ASEE 2022 ANNUAL CONFERENCE
Excellence Through Diversity
MINNEAPOLIS, MINNESOTA, JUNE 26TH-29TH, 2022 ASEE
Paper ID #37002

Work-in-Progress: Enabling Secure Programming in C++ & Java through Practice Oriented Modules

Kenneth Andrew Guernsey

Jacob Matthew Tietz (Purdue University Northwest)

Graduated from Purdue University Northwest with a Bachelors in Computer Engineering.

Quamar Niyaz

Quamar Niyaz received the B.S. and M.S. degrees in computer science and engineering from Aligarh Muslim University, in 2009 and 2013, respectively, and the Ph.D. degree from The University of Toledo, in 2017. He has been an Assistant Professor in computer engineering with the ECE Department, Purdue University Northwest, since 2017. He has published papers in the areas of computer and networks security, applied machine learning, and cybersecurity education. His research has been sponsored by the National Science Foundation.

Xiaoli Yang

Dr. Xiaoli (Lucy) Yang is currently the chair and professor of the Department of Computer Science and Engineering at Fairfield University. Dr. Yang's main research interests include virtual/augmented reality, , cybersecurity education, machine learning applications, and software engineering. She has published more than 80 papers in journals and refereed international conference proceedings, and one book by Springer. Dr. Yang has received grants from NSF-National Science Foundation, Indiana Commission of Higher Education, Northwest Indiana Computational Grid Grant, and NSERC-Natural Sciences and Engineering Research Council of Canada.

Ahmad Y Javaid (Dr.)

Ahmad Y. Javaid received his B.Tech. (Hons.) Degree in Computer Engineering from Aligarh Muslim University, India in 2008. He received his Ph.D. degree from The University of Toledo in 2015 along with the prestigious University Fellowship Award. Previously, he worked for two years as a Scientist Fellow in the Ministry of Science & Technology, Government of India. He joined the EECS Department as an Assistant Professor in Fall 2015 and is the founding director of the Paul A. Hotmer Cybersecurity and Teaming Research (CSTAR) lab. Currently, he is an Associate Professor in the same department. His research expertise focuses on application of computational intelligence to various computing domains including but not limited to education, cybersecurity, healthcare, human-machine teaming, and digital forensics. His projects have been funded by various agencies including the NSF (National Science Foundation), AFRL (Air Force Research Lab), NASA-JPL, Department of Energy, and the State of Ohio.

Sidike Paheding

© American Society for Engineering Education, 2022
Powered by www.slayte.com

Work-In-Progress: Enabling Secure Programming in C++ & Java through Practice Oriented Modules

Kenneth Andrew Guernsey[1], Jacob Matthew Tietz[1], Quamar Niyaz[1], Xiaoli Yang[2], Ahmad Y Javaid[3], Sidike Paheding[4]
[1]ECE Department, Purdue University Northwest, Hammond, IN 46323
[2]CSE Department, Fairfield University, Fairfield, 06824, CT, USA
[3]EECS Department, The University of Toledo, Toledo, OH 43606
[4]Applied Computing, Michigan Technological University, Houghton, MI 49931

1. Introduction

In today's society, we are becoming more reliant on technology all around us even for the simplest of tasks. We find increasingly more ways to embed technology into everything we do, which makes life simpler but also brings an underlying issue of cybersecurity beneath the surface. Cyberattacks are growing at an alarming rate with the average cost of $3.86 million for a data breach [1]. One of the key reasons for the growing cyberattacks is the lack of relevant industry workforce. In 2020, the Inter-agency security committee reported that the cybersecurity workforce must grow by 89% to defend critical organizational assets [2]. With the scarcity of cybersecurity experts in the industry workforce, a strong alternative is to expand the knowledge of the software developers to program with a cybersecurity mindset and enable them to use secure software development practices. A major problem encountered in computer science (CS) and computer engineering (CE) curricula is the lack of emphasis on cybersecurity awareness. Students develop programs and software in multiple programming languages, but are inherently unaware of the problems associated with these languages. Students are tasked with completing assignments and labs using the instructed methods, which are usually given under "*perfect*" conditions, i.e. an input will always be valid. This structure is great for learning the topic at hand, but vulnerable if students make habits out of assuming these conditions replicate the real world. Students must be made aware of the problems that coincide with writing code, which will serve as a great foundation and understanding as they continue to develop their knowledge on more advanced topics within their education.

