Secure Programming

GEEN163

“Treat your password like your toothbrush. Don't let anybody else use it, and get a new one every six months.”

Clifford Stoll

Computer Security Concepts

- The NIST Computer Security Handbook defines computer security as the protection to attain the objectives of preserving the integrity, availability, and confidentiality of information system resources
- ***Data integrity*** assures that information and programs are changed only in a specified and authorized manner. A loss in integrity is the unauthorized modification or destruction of information.
- ***Data Confidentiality*** assures that private or confidential information is not made available or disclosed to unauthorized users.
- ***Availability*** assures that systems work promptly and service is not denied to authorized users. A loss of availability is the disruption to access to or use of information or an information system.

Violations to Security

Violations to security can be placed in two categories:

- ***Accidental misuse*** is easier to protect against because normal protection mechanisms (access control, roles, etc.) usually stop these from occurring
- ***Malicious violations*** are more difficult to prevent because attackers may know how to avoid normal protection mechanisms

Software Security

- Computers are still being exploited because of vulnerabilities that occur due to insecure code
- Secure practices must be used throughout the entire software development
- Poor programming practices often result in many security vulnerabilities
  - Buffer overflow
  - Denial of service
  - Cross-site scripting
  - Code injections
  - Invalid input
  - Improper error handling
- All of them can be caused by insecure code

Buffer Overflow

- Buffer overflow does **NOT** happen in Java
- In other languages, such as C++, if you go past the end of an array it will not recognize the error
- A program can change other variables, change the method return location or load other programs

Buffer Overflow Example

```
int[] array = { 11, 22, 33};
int dog = 7;
for (int i = 0; i <= 3; i++) {
        array[i] = i;
}
System.out.println( dog );
```

Memory Layout

i	array[0]	array[1]	array[2]	dog
0	11	22	33	7

Buffer Overflow Example

```
int[] array = { 11, 22, 33};
int dog = 7;
for (int i = 0; i <= 3; i++) {
    array[i] = i;
}
System.out.println( dog );
```

Memory Layout

i	array[0]	array[1]	array[2]	dog
0	0	22	33	7

Buffer Overflow Example

```
int[] array = { 11, 22, 33};
int dog = 7;
for (int i = 0; i <= 3; i++) {
        array[i] = i;
}
System.out.println( dog );
```

Memory Layout

i	array[0]	array[1]	array[2]	dog
1	0	22	33	7

Buffer Overflow Example

```
int[] array = { 11, 22, 33};
int dog = 7;
for (int i = 0; i <= 3; i++) {
    array[i] = i;
}
System.out.println( dog );
```

Memory Layout

i	array[0]	array[1]	array[2]	dog
1	0	1	33	7

Buffer Overflow Example

```
int[] array = { 11, 22, 33};
int dog = 7;
for (int i = 0; i <= 3; i++) {
        array[i] = i;
}
System.out.println( dog );
```

Memory Layout

i	array[0]	array[1]	array[2]	dog
2	0	1	33	7

Buffer Overflow Example

```
int[] array = { 11, 22, 33};
int dog = 7;
for (int i = 0; i <= 3; i++) {
    array[i] = i;
}
System.out.println( dog );
```

Memory Layout

i	array[0]	array[1]	array[2]	dog
2	0	1	2	7

Buffer Overflow Example

```
int[] array = { 11, 22, 33};
int dog = 7;
for (int i = 0; i <= 3; i++) {
        array[i] = i;
}
System.out.println( dog );
```

Memory Layout

i	array[0]	array[1]	array[2]	dog
3	0	1	2	7

Buffer Overflow Example

```
int[] array = { 11, 22, 33};
int dog = 7;
for (int i = 0; i <= 3; i++) {
    array[i] = i;
}
System.out.println( dog );
```

Memory Layout

i	array[0]	array[1]	array[2]	dog
3	0	1	2	3

Buffer Overflow Example

```
int[] array = { 11, 22, 33};
int dog = 7;
for (int i = 0; i <= 3; i++) {
        array[i] = i;
}
System.out.println( dog );  // displays 3
```

Memory Layout

i	array[0]	array[1]	array[2]	dog
3	0	1	2	3

Programs with Bugs

- Most security flaws are caused by errors in the program or failure to check if the input is reasonable
- If you write a program that operates correctly for all input, then it is very unlikely that it can be exploited

It’s All About You

You are the cause

- As a programmer, insecure code that you may write can leave users vulnerable to attack

