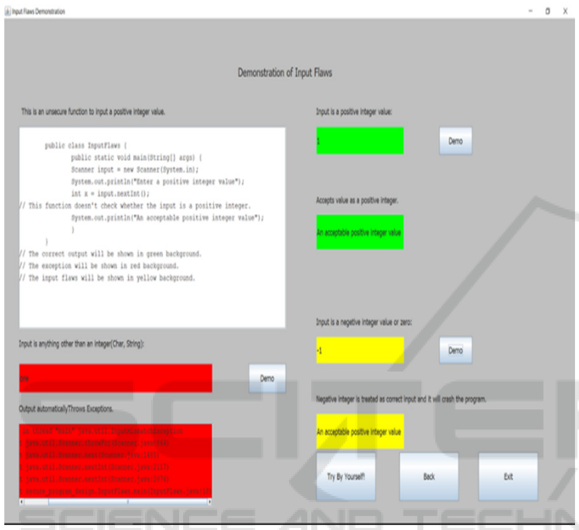
4.2 User Friendly and Interactive

User friendly and interactive is another important consideration to design the tool and make it easy for students to use it. On the input flaws demonstration page, there are several demo buttons to allow students to learn the different results of correct input, incorrect input and invalid input as shown in Figure 1. Students can click buttons to move to the next page or go back to the previous page.

Students can try to input various data by themselves in unsecure program and secure program. The tool will generate different results based on their inputs. Figure 2 is the screen that shows how unsecure program leads to input flaws. Although a correct input is a positive integer, a user can input a character, or negative number in a unsecure program. In these pages, users can still move to other pages at any time. Once a user inputs data the result will be displayed on the screen.

4.3 Consistent

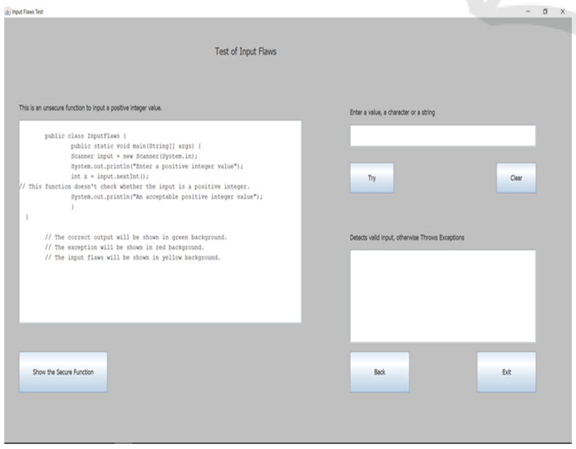
Consistent means that after the users have explored one page of the application, they can quickly know how to use other parts of the application. A consistent user interface will help students learn how to use it in an easy way. After the introduction, there is a selection page which let users select a topic of interest. Each section has two parts: demonstration and hands-on practice. In the demonstration part, students can click on the "demo" button then the results will be displayed. Green background means the input is correct, red background means the input is incorrect, yellow background means that although the input is not correct, the unsecure program treats it as a correct input that will cause input flaws. These shown in figure 1.


Input Flaws Demonstration
Demonstration of Input Flaws
This is an unsecure function to input a positive integer value.
Input is a positive integer value:
1
Demo
Accepts value as a positive integer.
An acceptable positive integer value
Input is a negative integer value or zero:
-1
Demo
Negative integer is treated as correct input and it will crash the program.
An acceptable positive integer value
Input is anything other than an integer(Char, String):
one
Demo
Output automaticallyThrows Exceptions.
Try By Yourself!
Back
Exit

```
        public class InputFlaws {
                public static void main(String[] args) {
                Scanner input = new Scanner(System.in);
                System.out.println("Enter a positive integer value");
                int x = input.nextInt();
// This function doesn't check whether the input is a positive integer.
                System.out.println("An acceptable positive integer value");
                }
        }
// The correct output will be shown in green background.
// The exception will be shown in red background.
// The input flaws will be shown in yellow background.
```

```
 in thread "main" java.util.InputMismatchException
t java.util.Scanner.throwFor(Scanner.java:864)
t java.util.Scanner.next(Scanner.java:1485)
t java.util.Scanner.nextInt(Scanner.java:2117)
t java.util.Scanner.nextInt(Scanner.java:2076)
t secure_program_design.InputFlaws.main(InputFlaws.java:18)
```

Figure 1: Demonstration screen for input flaws.


