Understanding C Integer Boundaries (overflows & underflows)

By Saif El-Sherei
www.elsherei.com

Introduction:

I decided to get a bit more into Linux exploitation, so I thought it would be nice if I document this as a good friend once said " you think you understand something until you try to teach it" This is my first try at writing papers. This paper is my understanding of the subject. I understand it might not be complete I am open for suggestions and modifications. I hope as this project helps others as it helped me. This paper is purely for education purposes.

Data Storage:

The C standards define an object as a region in memory in the data execution environment; its contents can represent values and each object has a certain type.

Integer types (including char types) represents different integer sizes that can be mapped to an architecture dependent data type. Integer types have certain value ranges to be able to represent them. They are largely dependent on the architecture and the compiler.

Signed integer type represent both positive and negative values. Unsigned on the other hand can represent only positive values. Each signed data type have its corresponding unsigned integer type that take the same amount of storage.

The unsigned integer type have two possible types of bits. The value bits which represents the value and the padding bits. The signed integer type has three types of bits; the value bits, the padding bits, & the sign bit which represents the sign associated with the integer type. Each integer type has a precision & width. The precision is the number of value bits, & the width is the number of bits the type uses to represent its value including value bits, padding bits, & sign bit.

Integer types are represented in memory in binary format which is a two's compliment representation.

Sizes and ranges for 32-bit integer types:

Type	Width (in Bits)	Minimum Value	Maximum Value
signed char	8	-128	127
unsigned char	8	0	255

short	16	-32,768	32,767
unsigned short	16	0	65,535
Int	32	-2,147,483,648	2,147,483,647
unsigned int	32	0	4,294,967,295
long	32	-2,147,483,648	2,147,483,647
unsigned long	32	0	4,294,967,295
long long	64	-9,223,372,036,854,770,000	9,223,372,036,854,770,000
unsigned long long	64	0	18,446,744,073,709,500,000

Arithmetic Boundary conditions:

32-bit Integer types can hold certain ranges of values. So what happens when we try to traverse this boundary? Simple arithmetic on the value like addition, multiplication, and subtraction. Can result in a value that can't be represented.

So if we have two unsigned integer types each with the value of 2147483648 (a & b).

a+b = 4294967296 which is larger than the maximum value that can be represented in an unsigned integer type. This is called an integer overflow.

Let's take a look at another example
Unsigned int a,b;
a=0
b=a-1
The value of b is -1 which is below than the minimum possible value that can be stored this is called an integer underflow.

Unsigned Integer Boundaries:

Unsigned integers are defined in the c specification as being subject to modular arithmetic. For an unsigned integer with width of X bits. Arithmetic on that unsigned integer is done modulo $2^{x.}$ Arithmetic operations on a 16-bit unsigned integer is performed modulo 2^{16}.

In modulus arithmetic for example (x % 5) if x value is larger than 4 (max value) it will wrap around to 0. And if x value is less than 0 it will wrap around the max. Value of 4.

Unsigned int x = 3758096416;
x=x+536870944

The result of the above calculation is 4294967360which is above the maximum value which an unsigned int can hold of 4294967295. But because of the modular arithmetic nature of unsigned int arithmetic functions the result of the above calculation is 4294967360 % 4294967296 which is 64.

Now let's turn to the integer underflow
Unsigned int x = 8;
x= 8 - 16

The result of the above calculation is -8 and after utilizing the modulus nature of the unsigned integer -8 % 4294967296 = -8. Since the unsigned integer can only hold a minimum value of 0. Because of unsigned integers are subject to modular arithmetic. So -8 will wrap around the minimum value to the maximum value which is 4294967295 and the result will be equal to 4294967287.

Signed integer Boundaries:

Signed integers are slightly different. Remember the signed integer two's compliment representation in binary will have value, padding, & sign bits. The sign bit represents the signess of the integer 0 for positive and 1 if the number is negative. When an overflow or underflow condition occurs on signed integers the result will wrap around the sign and causes a change in sign. For example a 32 bit number 2147483647 = 0x7FFFFFFF in hex. If we add 1 to this number it will be 0x80000000 which is equivalent to -2147483648 decimal.

With signed addition or subtraction, you can overflow the sign boundary by causing a positive number to wrap around 0x80000000 and become a negative number. You can also underflow the sign boundary by causing a negative number to wrap below 0x80000000 and become a positive number.