Many institutes have attempted to address the lack of cybersecurity awareness in industry workforce by introducing dedicated courses, tracks, and degree programs into their curricula [3-9]. Although this is a substantial step in the right direction, some problems arise when observed carefully. Cybersecurity degree programs develop experts in the field, but not at a high enough rate to benefit the large gap that exists within the cybersecurity workforce. Cybersecurity courses are efficient at raising awareness to the students who enroll in them, but offered as senior-level elective courses in many CS/CE programs. There is a lack of "bridging" in programming courses that would allow students to understand how programming and cybersecurity are intertwined with one another. If students were given the opportunity to learn the importance of cybersecurity in the programming courses, there would be more enrollment in the security courses that traditionally require a strong understanding of programming as well [10]. Students have the opportunity to take these cybersecurity courses with a prerequisite of programming courses. Once the security course is completed, the materials learned in the course will not be enforced further in their education as many CS/CE courses enforce little to no cybersecurity awareness unless the student is continuing with more individual projects. On the other hand, many CS/CE courses have an opportunity to help strengthen the ability of students to react to certain situations. The opportunity stated before

is the ability to reinforce simple cybersecurity practices, as well as enforce cybersecurity awareness throughout the courses. This can ensure that students are taught to use skills learned in cybersecurity classes in a variety of environments other than the cybersecurity course itself.

The approach that we took to this problem is to develop independent modules corresponding to the introductory topics in C++ and Java, which are taught in programming courses in most engineering schools. These modules will be used alongside students' coursework to emphasize certain security issues within the given language. This allows students to learn about basic cybersecurity topics alongside the programming languages, enforcing a strong foundation and understanding. It is important to mention that nowadays a programming course (e.g. C, C++ or MATLAB) is common in the electrical engineering (EE) curriculum at the freshman or sophomore level. Therefore, the developed modules will not only benefit CE or CS students, but also EE students in learning the fundamentals of cybersecurity concepts in programming.

In this work-in-progress paper, we present the design, formatting, and structure of the modules that we developed. The topic for each module is discussed detailing how the vulnerability is exposed to the student, and how we approached a solution. Finally, future work is discussed as we plan to dive further into the subject of integrating cybersecurity in undergraduate curricula.

2. Related Works

There have been significant efforts made for incorporating cybersecurity in CS/CE courses in the past few years. For example, Towson University has created a website that focuses on concise modules [11, 12]. It provides a good starting point for students to understand the importance of cybersecurity. However, these modules do not have the user actually fix any security flaws, it only serves to point them out and offer solutions to them. The main emphasis of these modules is to ensure secure coding by using a checklist in which students can go one by one to make sure they take care of every checkpoint. These modules are also limited in the scope of covered materials, but received positive feedback from students.

In [13], the authors emphasized cybersecurity awareness through exercises. They taught the importance of creating robust code in introductory courses and integrated lecture materials that focused on common secure programming issues. These exercises were mostly focused on a web development course; and they gave huge emphasis on input validation. In a software engineering course, three main areas were focused: case studies, code review, and version control. A large emphasis was placed on reviewing problems that occurred in real world environments, and the exercises discussed strategies to prevent it.

In [14], the approach was taken in the form of lectures. A survey was conducted with course instructors about what cybersecurity issues affect the information taught in their corresponding courses the most. Once the instructor was able to allocate time, a 75 minutes lecture was given to the students who chose to attend. The topics discussed were the same as the instructor mentioned previously. After the lecture, a questionnaire was given to the students for feedback; the results showed that the majority of students found the cybersecurity topics discussed during the lecture were interesting, useful, and relevant.

Compared to the previous efforts, we have developed interactive modules for comprehensive understanding of cybersecurity in each topic. Students can go through the code in each module and fix security vulnerabilities in it. As time progresses we will not only expand the number of modules we currently have, but also the complexity of the vulnerabilities will also increase with the ongoing project.

