Arithmetic & Integer Boundaries

Integer Ranges

- Maximum and Minimum Values for Integers

	8-bit	16-bit	32-bit	64-bit
Signed	-2^7 to $2^7 - 1$ -128 to 127	-2^{15} to $2^{15} - 1$ -32768 to 32767	-2^{31} to $2^{31} - 1$	-2^{63} to $2^{63} - 1$
Unsigned	0 to $2^8 - 1$	0 to $2^{16} - 1$	0 to $2^{32} - 1$	0 to $2^{64} - 1$

Typical Sizes & Ranges for 32 bit Integer Types

Type	Width (in bits)	Min Value	Max Value
signed char	8 (1 byte)	-128	127
unsigned char	8 (1 byte)	0	255
Short	16 (2 bytes)	-32768	2^{15}- 1 (32767)
unsigned short	16 (2 bytes)	0	2^{16}- 1 (65535)
int	32 (4 bytes)	-2147483648	2^{31} (2147483647)
unsigned int	32 (4 bytes)	0	2^{32} - 1 (4294967295)
long	32 (4 bytes)	-2147483648	2^{31}- 1 (2147483647)
unsigned long	32 (4 bytes)	0	2^{32} - 1 (4294967295)
long long	64 (8 bytes)	-2^{63}	2^{63}+1
unsigned long long	64 (8 bytes)	0	2^{64} - 1

Arithmetic Boundaries – Unsigned

- Overflow
  - unsigned int's max value is 4294967295, thus the code below will result in overflow

```
unsigned int a;
a=3758096416;
a=a+536870944;
// a should be 4294967360
```

- Underflow
  - unsigned int's min value is 0, thus the code below will result in underflow

```
unsigned int a;
a=0;
a=a-1;
// a should be -1
```

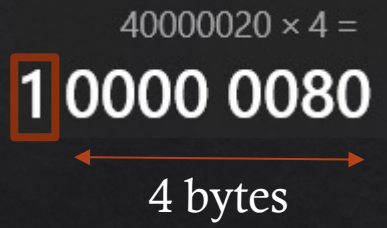
Arithmetic Boundaries – Unsigned Example

- OpenSSH 3.1

```
u_int nresp; // user controlled, num responses to expect
nresp = packet_get_int();
if (nresp > 0)
{
    response = xmalloc(nresp * sizeof(char*));
    for (i = 0; i < nresp; i++)
        response[i] = packet_get_string(NULL);
}
```

What is the problem here?
Func def:
xmalloc(size_t)

- sizeof(char*) == 4 (32bit systems)
- If nresp is a large enough, then nresp * 4 may overflow unsigned int's range
- Eg: if nresp == 0x40000020, nresp * 4 == 0x80
- "response" will be allocated with size 0x80
- However the loop will execute 0x40000020 times
- Resulting in an OOB write



Arithmetic Boundaries – Signed

- When an overflow or underflow occurs, the resulting value wraps around the sign boundary and can cause a change in sign
- Eg 0x7FFFFFFF => 2147483647, 0x7FFFFFFF + 0x1 = 0x80000000 => -2147483648

```
int a;
a=0x7ffffff0;
a=a+0x100;
```

- Consider this program, 0x7FFFFFF0 (+ve) + 0x100 (+ve) = 0x800000F0 (2147483888)
- However, as this value overflows int's range, it will be wrap around and become -0x7fffff10 (-2147483408)

Arithmetic Boundaries – Signed Example

- What is the problem here?

```
char *read_data(int sockfd)
{
    char *buf;
    int length = network_get_int(sockfd); // user controlled
    if (!(buf = (char*)malloc(MAXCHARS)))
        quit();

    if (length < 0 || length + 1 >= MAXCHARS)
        quit();

    if (read(sockfd, buf, length) <= 0)
        quit();

    buf[length] = '\0';
    return buf;
}
```

The code checks if length is +ve and less than MAXCHARS

However "length + 1" can cause length to overflow and become a negative value

Thus, if length == 0x7FFFFFFF, this check will be bypassed, thus passing to read:

buf – created with MAXCHARS

length – largest int value

Therefore read() will read a large amount from the socket into the buffer, resulting in overwrite

eg length +1 ➔ 0x7FFFFFFF + 0x1 = 0x80000000 ➔ negative value, which is *not* >= MAXCHARS

Arithmetic Boundaries – Signed Example

- MAXCHARS = 0xffff (65535)
- a = 0x7fffffff (2147483647), which is larger than MAXCHARS
- However, due to the +1, it overflows and becomes -ve so “here” is not printed

```c
// Online C compiler to run C program online
#include <stdio.h>

int main() {
    int MAXCHARS = 0xFFFF;
    int a = 0x7FFFFFFF;
    if (a < 0 || a + 1 >= MAXCHARS)
        printf("here");
    else
        printf("there");
    return 0;
}
```

```
/tmp/OlC
there
```

Arithmetic Boundaries – Overflow Example

- JPEG code handling comments. What is the problem?

```
void getComment(unsigned int len, char *src)
{
    unsigned int size;
    size = len - 2;
    char *comment = (char *)malloc(size + 1);
    memcpy(comment, src, size);
    return;
}

int _tmain(int argc, _TCHAR* argv[])
{
    getComment(1, "Comment "); // call getComment with len == 1
    return 0;
}
```

getComment is called with 1

size will be -1 (or 0xFFFFFFFF)
This will be interpreted as a positive integer as size is unsigned int

malloc will be called with
0xFFFFFFFF + 1 ➔ 0x0 (overflow!)

memcpy will be called with 0xFFFFFFFF as the size, resulting in OOB write

Further Reading

- http://www.sis.pitt.edu/jjoshi/courses/IS2620/Spring13/Lecture7.pdf