Secure Coding

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Secure Coding: Insecure Functions

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Secure Coding: Insecure Functions

http://stackoverflow.com/questions/2565727/what-are-the-c-functions-from-the-standard-library-that-must-should-be-avoided

Functions which can create a buffer overflow:

- gets(char *s)
- scanf(const char *format, ...)
- sprintf(char *str, const char *format, ...)
- strcat(char *dest, const char *src)
- strcpy(char *dest, const char *src)

Secure Coding: Insecure Functions

Recap:

- Don’t use functions which do not respect size of destination buffer

C Strings

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

C Strings

Strings in C:

```
Byte 0 to (n-1): String
Byte n         : \0
```

Strings in Pascal:

```
Byte 0         : Length of string (n)
Byte 1 to (n+1): String
```

C Strings

Threrefore:

```
char str[8];
strcpy(str, "1234567"); // str[7] = '\0'
strlen(str);            // 7

strcpy(str, "12345678");  // str[7] = '8'
                          // str[8] = '\0'
strlen(str);              // 8

strcpy(str, "123456789");   // str[7] = '8'
                            // str[8] = '9'
                            // str[8] = '\0'
strlen(str);                // 9
```

Overflow for input strings which are too large

```
strcpy(str, "1234567"); // str[7] = '\0'
strlen(str);            // 7

strcpy(str, "12345678");  // str[7] = '8'
                          // str[8] = '\0'
strlen(str);              // 8

strcpy(str, "123456789");   // str[7] = '8'
                            // str[8] = '9'
                            // str[8] = '\0'
strlen(str);                // 9
```

C Strings

Threrefore:

```
char str[8];
strcpy(str, "1234567"); // str[7] = '\0'


strncpy(str, "1234567", 8); // str[7] = '\0'

strncpy(str, "12345678", 8); // str[7] = '7'

strncpy(str, "123456789", 8); // str[7] = '7'
```

No null terminator if input string is too large (>=dest_len)

0’

```
strncpy(str, "1234567", 8); // str[7] = '\0'

strncpy(str, "12345678", 8); // str[7] = '7'

strncpy(str, "123456789", 8); // str[7] = '7'
```

C Strings

Using standard C string functions on strings with missing \0 terminator is bad

```
char str1[8];
char str2[8];

strncpy(str1, "XXXXYYY", 8);
strncpy(str2, "AAAABBBB", 8);
```

Result: (strlen, printf)

```
Len str1: 7
Len str2: 15
str1: XXXXYYY
str2: AAAABBBBXXXXYYY
```

C Strings

How to do it correctly:

```
strncpy(str2, "AAAABBBB", 8);
str2[7] = "\0";
```

Secure Coding: Integer Overflow

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Integer Overflow: example 1

Using int (= signed int) when you should use unsigned int

Integer Overflow: example 1

```
void test3(int inputLen) {
        char arr[1024];
        printf("Input len : %i / 0x%x\n", inputLen, inputLen);

        if (inputLen > 1024) {
                printf("Not enough space\n");
                return;
        }
        printf("Ok, copying...\n");
}
```

Integer Overflow: example 1

```
void test3(int inputLen) {
        char arr[1024];
        printf("Input len : %i / %u / 0x%x\n",
                inputLen, inputLen, inputLen);
        if (inputLen > 1024) {
```

test3(**0x7fffffff**);

Input len : **2147483647** / 2147483647 / 0x7fffffff

Not enough space

test3(**0x80000000**);

Input len : **-2147483648** / 2147483648 / 0x80000000

Ok, copying...

Integer Overflow: example 1

Integer overflow problem:

Programs:

- Usually use “unsigned int”
- Indexes should be “unsigned int” (cannot be <0)
- malloc() takes a size_t (unsigned int)

Developers:

- Usually use “signed int”
- Don’t want to type “unsigned…”
- Don’t understand size_t
- Want to communicate error: if(result < 0) { }

Integer Overflow: example 2

"Adding a positive number to an integer might make it smaller"

If you add a positive integer to another positive integer, the result is truncated. Technically, if you add two 32-bit numbers, the result has 33 bits.