3. Methodology

We decided to develop cybersecurity modules for C++ & Java as they are widely used in introductory programming courses. While designing these modules, we kept in mind that the topics must be relevant to real-world issues that we face in the software industry. We used a variety of resources and benchmarks to ensure the authenticity of our chosen topics including Common Vulnerabilities and Exposures (CVE) and Common Weakness Enumeration (CWE) websites [15, 16]. While selecting the module topics for development, we had some restrictions, for example, the topics must be introductory and easy to understand. These modules are geared towards freshman or sophomore level undergraduate students who have just started programming. The security modules that we developed have the following main components as shown in Figure 1:

- **Handouts & Animated slides** – we prepared handouts and presentation slides with animation components to inform students of vulnerabilities, explaining how they occur, and how to respond to them.
- **Lab assignments** – They are used to enforce the information given in the handouts and slides in interactive ways. Students will be given questions and scenarios in which they will see firsthand how the vulnerabilities occur, and how to respond to them.
- **Code templates** – They are frameworks in which students will write code to complete the lab assignments. The purpose of these is to give the student an environment that emphasizes the problem at hand allowing them to focus on the vulnerability corresponding to the lab assignment.
- **Complete solution** – We have prepared the solution source code for the instructors to check students' work, if instructors are adopting these modules in their courses. These solutions could also be used by students to verify their code if they work on the modules independently.

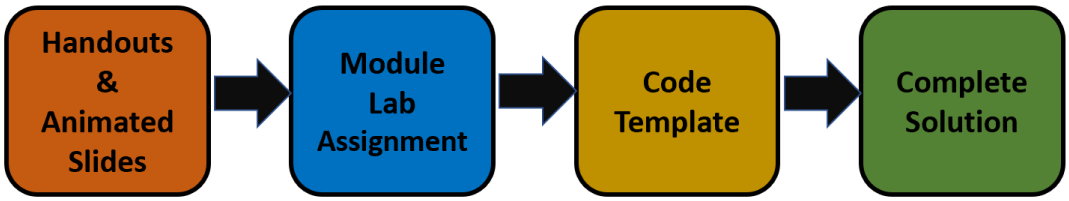

Figure 1: Various components of cybersecurity modules developed for C++/Java.

4. Common Programming Vulnerabilities

The first step towards the development of these modules was to research and analyze the most common programming vulnerabilities students may encounter. The decisions for the topics in the modules weighed heavily on the content taught in introductory programming courses, and on websites such as CWE and CVE. We ultimately came up with the following topics in which we believe students should be aware of, and know how to address them.

4.1 Input Validation

In introductory programming courses, students are introduced to the idea of receiving user input. Lab assignments or projects are typically assigned that can relate to the real world ideas. It is important to demonstrate to students at an early level that the real world does not replicate the ideal conditions taught in the courses, and we must take certain precautions. It is necessary that when working with external data, whether it comes from a keyboard, file, or server, it must be checked before being passed to a method. Validating input can prevent a multitude of problems from

occurring including program crashing, incorrect data flow inside the program, and unexpected results.

4.2 Integer Overflow

Integer overflow is a programming issue that happens with numbers too big or small to be stored in the selected integer data types. C++ and Java have multiple representations for integers with different storages, for example, 8, 16, 32, and 64 bits are used to store `char`, `short`, `int`, and `long` integers, respectively. The default type of integers is signed that allows them to "wrap" around negative numbers once they reach the maximum value. The risk at hand is when an incorrect value is flowing through the program causing a trickle-down effect wherever the variable is used.

4.3 Random Number Generation

Students are introduced to random number generation early in their programming courses as a small subtopic that can be used in future labs and/or projects. Students are not really exposed to the applications of random number generation and the actual engineering behind it, along with the risks it poses. The risk occurs when one is involved with sensitive data like passwords, or cryptography. The methods students are exposed to generate random numbers are deterministic. This implies that the numbers generated using a carefully designed and consistently repeated algorithm are not "*truly*" random that poses risks as adversaries can run brute force algorithms to predict the random numbers generated through a deterministic method to crack passwords, session tokens, and so on [11].

4.4 Null Pointer Reference

Any student exposed to an object-oriented programming language is certainly going to be involved with classes, objects, and methods. Students need a solid understanding of reference-type data types and how a literal called `null` can be injected into a program. When `null` is injected into a program, the compiler does not allocate memory for the `null` object, so if a statement references an object or a variable with this value, an exception will be thrown. If the exception is not handled correctly, the program will crash and stop running.