You are the solution

- By writing secure code, you can protect users from attack

Planning to be Secure

- Crucial steps are
  - Awareness of the many security vulnerabilities
  - Accounting for and handling all error states
  - Using various testing techniques to identify and eliminate bugs
  - Writing secure code

Java Security

- Java always checks array bounds. This protects against buffer overflow exploits
- The Java Security Manager limits what a program can do
- Java applets cannot:
  - Read or write files
  - Open network connections to other computers
  - Get information about the user's computer
- Digitally signed applets are trusted and can do more

Input Validation

- In computer science, data validation is the process of ensuring that a program operates on clean, correct and useful data
- A good program should check that data received from a user is reasonable
- It is very easy for someone to accidently enter incorrect information
- Catching the mistake early will avoid future problems

Write with your team

- Read a number from a keyboard Scanner object and make sure it is between 1 and 5
- If invalid, tell the user and ask for better input

Possible Solution

```
int num;
do {
   System.out.println("enter a number from 1 to 5");
   num = keyboard.nextInt();
   if ( num < 1 || num > 5 ) {
      System.out.println("Bad number, try again");
   }
} while( num < 1 || num > 5) ;
```

Validation Example

- Imagine you are writing a phone book program and are reading phone numbers
- You might want to check that the user entered 10 numerical digits

```
String phone = keyboard.next();  // read number
int numDigit = 0;                // count of numbers
for (int i = 0; i < phone.length(); i++) {
    if (Character.isDigit( phone.charAt(i) ) )
        numDigit++;
}
if (numDigit == 10) // Check if 10 numbers
```

Pattern Class

- The java.util.regex.Pattern class can be very useful for making sure input matches the expected “pattern”
- Patterns are expressed as Regular Expressions
- You could define a pattern that allows only
  - 9876543210
  - (987)654-3210
  - 987.654.3210

Range Checking

- Consider the following program segment

```
String[] rate = {"good", "OK", "bad"};
System.out.println("Enter a number from 1-3");
int num = keyboard.nextInt();
System.out.println( rate[num - 1] );
```

- What happens if the person enters 5?

What error check will best protect the program?

1. if (num < 5)
2. if (num <3 && num > 1)
3. if (num <=3 && num >= 1)
4. if (num <=3 || num >= 1)

Check Digits

- Many system incorporate a check digit to ensure input is correct
- The last digit of an International Standard Book Number (ISBN) is a check digit

$$lastDigit = \left(\sum_{i=1}^{9} i * digit_i \right) mod\ 11$$

- The letter X is used if the calculation gives 10

Complete the Program

```
int[] digit = {1, 4, 4, 9, 6, 0, 4, 3, 8};
int lastDigit, sum = 0;

// Calculate the ISBN check digit here


if ( lastDigit  == 10 )
        System.out.println("X");
else
        System.out.println( lastDigit );
```

Possible solution

```
int[] digit = {1, 4, 4, 9, 6, 0, 4, 3, 8};
int lastDigit, sum = 0;
for (int i = 0; i < digit.length; i++) {
        sum += digit[i] * (i+1);
}
lastDigit = sum % 11;
if ( lastDigit  == 10 )
        System.out.println("X");
else
        System.out.println( lastDigit );
```

$$lastDigit = \left(\sum_{i=1}^{9} i * digit_i \right) mod\ 11$$

Decimal Numbers

- Decimal numbers work well for humans.
- Each position is 10 times the one to the right

3,148 is

1000's	100's	10's	1's
3	1	4	8

Binary Numbers

- Computers usually use binary numbers, base 2
- Binary numbers only have two digits, 0 and 1
- Each position is 2 times the one to the right

010011_2 (19_{10}) is

32's	16's	8's	4's	2's	1's
0	1	0	0	1	1

Integer Numbers

- An **`int`** is almost universally stored as a binary number

- When you enter an integer from the keyboard, software converts the ASCII or Unicode characters to a binary integer

What is binary for 5?