Input Flaws Test
Test of Input Flaws
This is an unsecure function to input a positive integer value.
Enter a value, a character or a string
Try
Clear
Detects valid input, otherwise Throws Exceptions
Show the Secure Function
Back
Exit

```
        public class InputFlaws {
                public static void main(String[] args) {
                Scanner input = new Scanner(System.in);
                System.out.println("Enter a positive integer value");
                int x = input.nextInt();
// This function doesn't check whether the input is a positive integer.
                System.out.println("An acceptable positive integer value");
                }
  }


        // The correct output will be shown in green background.
        // The exception will be shown in red background.
        // The input flaws will be shown in yellow background.
```

Figure 2: Unsecure program leading to input flaws.

After demonstration, students can choose to try it by themselves. They can test different inputs, learn where the vulnerability is and how to prevent it. They can also move to a different pages to see how the secure program works and prevents these vulnerabilities.

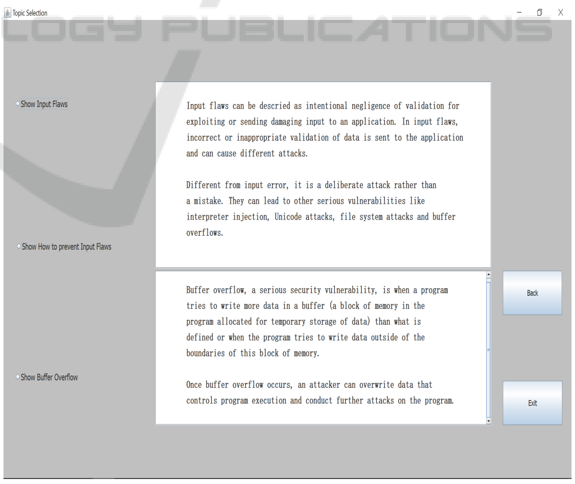
In every page, a "back" button allows students to go back to the previous page. At the end of each section, a button allows the user to select a different topic. In this way, students can select a different topic easily and find out the difference between secure and unsecure programs.

4.4 Platform Independence

The Secure Program Design Education Tool is platform independent. It is implemented by Java language with JavaFX. The .jar file can run on any platform that has Java runtime environment.

5 IMPLEMENTATION OF THE SECURE PROGRAM DESIGN EDUCATION TOOL

The Secure Program Design Education Tool includes three sections that are input flaws, buffer overflow and how to prevent them. As seen in figure 3, the introduction screen features three buttons on the left side that lead to each of the section mentioned. The right parts contain brief introductions to input flaws and buffer overflow.



Input flaws can be descried as intentional negligence of validation for exploiting or sending damaging input to an application. In input flaws, incorrect or inappropriate validation of data is sent to the application and can cause different attacks.

Different from input error, it is a deliberate attack rather than a mistake. They can lead to other serious vulnerabilities like interpreter injection, Unicode attacks, file system attacks and buffer overflows.

Buffer overflow, a serious security vulnerability, is when a program tries to write more data in a buffer (a block of memory in the program allocated for temporary storage of data) than what is defined or when the program tries to write data outside of the boundaries of this block of memory.

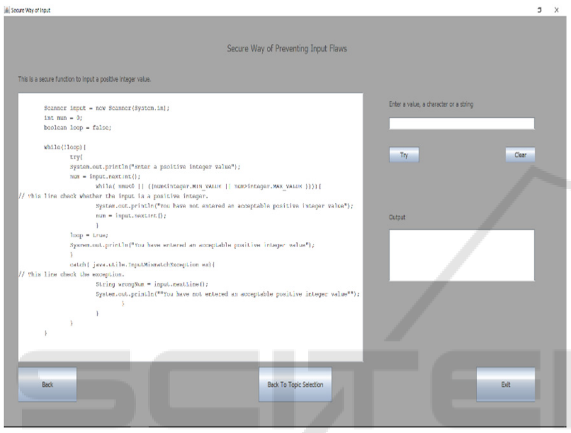
Once buffer overflow occurs, an attacker can overwrite data that controls program execution and conduct further attacks on the program.

Figure 3: Introduction Page.