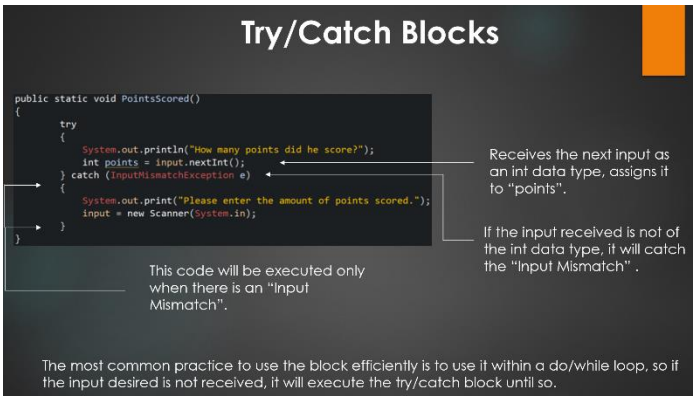

Figure 2: Try/Catch block for Input Validation.

4.5 Constructors, Public & Private Keywords

Constructors are used to initialize objects from classes. Based on their implementation, they assign initial values to class variables. This can be directly helpful to address the *Null Pointer Reference* problem as any reference-type data types in the class can be given an initial value instead of `null`. The keywords public, private, and protected are used to implement data encapsulation into the programs. The `public` keyword allows a variable to be accessed and modified by any class, which is not recommended as programmers should control who can modify the variables within their classes. The `private` keyword allows a variable to be modified and accessed only by the class inside which it has been declared. This provides a layer of security as the programmer can control

who can modify the variable. If access to the variable is needed to provide outside its class, then `get()` and `set()` methods must be implemented for them.

5. Overview of Developed Modules in C++ and Java

In this section, we provide an overview of a few modules that we developed for introductory programming courses in C++ and Java.

5.1 Modules developed for an Introductory Java Course

5.1.1 Input Validation

To prepare students better for situations in which they may encounter input validation, it has been emphasized that external data input cannot be trusted. The objective for this module is to discuss three main ideas: i) need for input validation ii) types of input validation, and iii) approach for input validation. We discuss that input validation is must because improper validation of external data may cause problems like program crashing, incorrect data flow, unexpected results, and security vulnerabilities. Following that, we provide overview of different types of input validation including type, length, range, divide by zero, and format. We then discuss how we can validate input, and the two methods we suggested were `Try` and `Catch` blocks and regular expressions. An example of Try/Catch blocks is shown in Figure 2, taken from the PowerPoint slides that we prepared.

The lab assignment is focused on having the students validate different types of input based on the situation. They are given the code outline and asked to modify given methods based on the specific task they are given. Once they complete the assignment, they can either submit it to an instructor for evaluation or use the provided solution code to check their work if learning independently. Figure 3 shows an example of a completed question. The method `validateUsername()` initially assigns a user input to the string variable `username`. Then, the while loop checks that if string username fails the pattern in regular expression or is an empty string. The regular expression allows only lowercase and uppercase letters for username. If either of these conditions are true it will prompt the user to enter a valid username.

```
public static void validateUsername()
{
     username = input.nextLine();
     while (!username.matches("[a-zA-Z]*") || username.equals(""))
     {
          System.out.print("Please enter a valid username: ");
          username = input.nextLine();
     }
}
```

Figure 3: Regular expression for input validation.

5.1.2 Integer Overflow

In this module, we discuss the multiple ways to trigger an integer overflow. The objective of this module was to cover four main ideas:

- What types of integers are there?
- Why must we prevent integer overflow?
- How does an integer overflow occur?
- How can we prevent integer overflow?