A. 0005

B. 0100

C. 0101

D. 0111

E. 1000

Negative Integers

- Almost all systems for storing negative binary numbers set the left most bit or the most significant bit (MSB) to indicate the sign of a number

Common formats:

- Signed Magnitude
- Ones Complement
- Twos Complement (most commonly used)

Signed Magnitude

- Negative number are the same as positive with the sign bit set

Three bit example

000	001	010	011	100	101	110	111
0	1	2	3	-0	-1	-2	-3

- There are two zeroes, positive and negative
- Rarely used for integer numbers
- doubles and floats use signed magnitude

Twos Complement

- Negative number are the logical inverse of positive numbers plus 1

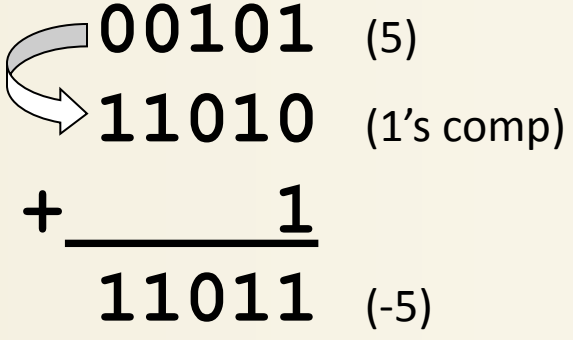
Three bit example

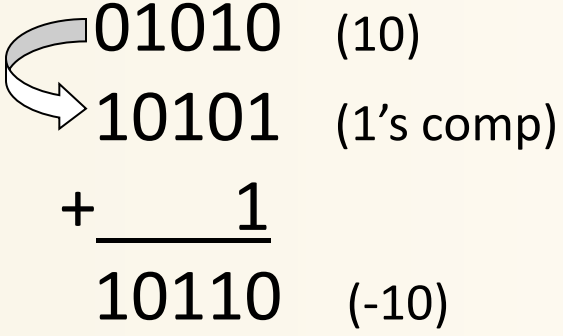
000	001	010	011	100	101	110	111
0	1	2	3	-4	-3	-2	-1

Normal binary arithmetic works for positive and negative numbers

Two's Complement Representation

- If number is positive or zero,
  - normal binary representation, zero in upper bit
- If number is negative,
  - start with positive number
  - flip every bit (i.e., take the one's complement)
  - then add one





Two's Complement Signed Integers

2^3	2^2	2^1	2^0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

2^3	2^2	2^1	2^0	
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

Range of Numbers

- byte from -128 to 127
- short from -32768 to 32767
- int from -2,147,483,648 to 2,147,483,647
- long from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

What is the decimal value of the 4 bit two's complement number?

1011

A. 11

B. 5

C. -11

D. -5

E. -3

2's Complement Addition

- Use normal binary addition regardless of sign.
- Ignore carry out from the sign bit

```
  01101000  (104)        00000111  (7)
+ 11110000  (-16)     +  00000010  (2)
  01011000  (98)         00001001  (9)
```

Assuming 8-bit twos complement numbers.

2's Complement Addition

- Use normal binary addition regardless of sign
- Carry can propagate to the sign bit

```
  0110    (6)
+ 0011    (3)
  1001    (-7)
```

Assuming 4-bit twos complement numbers.

Write an IF statement

- Write an if statement to set the boolean variable ok to true if cow plus 5 is less than 12

```
boolean ok = false;
cow = keyboard.nextInt();
if (          what goes here?          ) {
   ok = true;
}
```

Possible Solution

- Write an if statement to set the boolean variable ok to true if cow plus 5 is less than 12

```
boolean ok = false;
cow = keyboard.nextInt();
if (          cow + 5 < 12          ) {
   ok = true;
}
```

Another Possible Solution

- Write an if statement to set the boolean variable ok to true if cow plus 5 is less than 12

```
boolean ok = false;
cow = keyboard.nextInt();
if (          cow  < 7          ) {
   ok = true;
}
```

Integer Overflow

- The largest value for an int is 2147483647 and the most negative value is -2147483648

2147483647

+5

is

```
 01111111111111111111111111111111
+00000000000000000000000000000101
 10000000000000000000000000000100
```

- The result is negative

Integer Overflow

- When cow is 2147483647, cow + 5 is -2147483644

```
if ( cow + 5 < 12 )// this would be true!!

if ( cow < 7 )          // this would be false
```

- This can be a potential security exploit.

Avoid Integer Overflow Exploits

- To keep a program secure, check that numbers are in the proper range before doing any arithmetic with them
- Use a **`long`** if reasonable values might exceed the size of an **`int`**

Schedule

Monday, November 18 Secure Programming Lab	Wednesday, November 20 Programming practice Quiz Lab	Friday, November 22 review Quiz
Monday, November 25 **Exam 3**	Wednesday, November 27 *Thanksgiving Holiday* *(no classes)*	Friday, November 29 *Thanksgiving Holiday* *(no classes)*
Monday, December 2 Software engineering Lab	Wednesday, December 4 review Final Lab	Final