Security Module: Integer Overflow

Description

Integer overflow is a security issue caused by overflowing a data type’s storage limitations. Integer overflow occurs when arithmetic operation give a numeric value that is too big to be stored in integer variable.

Objective

The objectives of this security module are to illustrate the limitations that a data type can have and show the consequences of integer overflow can have on your system data.

Activities

There are three activities for this module. The first is a discussion assignment, where students are given a short narrative and questions covering integer overflow. Second, students are given a Java program they can run that simulates a bank account. The program uses a command line interface. The account starts with an initial balance of 0 dollars. The user has 3 options in the menu: deposit money, withdraw money, and exit. The data type for storing the balance will be a short. Ask the student to attempt to break the program by depositing and/or withdrawing money. Third, students are given the code for the previous assignment. They are asked to modify the program to fix the security flaw caused by overflow. They can do this in two ways: change the data type to be a long or check for max values before adding/subtracting the deposit/withdraw amount.

Module Contents

- Assignment 1: Integer Overflow Discussion
- Assignment 2: Break the Program
- Assignment 3: Fix the Program w/source code
- Source code for IntegerOverflow class
- Source code for Menu class (used in IntegerOverflow program)
Assignment 1: Integer Overflow Discussion

What is Integer Overflow?

Integer overflow is a security issue caused by overflowing a data type’s storage limitations. Integer overflow occurs when arithmetic operations result in a numeric value that is too big to be stored in integer variable. A physical example of this problem can be seen with your car’s odometer, which has only 6 dials. If your car’s odometer is reading 999500 for having traveled 99,950.0 miles, what happens when you travel another 1000 miles? The odometer will roll over to 000500, resulting in the appearance of the car having only ever traveled 500 miles. This is an mechanical example of the integer overflow problem. Even though it’s the “Integer Overflow” problem, it applies to all primitive data types.

This same principal can be applied to primitive data types, such as Java’s short data type. Short can only store values between -32,768 and 32,767 inclusively. Let say you have the variable shortOdometerReading defined as a short. If shortOdometerReading has a value of 31,890 and 3,000 was added to it. What would be the value stored in shortOdometerReading? -30646 When storing a value the system will only use the least significant bits that can be stored in a variable. Since the short data type can only store 16 bits anything over 16 bits is ignored. This cause the value to wrap around to the lowest number it can store and start counting up from there. Since the lowest value is -32,768 and the highest value is 32,767 the variable will wrap around when it hits 32,768.

To demonstrate the limitations of the data type short the below code is given for you to analyze and answer questions about.
```java
1  short myShort = 32765;
2
3  myShort = myShort + 1;
4  System.out.println(myShort);
5
6  myShort = myShort + 1;
7  System.out.println(myShort);
8
9  myShort = myShort + 1;
10 System.out.println(myShort);
11
12 myShort = myShort + 1;
13 System.out.println(myShort);
14
15 myShort = myShort + 1;
16 System.out.println(myShort);
```

1. Choose the two answers that equal the minimum and maximum values that a `short` can store?

   (a) -2,147,483,648  
   (b) -32,768  
   (c) 2,147,483,647  
   (d) 32,767

2. What will be the result of the `System.out.println(myShort)` statements on lines 4, 7, 9, 13, 16?

3. Explain what happens when the value being stored in `short` is too large to fit in the data type.
Assignment 2: Break the Program

In this assignment, you will be given a Java program for a simple bank account. This program will track the balance for your account. You can deposit and withdraw money from your account using the command line interface. To deposit money into your account, use option 1. The prompt will ask you “How much money to deposit?”. You must input a positive number that will be added to your balance. To withdraw money out of your account, use option 2. The prompt will again ask you, “How much money to withdraw”. You must enter a positive number that will subtracted from your balance.

Your assignment is to try and break the simple bank program. As you try to break the program, keep track of the amount you deposit or withdraw along with the expected balance and the actual balance. Hint: try depositing or withdrawing large amounts of money.