5.1 Input Flaws

Input flaws results from failing to check whether the input data satisfies given requirement. Java language checks some invalid inputs such as input a string when required input is an integer. In this case an exception will occur. Java cannot check the input that is required type but does not meet certain specific requirement. For example, a program requires to input a positive integer while input is a negative integer. Once these incorrect inputs are sent to an application, they can be used by hackers to create attacks. Different from input error, input flaw is a deliberate vulnerability and can lead to serious attacks such as interpreter injection, Unicode attacks and file system attacks, etc. (Morana, Nusbaum, 2008)

This tool shows students how to modify unsecure code into secure. One way to prevent input flaws is checking all possible input types and giving out error message besides the "Throwing and Catching Exceptions" like the program in Figure 4.


Secure Way of Input
Secure Way of Preventing Input Flaws
This is a secure function to input a positive integer value.
Enter a value, a character or a string
Try
Clear
Output
Back
Back To Topic Selection
Exit

```
        Scanner input = new Scanner(System.in);
        int num = 0;
        boolean loop = false;

        while(!loop){
                try{
                system.out.println("Enter a positive integer value");
                num = input.nextInt();
                        while( num<0 || ((num<Integer.MIN_VALUE || num>Integer.MAX_VALUE ))){
// this line check whether the input is a positive integer.
                        System.out.println("You have not entered an acceptable positive integer value");
                        num = input.nextInt();
                        }
                loop = true;
                System.out.println("You have entered an acceptable positive integer value");
                }
                catch( java.util.InputMismatchException ex){
// this line check the exception.
                        String wrongNum = input.nextLine();
                        System.out.println(""You have not entered an acceptable positive integer value"");
                                }
                        }
                }
        }
```

Figure 4: Secure Program.

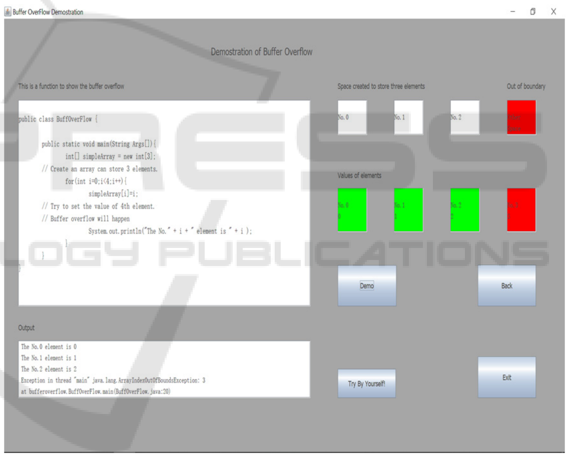
5.2 Buffer Overflow

One type of buffer overflow is over boundary. When a program tries to write more data into a buffer (a block of memory in the program allocated for temporary storage of data) than what is defined or when the program tries to write data outside of the boundaries of this block of memory (Wikipedia, 2011). Once it happens, buffer overflow will occur like the result in Figure 5. An attacker can use it to control program execution. In Java language, complier will display a "java.lang.IndexOutOfBoundsException" error message for data structures like array and linked list to stop execution when a program tries to write out of boundary. However, this will also happen when using the functions about stack and heap, which has been used as a famous kind of attack: Morris Worm.

In this education tool, an obvious type of buffer overflow is displayed. Students can learn how the misunderstanding of element number in an array causes buffer overflow, the relationship between size of the array and number of the elements to be stored in the array.

6 TEACHING EXPERIMENTAL RESULTS

The Secure Programming Design Education Tool has been used in COMP 280 Date Structures class Fall 2019 in the Department of Computer Science at NC A&T SU and shown successful results. Forty-four students attended the survey. Before distributing the tool to the students two lectures was given to introduce Secure Program Design that include Introduction, Secure Program Design Considerations, Insecure Programs, How modifying insecure code into secure program and Safe Program Design. Students used learned knowledge with the assistance of the Secure Program Design Education Tool to modify unsecure codes to prevent input errors and buffer overflow.