An integer overflow must be prevented because it can cause problems such as program crashing, incorrect data flow, unexpected results, and security vulnerabilities. In our learning module, we describe the two main types of integer overflow: arithmetic overflow and conversion overflow. Arithmetic overflow is when a mathematical operation is performed and the result is stored in an integer value that is too small. Conversion overflow is when an integer is converted

to a different data type i.e. type-casted, and the destination data type is too small to hold the original value. The lab assignment is focused on having the student visualize integer overflow firsthand, by both arithmetic and conversion. In the lab assignment, students are given a variety of different numbers and are asked to determine which integer data type would fit best. They are also given a sequence of steps they must follow to visualize the orderly occurrence of an integer overflow. Once the student has completed the assignment, they can either submit it to an instructor or use the provided completed solution to check their work. An example of a completed question is shown in Figure 4. It shows the lab assignment that simulates a simple cafeteria store. Students are asked to enter an option: 0, 1, or 2. The option 0 will "leave" the store, while options 1 and 2 allow purchasing items of different prices. The variables `totalQuantity` and `price` are both initially set as byte data types, which have a maximum value of 127. Students are then instructed to enter specific values to show three main conditions:

- Both variables store data correctly as the input is smaller than 127, and the result of the multiplication is smaller than 127.
- Only the quantity variable stores the data correctly as the input is smaller than 127, and the result of the multiplication is larger than 127, causing the variable `price` to wrap around.
- Both variables store data incorrectly as both the input and the multipiction result are larger than 127.

By performing the tasks in order, students will be able to visualize that the `price` variable is highly prone to overflow as it stores the multiplication result. Students are expected to fix the problem by changing the data types of two variables mentioned previously. Students need to be aware of the nature of signed numbers. Students must understand that once a signed integer reaches its maximum value, it wraps around with a negative value. Ensuring that students use the correct data type when assigning a value can be crucial to prevent overflows. Students being aware about which mathematical operations are at risk of growing quickly for values is also essential, like exponential, multiplication, and factorials. When performing tasks with the risk of growing quickly, validating parameters to ensure that numbers will not be too large would work as an effective method as well.

```
public static void main (String args[])
{
    byte totalQuantity;
    byte price;
    System.out.println("Welcome to the store!");
    do {
        System.out.println("Please enter 1 for item 1, which costs $10");
        System.out.println("Please enter 2 for item 2, which costs $25");
        System.out.println("Please enter 0 or else to leave and purchase items.");
        inputByte = in.nextByte();
        if (inputByte==1) {
            System.out.println("How many would you like to buy?");
            quantity = in.nextInt();
            totalQuantity += quantity;
            price += quantity * 10;
        } else if (inputByte==2) {
            System.out.println("How many would you like to buy?");
            quantity = in.nextInt();
            totalQuantity += quantity;
            price += quantity * 25;
        } else{
            break;
        }
    }while(true);
}
```

Figure 4: Listing for integer overflow in the store problem.

5.1.3 Random Number Generation

The main idea behind this module is that students should have a solid understanding between the deterministic and non-deterministic random number generation. The module focuses on four main topics:

- How to traditionally generate random numbers?
- What are the usages of random numbers?

- What are the risks of deterministic random number generation?
- What is the secure way to generate random numbers?

First, we comment on how an introductory programming course might introduce random number generation, i.e., invoking Java's `Math.random()` method. We then discuss risks of using deterministic random number generation as numbers generated are pseudo-random and not truly random [9]. With this knowledge, students should understand not to use this random number generation technique when dealing with sensitive data and other things of that nature. Ensuring that students understand the usage of the correct type of random number generator can guarantee little vulnerabilities in their programs caused by the random number generator itself. When working with sensitive data, web applications, and cryptography, Java implements its own non-deterministic random number generator class `SecureRandom` that students are encouraged to use.

```
import java.security.SecureRandom;
public class NumberGenerator {
     SecureRandom generator = new SecureRandom();
     int randomNumber = generator.nextInt(1000);
}
```

Figure 5: Secure random number example in Java

Although there is no lab assignment for this module, there is a Java program for students to run. The program is a "guessing game" in which a random number is generated and the student must guess the number. Once the number is guessed correctly, the program returns the time that it takes to guess the number and the number of guesses. The student has the option to let the computer guess the number as well, in which we implemented a simple binary search algorithm to find the number. The reasoning for this lab is to help students understand how fast computers can calculate and how certain things can be guessed through brute-force.

5.1.4 Null Pointer Reference

The main goal of this module is to help students fundamentally understand what is `null`, and how to avoid the run-time `NullPointerException`. Four main concepts are discussed in this module:

- What is `null`?
- How does `null` enter into the program?
- What is `NullPointerException`?
- How to avoid `null` & `NullPointerException`?