1. Did you have an actual balance that did not match your expected balance?

2. If yes, what was your actual balance and your expected balance?

3. Based on the values from the previous question, can you determine the date type being used to store the balance? What is that type?
Assignment 3: Fix the Program

The next page has the source code from the previous assignment, where you were asked to break the simple bank program. Your assignment is to identify what needs to be modified in the source code to fix the overflow problem. Use your answers from the previous assignment to help you identify three locations in the code needing modification. When asked to identify a line of code, provide the line number.

1. What data type is used to store the balance?

2. What other data types could be used to store the balance and why?

3. What is the first line of code that needs modified? What do you suggest it be changed to?

4. What is the second line of code that needs modified? What do you suggest it be changed to?

5. What is the third line of code that needs modified? What do you suggest it be changed to?
import java.util.Map;
import java.util.HashMap;
import java.util.ArrayList;
import java.util.Scanner;

class IntegerOverflow {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        short balance = 0;

        Menu menu = new Menu("Main Menu");
        menu.addOption(1, "Deposit Money");
        menu.addOption(2, "Withdraw Money");
        menu.addOption(9, "Exit");

        while(true) {
            System.out.print("\n\n\nCurrent balance: " + balance);
            switch(menu.getOption()) {
                case 1:
                    System.out.print("How much to deposit: ");
                    short depositAmount = in.nextShort();
                    balance += depositAmount;
                    break;
                case 2:
                    System.out.print("How much to withdraw: ");
                    short withdrawAmount = in.nextShort();
                    balance -= withdrawAmount;
                    break;
                case 9:
                    System.out.println("Exiting");
                    System.exit(0);
                    break;
            }
        }
    }
}
import java.util.Map;
import java.util.HashMap;
import java.util.ArrayList;
import java.util.Scanner;

class IntegerOverflow
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        short balance = 0;

        Menu menu = new Menu("Main Menu");
        menu.addOption(1, "Deposit Money");
        menu.addOption(2, "Withdraw Money");
        menu.addOption(9, "Exit");

        while(true)
        {
            System.out.print("\n\n\n\nCurrent balance: "+ balance);
            switch(menu.getOption())
            {
                case 1:
                {
                    System.out.print("How much to deposit: ");
                    short depositAmount = in.nextShort();
                    balance += depositAmount;
                    break;
                }

                case 2:
                {
                    System.out.print("How much to withdraw: ");
                    short withdrawAmount = in.nextShort();
                    balance -= withdrawAmount;
                    break;
                }

                case 9:
                {
                    System.out.println("Exiting");
                    System.exit(0);
                    break;
                }
            }
        }
    }
}
Menu Class Source Code

class Menu
{
    class Option
    {
        private int id;
        private String desc;

        public Option(int id, String desc)
        {
            this.id = id;
            this.desc = desc;
        }
        public int getId()
        {
            return this.id;
        }
        public String getDesc()
        {
            return this.desc;
        }
    }

    private ArrayList<Option> _options;
    private String _title;

    public Menu(String title)
    {
        this._options = new ArrayList<Option>();
        this._title = title;
    }

    public void addOption(int id, String desc)
    {
        this._options.add(new Option(id, desc));
    }

    public void clearOptions()
    {
        this._options.clear();
    }

    public int getOption()
    {
        Scanner in = new Scanner(System.in);
        while(true)
        {
            System.out.println("\n------------------------------------------------------");
        }
    }
}
System.out.println("" + this._title + "\n");

for (Option option : this._options) {
    System.out.println("" + option.getId() + " " + option.getDesc());
}
System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-");

System.out.print("Select option: ");
int optionInt;
String optionStr = in.nextLine();

    //--- Attempt to convert the inputed string to an integer
try {
    optionInt = Integer.parseInt(optionStr);
} catch (NumberFormatException ex) {
    System.out.println("Error: It appears you did not input a option number");
    continue;
}

    //--- Check if the option exists in our menu
for (Option option : this._options) {
    if (option.getId() == optionInt) {
        return optionInt;
    }
}
System.out.println("Error: Invalid option selected");