# Computer System B -Security

Introduction to Software Vulnerabilities Part 3 Integer Overflow

Sanjay Rawat

# Integer Operation Errors

- Integers are native datatypes in C/C++.
- There are multiple ways to represent numbers
  - `signed int`
  - `unsigned int`
  - `short/long`
  - `etc….`

# Integer Operation Errors

- Integers are native datatypes in C/C++.
- There are multiple ways to represent numbers
  - `signed int`
  - `unsigned int`
  - `short/long`
  - `etc….`
- We can type-cast one to other!

# Integer Operation Errors

- Integers are native datatypes in C/C++.
- There are multiple ways to represent numbers
  - `signed int`
  - `unsigned int`
  - `short/long`
  - `etc….`
- We can type-cast one to other!
- Each can hold values of certain size!

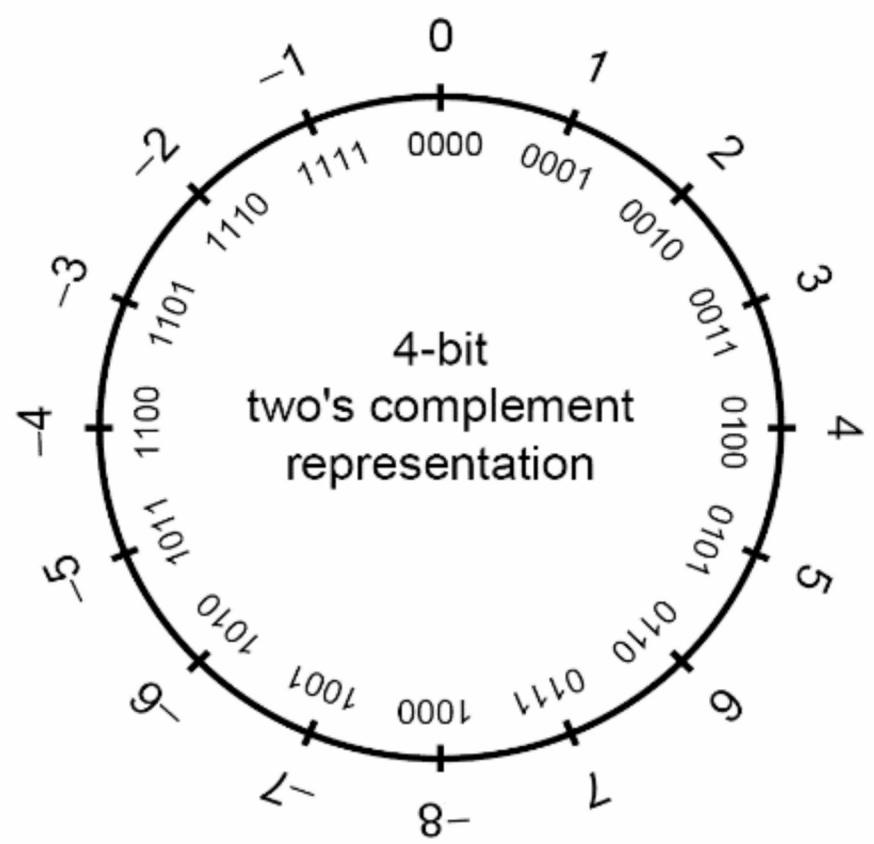
# Unsigned Integers

- Unsigned integer values range from zero to a maximum that depends on the size of the type.
- This maximum value can be calculated as $2^{n-1}$, where n is the number of bits used to represent the unsigned type.
- For each signed integer type, there is a corresponding unsigned integer type.