Buffer OverFlow Demostration
Demostration of Buffer Overflow
This is a function to show the buffer overflow
Space created to store three elements
Out of boundary
No. 0
No. 1
No. 2
Other Space
Values of elements
No. 0 0
No. 1 1
No. 2 2
No. 3 3
Demo
Back
Output
Try By Yourself!
Exit

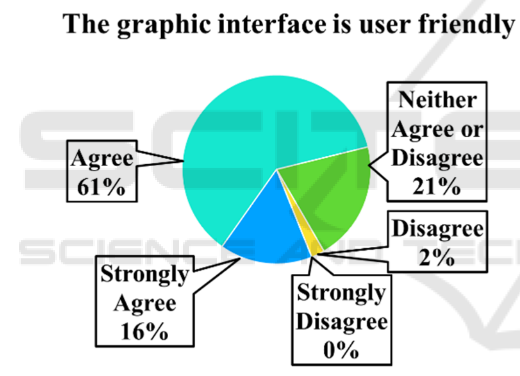
```
public class BuffOverFlow {

        public static void main(String Args[]){
                int[] simpleArray = new int[3];
        // Create an array can store 3 elements.
                for(int i=0;i<4;i++){
                        simpleArray[i]=i;
        // Try to set the value of 4th element.
        // Buffer overflow will happen
                        System.out.println("The No." + i + " element is " + i );
                }
        }
}
```

```
The No.0 element is 0
The No.1 element is 1
The No.2 element is 2
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
at bufferoverflow.BuffOverFlow.main(BuffOverFlow.java:20)
```

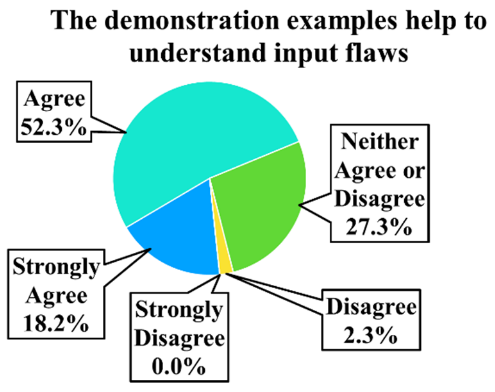
Figure 5: Demonstration of buffer overflow.

After students utilizes the tool, a survey was conducted to measure the successfulness of the education tool gad in helping the students with knowledge study and project implementation. The survey consists of seven questions and students can add any additional comments to help best evaluate the Secure Program Design Education Tool. These questions fall under three different categories which are 1) user friendly, 2) the helpfulness and effectiveness of helping students learning related topics, and 3) open-ended questions. The first set of questions evaluated that the graphic user interface is user friendly or not. The second set of questions evaluated the students on how well their knowledge and skills improved after using the tool. We ask students to use Strongly agree, Agree, Neither Agree or Disagree, Disagree, and Strongly Disagree to evaluate the tool. These questions are: 1) the graphic interface is user friendly, 2) the demonstration examples help to understand input flaws, 3) the demonstration examples help to understand buffer overflow, 4) the tool is practical and helps to understand input validation strategies, and 5) the tool is practical and helps to understand buffer overflow prevention. The survey results are shown in figure 6. The dark blue shows percentage of strongly agree, light blue shows percentage of agree, green shows percentage of neither agree or disagree, yellow shows disagree and light yellow shows strongly disagree. For all questions most students gave strongly agree or agree, no student gives strongly disagree for any question.

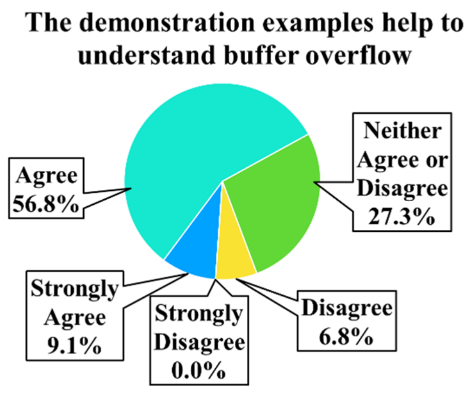
The section asking students to express their level of agreement to how helpful and effective the tool was has yielded positive results with a majority of students agreeing that the tool was helpful, effective, and motivating.