On the CPU level, if you add two 32-bit integers, the lower 32 bits of the result are written to the destination, and the 33rd bit is signalled out in some other way, usually in the form of a "carry flag".

Integer Overflow: example 2

```
int catvars(char *buf1, char *buf2, unsigned int len1,
  unsigned int len2)
{
  char mybuf[256];

  if((len1 + len2) > 256){     /* [3] */
      return -1;
  }

  memcpy(mybuf, buf1, len1);        /* [4] */
  memcpy(mybuf + len1, buf2, len2);

  do_some_stuff(mybuf);
```

Integer Overflow: example 2

```
len1: 260  / 260          / 0x104
len2: -4   / 4294967292 / 0xfffffffc

len1 + len2: 256 / 256 / 0x100
```

```
if((len1 + len2) > 256){     /* [3] */
    return -1;
}

memcpy(mybuf, buf1, len1);        /* [4] */
memcpy(mybuf + len1, buf2, len2);

do_some_stuff(mybuf);
```

Integer overflows

Other types of overflows:

Different sizes of types

- x32, x64, x32@x64, ARM, …

Integer overflows

C types

Type specifier	Equivalent type	Width in bits by data model				
		C++ standard	LP32	ILP32	LLP64	LP64
short	short int	at least 16	16	16	16	16
short int						
signed short						
signed short int						
unsigned short	unsigned short int					
unsigned short int						
int	int	at least 16	16	32	32	32
signed						
signed int						
unsigned	unsigned int					
unsigned int						
long	long int	at least 32	32	32	32	64
long int						
signed long						
signed long int						
unsigned long	unsigned long int					
unsigned long int						
long long	long long int (C++11)	at least 64	64	64	64	64
long long int						
signed long long						
signed long long int						
unsigned long long	unsigned long long int (C++11)					
unsigned long long int						

http://en.cppreference.com/w/cpp/language/types

Integer Overflow

Integer overflow 1:

```
int myfunction(int *array, int len){
    int *myarray, i;

    myarray = malloc(len * sizeof(int));  /*[1]*/
    if(myarray == NULL){
        return -1;
    }

    for(i = 0; i < len; i++){                /*[2]*/
        myarray[i] = array[i];
    }

    return myarray
}
```

Compiler Warnings

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Compiler Warnings

Anti-Formatstring Vulnerability:

Compiler:

```
-Wformat-security
```

Code:

```
printf(argv[1]);
```

Warning:

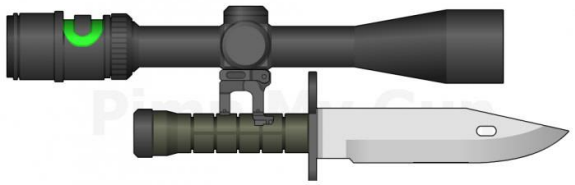
```
warning: format not a string literal and no
format arguments [-Wformat-security]
```

Compiler Warnings

Make warnings into errors:

```
-Werror
```

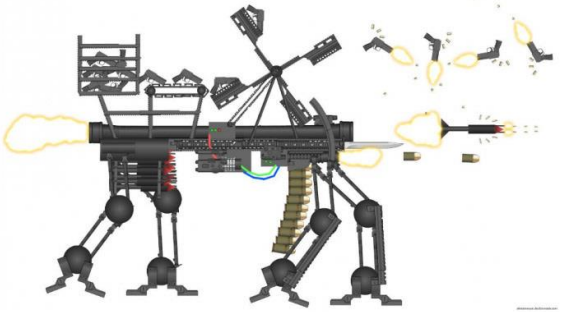
Assembly



C

C++

Python



References

References:

Catching Integer Overflows in C

- https://www.fefe.de/intof.html