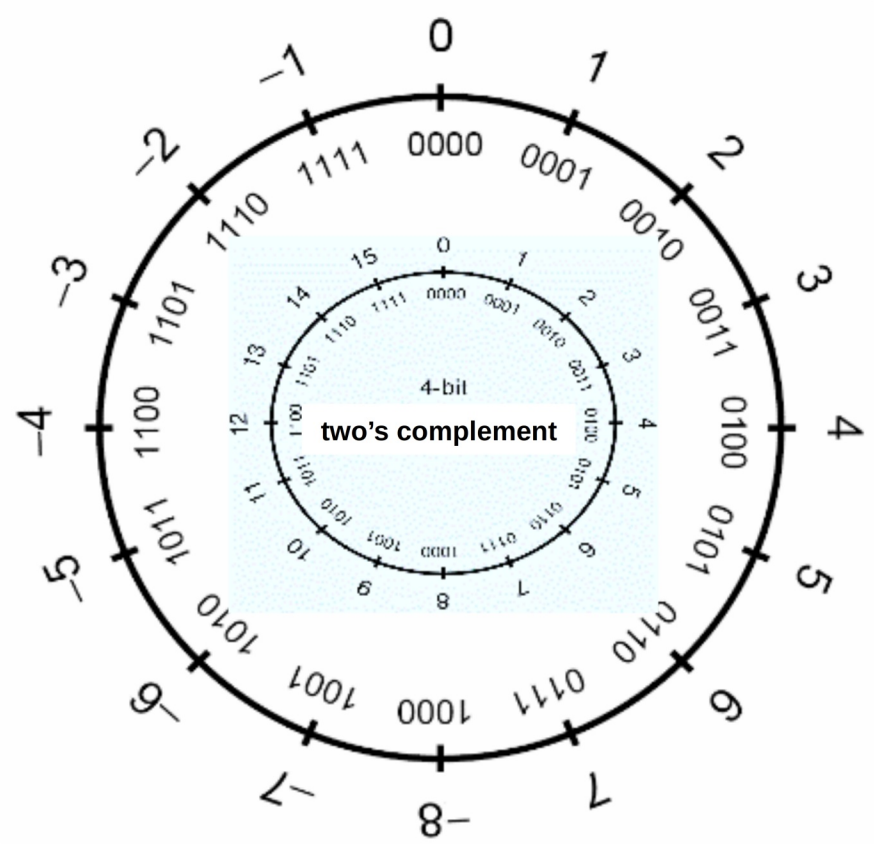
# Signed Integers

- Signed integers are used to represent positive and negative values.
- On a computer using two's complement arithmetic, a signed integer ranges from $-2^{n-1}$ through $2^{n-1}-1$.

# (un)signed Integer Representation



# (un)signed Integer Representation



# Example Integer Ranges

| Type | Storage size | Value range |
|---|---|---|
| char | 1 byte | -128 to 127 or 0 to 255 |
| unsigned char | 1 byte | 0 to 255 |
| signed char | 1 byte | -128 to 127 |
| int | 2 or 4 bytes | -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647 |
| unsigned int | 2 or 4 bytes | 0 to 65,535 or 0 to 4,294,967,295 |
| short | 2 bytes | -32,768 to 32,767 |
| unsigned short | 2 bytes | 0 to 65,535 |
| long | 8 bytes | -9223372036854775808 to 9223372036854775807 |
| unsigned long | 8 bytes | 0 to 18446744073709551615 |

# Unsigned Integer Conversions

- Conversions of smaller unsigned integer types to larger unsigned integer types is
  - always safe
  - typically accomplished by zero-extending the value

- When a larger unsigned integer is converted to a smaller unsigned integer type the
  - larger value is truncated
  - low-order bits are preserved

# Integer Error Conditions 1

- Integer operations can resolve to unexpected values as a result of an
  - overflow
  - sign error
  - truncation

# Overflow

- An integer overflow occurs when an integer is increased beyond its maximum value or decreased beyond its minimum value.
- Overflows can be signed or unsigned

**A signed overflow occurs when a value is carried over to the sign bit**

**An unsigned overflow occurs when the underlying representation can no longer represent a value**

# Overflow Examples 1

```
1. int i;
2. unsigned int j;

3. i = INT_MAX;  // 2,147,483,647
4. i++;
5. printf("i = %d\n", i);

6. j = UINT_MAX; // 4,294,967,295;
7. j++;
8. printf("j = %u\n", j);
```

# Overflow Examples 1

```
1. int i;
2. unsigned int j;

3. i = INT_MAX;  // 2,147,483,647
4. i++;
5. printf("i = %d\n", i);

6. j = UINT_MAX; // 4,294,967,295;
7. j++;
8. printf("j = %u\n", j);
```

```
i=-2,147,483,648
```

# Overflow Examples 1

```
1. int i;
2. unsigned int j;

3. i = INT_MAX;  // 2,147,483,647
4. i++;
5. printf("i = %d\n", i);

6. j = UINT_MAX; // 4,294,967,295;
7. j++;
8. printf("j = %u\n", j);
```

```
i=-2,147,483,648
```

```
j = 0
```

# Truncation Errors

- Truncation errors occur when
  - an integer is converted to a smaller integer type and
  - the value of the original integer is outside the range of the smaller type
- Low-order bits of the original value are preserved and the high-order bits are lost.

# Truncation Error Example

- 1. `char cresult, c1, c2, c3;`
- 2. `c1 = 100;`
- 3. `c2 = 90;`
- 4. `cresult = c1 + c2;`

# Truncation Error Example

- `1. char cresult, c1, c2, c3;`
- `2. c1 = 100;`
- `3. c2 = 90;`
- `4. cresult = c1 + c2;`

Adding **`c1`** and **`c2`** exceeds the max size of **`signed char (+127)`**

# Truncation Error Example

- `1. char cresult, c1, c2, c3;`
- `2. c1 = 100;`
- `3. c2 = 90;`
- `4. cresult = c1 + c2;`

Adding **`c1`** and **`c2`** exceeds the max size of **`signed char (+127)`**

Truncation occurs when the value is assigned to a type that is too small to represent the resulting value

```
int main(int argc, char *argv[]){
            unsigned short s;
            int i;
            char buf[80];
            if(argc < 3){
                    return -1;
            }
            i = atoi(argv[1]);
            s = i;
            if(s >= 80){            /* [w1] */
                    printf("Oh no you don't!\n");
                    return -1;
            }
            printf("s = %d\n", s);
            memcpy(buf, argv[2], i);
            buf[i] = '\0';
            printf("%s\n", buf);
            return 0;
    }
```

# Precondition unsigned

Overflow occurs when **`A`** and **`B`** are **`unsigned int`** and

```
A + B > UINT_MAX
```

# Precondition unsigned

Overflow occurs when **`A`** and **`B`** are **`unsigned int`** and

```
A + B > UINT_MAX
```

To prevent the test from overflowing, code this test as

```
A > UINT_MAX – B
```

Overflow also occurs when **`A`** and **`B are long long int`** and

```
A + B > ULLONG_MAX
```