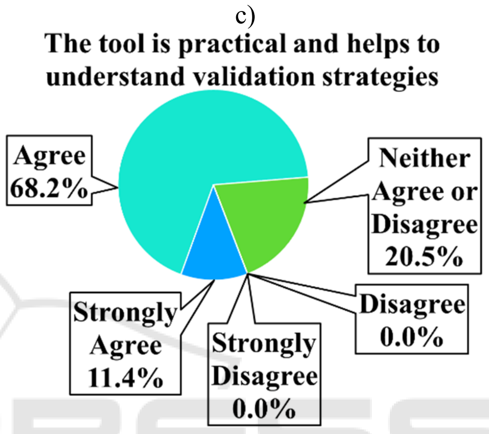
a)



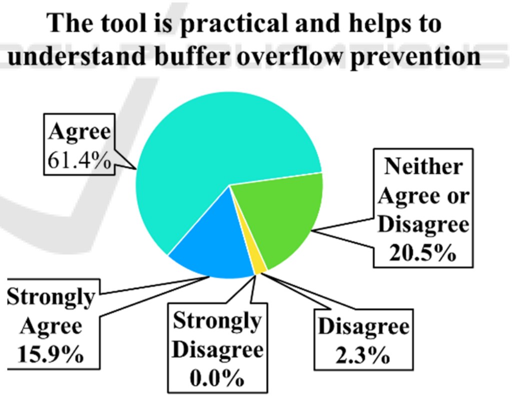
b)



c)



d)



e)

Figure 6: Survey Results of the Education Tool.

Lastly, the students were asked opened questions which were 1) What did you like best about this tool, 2) What did you like least about this tool and 3) any additional comments. Students are sure this tool is a significant resource to them. The student's appreciated the step by step instructions and demonstrations finding that they were helpful and clear. Student found that the Secure Program Design Education Tool was very friendly and very educational saying that it was a great tool to teach students how to design and implement secure applications.

7 CONCLUSIONS

The Secure Programming Design Education Tool was designed and implemented to provide a visual interactive tutorial and step-by-step demonstrations on how input flaws and buffer overflow occur in unsecure software and how to prevent them by modifying unsecure code. The goal of this tool is to help students to better understand the difference of input errors and input flaws, how buffer overflow occurs, how they impact software secure, and how preventing them. This tool also aids students to analyze various cases and modify the code to avoid input flaws. It demonstrates buffer overflow and lets students to change the code to prevent buffer overflow. These step-by-step demonstrations show students various vulnerabilities and give them a chance to modify the codes.

The Secure Program Design Education Tool has been used in COMP 280 Data Structures class Fall 2019. Students' survey results exhibit that the tool assists them in learning input flaws, buffer overflow and how preventing them by hands-on experiences, greatly helps them understand the topics and increases their knowledge level. They really appreciated the step-by-step demonstrations. The results of students' survey are very positive and show that the tool is effective for the course.

ACKNOWLEDGEMENT

This work is supported by National Science Foundation under the award number 1662469.

REFERENCES

Frej, M. O. P., 2002. Analysis of Buffer Overflow Attacks. http://techgenix.com/analysis_of_buffer_overflow_attacks/.

IEEE, 2017. Avoiding the top 10 software security design flaws. http://cybersecurity.ieee.org/center-for-secure-design.

Quinson, M., Gérald, O., 2015. A Teaching System to Learn Programming: the Programmer's Learning Machine. Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education.

Williams, K., Yuan, X., Yu, H., Bryant, K., 2014. Teaching Secure Coding for Beginning Programmers. Journal of Computing Sciences in Colleges (CCSC:MS), Volume 29, Issue 5.

Wikipedia, 2011. Buffer Overflow. 2011. http://en.wikipedia.org/wiki/Buffer_overflow.

Yuan, X., Yang, L., Jones, B., Yu, H., Chu, B., 2015. Secure Software Engineering Education: Knowledge Area, Curriculum and Resources. Information Security Education Journal.

Yu, H., Jones, N., 2014. Secure Software Programming. Journal of American Business Review, Vol. 3, Num. 1.

Abuzaid, A., Yu, H., Yuan, H., Chu, B., 2011. The design and implementation of a cryptographic education tool. Proceedings of the International Conference on Computer Supported Education.

Morana, M., Nusbaum, S., 2008. Input Validation Vulnerabilities, Encoded Attack Vectors and Mitigations. https://www.owasp.org/images/6/6c/Encoded_Attacks_Threats_Countermeasures_9_30_08.pdf.