First, we laid a basis on the concept of `null`. We discuss that `null` is a default value for uninitialized variables of reference type and its binary representation is all 0's. We then discuss that `NullPointerException` is a run-time error that occurs when following conditions are met:

- A method is called with a `null` object.
- Attempted access to an instance variable of a `null` object.
- A `null` object is passed as an argument.
- Retrieving length or indices for a `null` array.

We then discuss how we can avoid `NullPointerException` with four different methods shown in Figure 6 taken from our PowerPoint slides. These methods ensure that an object should not be `null` before using it. These are accomplished in two main ways by using either logical operators or methods inherited from the Object class. By ensuring that the value of an object is not `null` before its usage, the program will not crash by throwing the `NullPointerException`. The lab assignment for this module allows students to visualize how null can be injected into a program through misuse of constructors, and calling methods with a null array. Figure 7 shows an example of the code students are expected to write while completing the lab assignment.

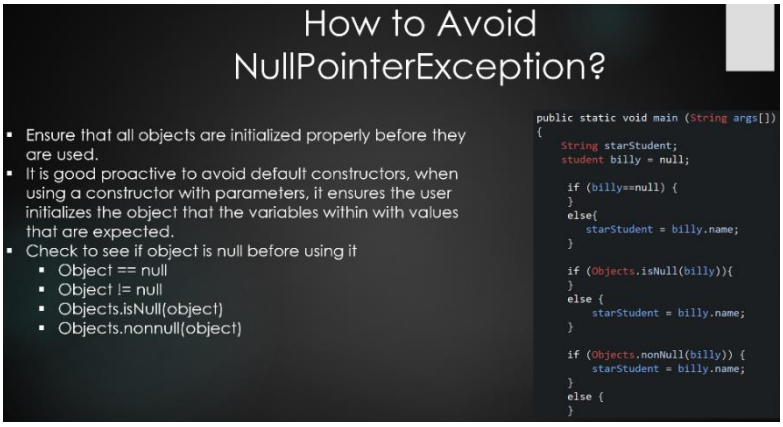

Figure 6: Null pointer module for Java.

5.1.5 Constructors, Private & Public Keywords

The purpose of this module is to give students a solid understanding of data encapsulation and constructors. The objective of the module was to enforce three main ideas:

- What are constructors?
- What are the public and private keywords?
- What are get/set methods?

```
Student bob = null;
jim = new Student("Jim", 19, 13);
clara = new Student("Clara", 12, 7);
if(Objects.isNull(bob)){

}else {
     System.out.println(bob.older(jim));
}
if (bob!=null){
     System.out.println(bob.name);
}
```

Figure 7: NullPointerException code snippet.

We begin by discussing what is a constructor and its application. We go into detail about how this is a method, which is called when an instance of an object is created or initialized. We also discuss how constructors are used to initialize variables contained within an object. We then discuss the difference between private and public keywords, and their advantages and disadvantages. The `get()` and `set()` methods are explained in detail about how they are used to access private variables to either gather or set their values.

```
public Customer (float y){
     money = y;
     hasTicket = false;
}
public void BuyTicket(TicketBooth a){
     if(money >= a.getTicketPrice() && a.getTickets()>0){
          a.sellTicket();
          money -=a.getTicketPrice();
          hasTicket = true;
     }
}
```

Figure 8: A code snippet for constructor module.

The lab assignment for this module is focused on students being in-charge of a ticket booth at a basketball game, and they must focus on implementing constructors and get/set methods correctly. They are given a test file to evaluate whether or not their code is correct. Figure 8 shows an example of what students are expected to write while completing the lab assignment. The figure shows code snippet for the `Customer`, which is a constructor and `BuyTicket()` method that takes an object of `TicketBooth` class as a parameter. The student must use a get method to get the value of a `TicketBooth` variable,

which is set as private. Students are expected to do this with a multitude of variables throughout the lab assignment.

5.2 Modules developed for an Introductory C++ course

5.2.1 Integer Overflow

In this module, students must figure out how to take a valid input that does not overflow the variable they are using it to store the input. The code snippet in Figure 9 is one of the ways students can achieve this. Students must also figure out how to detect an integer overflow that occurs due to addition, subtraction, multiplication, and division. This is done by providing students with a scenario in which they will find out implications of not checking for an overflow.

```
while (!(cin >> input1)) {
     cout << "Input invalid, try again." << endl;
     cin.clear();
     cin.ignore(numeric_limits<streamsize>::max(),'\n');
     cout << "Please enter a valid integer value for input1: ";
}
```

Figure 9: Valid range check for integer input.

