C++ Security Risks

CMSC 240 Software Systems Development

Today – C++ Security Risks

1. Buffer overflow
2. Integer overflow and underflow
3. Pointer initialization
4. Incorrect type conversion
5. Format string vulnerability

RICHMOND

#1 Buffer Overflow

- Definition: Writing more data to a buffer than it can hold
- Causes: Lack of bounds checking, use of unsafe functions (strcpy, gets, etc.)
- Consequences: Memory corruption, unexpected behavior, security vulnerabilities
- Prevention: Using safe functions (strncpy, snprintf, etc.), bounds checking, stack canaries

#2 Integer Overflow and Underflow

- Definition: Wrapping around the maximum (overflow) or minimum (underflow) value of an integer type
- Causes: Arithmetic operations exceeding the limits of the data type
- Consequences: Incorrect calculations, control flow issues
- Prevention: Checking for overflow/underflow, using larger data types, using libraries for safe arithmetic

#3 Pointer Initialization

- Definition: Setting a pointer to a valid address before use
- Causes: Uninitialized pointers contain garbage values
- Consequences: Unpredictable behavior, crashes, security risks
- Prevention: Always initialize pointers, preferably to `nullptr`

#4 Incorrect Type Conversion

- Definition: Converting data between types in an unsafe manner
- Causes: Implicit conversions, C-style casts
- Consequences: Data corruption, loss of precision, security vulnerabilities
- Prevention: Using C++ style casts (`static_cast`), type-safe conversions

#5 Format String Vulnerability

- Definition: Using user input in format strings without proper sanitization
- Causes: Passing user input directly to `printf` like functions
- Consequences: Information leakage, arbitrary code execution
- Prevention: Never use user input as the format string, always specify format specifiers

Format Specifiers for `printf`

- `%d` or `%i`: Signed decimal integer
- `%u`: Unsigned decimal integer
- `%f`: Decimal floating point
- `%e`: Scientific notation using e
- `%x`: Unsigned hexadecimal integer
- `%s`: Null-terminated string
- `%c`: Character
- `%p`: Pointer address (printed in hexadecimal)

Format Specifiers for `printf`

```
#include <cstdio>

int main()
{
    printf("Number is %d, floating point is %f, hex is %x \n", 123, 3.141592, 255);

    return 0;
}
```

Format Specifiers for `printf`

```cpp
#include <cstdio>
#include <iostream>
using namespace std;

int main()
{
    // C style
    printf("Number is %d, floating point is %f, hex is %x \n", 123, 3.141592, 255);

    // C++ style
    cout << "Number is " << 123 << ", floating point is " << 3.141592 << ", hex is " << hex << 255 << endl;

    return 0;
}
```