The code snippet in Figure 10 can be used to solve this problem. It involves taking the sum which may or may not have overflowed then does the reverse operation. In this case, it performs subtraction to test if an overflow has occurred as it will produce a different result if an overflow has occurred. This module uses constants in the `limits` library to show students the minimum and maximum values of different variable types. The `bitset` library is used to display the underlying binary of each number a student inputs into the example file. This helps the student visualize what is happening during the overflow.

```
int16_t output1 = input1 + input2;
cout << "input1 + input2: ";
if (output1 - input1 != input2) {
     cout << "An overflow occurred" << endl;
}
else {
     cout << output1 << endl;
}
```

Figure 10: Overflow check in addition.

5.2.2 Random Numbers

With modern hardware only ever increasing in speed, random numbers become easier and faster to brute force through simple methods. This module includes a guessing game in which students can experiment with guessing random numbers of varying length. It will also display the time that students take to guess the random number. For the main problem of the module, students are given skeleton code in which they experiment with generating seeds for pseudo-random number generation. There is also a simple loop, which iterates up to show students just how fast every single number up to the maximum random number can be guessed.

5.2.3 Dangling Pointers

Pointers are used to point to memory locations containing variable data. They store addresses, which are memory locations of the data. If a piece of data has a singular pointer pointing to it, that is the only way to access that data; if that pointer were to be deleted, the data would be stuck not being able to be accessed or deleted. When memory cannot be accessed or deleted, it is a memory leak, it takes up memory whilst being completely useless. This can happen if a pointer points to a value, then that pointer is modified to point to a different value, leaving the first value inaccessible. Dangling pointers, on the other hand, are pointers that point to memory that has been freed. If memory is freed, it can be overwritten. Accessing memory that may have been overwritten can

lead to undefined and seeming random behavior of a program. A dangling pointer can be created by calling "delete" along with a pointer name. This frees the memory at the location the pointer points to but does not delete the pointer or change where the pointer points, creating a dangling pointer. A dangling pointer can be fixed by either assigning a new address to the pointers or setting them to `NULL` which sets them to be null pointers. In this module, students are challenged to use only three addresses to complete a series of calculations all of which have their own pointers but must be properly set to null pointer after use for avoiding dangling pointers. This helps the student deepen their understanding of pointers.

5.2.4 Parameter Mismatch

In C++, function parameters can differ from what the function call sends to the function all whilst not showing any signs of odd behavior. What can happen is when the function asks for a `short` integer but receives an `int` for example, which may cause overflow problem inside the function when performing its task. If the `int` that was sent to the function was larger than what could fit into a `short`, that would be loss of data, the `int` would be truncated to fit into the short. This may seem easy to avoid at first, but in large projects with many developers, small details like this are easily missed. It is also not just the input parameters; the return values can also be forced into the wrong type of variable, which can cause issues as well. If a function returns a `double`, but then the returned value is stored in an `int`, this would also cause loss of data. In Figure 11, students are given four inconsistent functions all with different return types and input parameter types. These functions are then called and their return values are stored in different data types than the return types, which can cause data loss if the variable in which the function returns is stored is not of adequate type. The solution to overcome with these issues is to make sure that students understand multiple data types and the size difference between them by setting all the input parameters and returning values to values that will not cause a loss in data.

```
// Called inside a main function
short sum = add(x, y);
cout << "Sum: " << sum << endl;

short difference = subtract(x, y);
cout << "Difference: " << difference << endl;

double product = multiply(x, y);
cout << "Product: " << product << endl;

int quotient = divide(x, y);
cout << "Quotient: " << quotient;

// Function definitions
double add(short x, short y) {
    return x + y;
}
short subtract(int x, int y) {
    return x - y;
}
long multiply(char x, char y) {
    return x * y;
}
float divide(double x, double y) {
    return x / y;
}
```

Figure 11: Code snippet for parameter mismatch.

6. Conclusion and Future Works

In this work-in-progress paper, we discussed multiple programming modules developed for enforcing the secure programming mindset in introductory programming courses on C++ and Java. The modules will guide students through specific cybersecurity topics to improve their skills and help students in their future programming careers by teaching them the necessary skills to prevent common cybersecurity vulnerabilities. Future work will include developing more modules focused on more advanced topics in Java and C++ taught later in the CS/CE curriculum. Before disseminating these modules for adoption in different institutes, we will evaluate their impact in delivering cybersecurity concepts for secure programming. We will use these modules in our programming courses in Fall 2022 and Spring 2023 at Purdue University Northwest, The University of Toledo, and Michigan Technological University; and assess students' performance. For each module, pre-module and post-module surveys will be conducted to measure students'

learning for cybersecurity concepts relevant to the topic. Students' submission will also be evaluated to investigate their logic to solve a module. In addition, we will also take students' feedback for each module for its quality and documentation. Based on students' evaluation and feedback, we will revise our modules and disseminate them to other institutes for adoption in programming courses. A website will be hosted for the distribution of these modules to other institutes.

References

1. "Cost of a Data Breach Report", IBM Security [Online]. Available: https://www.capita.com/sites/g/files/nginej291/files/2020-08/Ponemon-Global-Cost-of-Data-Breach-Study-2020.pdf. Accessed Mar 27, 2022.
2. "Cybersecurity Professionals Stand Up to a Pandemic," $(ISC)^2$ Cybersecurity Workforce Study, 2020 [Online]. Available: https://www.isc2.org/-/media/ISC2/Research/2020/Workforce-Study/ISC2ResearchDrivenWhitepaperFINAL.as. Accessed Mar 27, 2022.
3. Sami Krause, "College of Business launches new cybersecurity management programs," [Online]. Available: https://communique.uccs.edu/?p=118684. Accessed Mar 27, 2022.
4. R. Weiss, F. Turbak, J. Mache, and M. E. Locasto, "Cybersecurity education and assessment in EDURange." IEEE Security & Privacy, 15(03), 90-95.
5. N. Rahman, I. Sairi, N. Zizi, and F. Khalid. "The importance of cybersecurity education in school." International Journal of Information and Education Technology, 10(5), 378-382.
6. A. Igonor, R. L. Forbes, and J. McCombs, "Cybersecurity Education: The Quest to Building Bridge Skills." ISSA Journal, 17(18), 18-26, 2019.
7. T. Lowe, and C. Rackley. "Cybersecurity education employing experiential learning." KSU Proceedings on Cybersecurity Education, Research and Practice, 5, 2018.
8. J. Ricci, F. Breitinger, and I. Baggili. "Survey results on adults and cybersecurity education." Education and Information Technologies, 24(1), 231-249.
9. W. A. Hill Jr, M. Fanuel, X. Yuan, J. Zhang, & S. Sajad (2020). "A survey of serious games for cybersecurity education and training." KSU Proceedings on Cybersecurity Education, Research and Practice, 7, 2020.
10. Jason M. Rubin, "Can a computer generate a truly random number?" Online. Available: https://engineering.mit.edu/engage/ask-an-engineer/can-a-computer-generate-a-truly-random-number/. Accessed Mar 27, 2022.
11. "Cybersecurity Modules: Security Injections," [Online]. Available: https://cisserv1.towson.edu/~cssecinj/, Accessed Mar 27, 2022.
12. S. Raina, S. Kaza, and B. Taylor. "Security Injections 2.0: Increasing Ability to Apply Secure Coding Knowledge Using Segmented and Interactive Modules in CS0." In Proceedings of the 47th ACM Technical Symposium on Computing Science Education, 2016.
13. K. Nance, B. Hay, and M. Bishop. "Secure Coding Education: Are We Making Progress?" In 16th Colloquium for Information Systems Security Education, 2012.
14. C. Yue. "Teaching computer science with cybersecurity education built-in." In 2016 USENIX Workshop on Advances in Security Education (ASE), 2016.
15. "Common Weakness Enumeration: CWE", Online. Available: https://cwe.mitre.org/, Accessed Mar 27, 2022.
16. "Common Vulnerabilities and Exposures (CVE)", Online. Available: https://cve.mitre.org/, Accessed Mar 27, 2022.