Bits, Bytes, Integers
Case studies

Arquitectura de Computadores

Departamento de Engenharia Informática

Instituto Superior de Engenharia do Porto

Luís Nogueira (lmn@isep.ipp.pt)

Security vulnerability in *getpeername*

In 2002, programmers involved in the FreeBSD open source operating systems project realized that their implementation of the *getpeername* library function had a security vulnerability. A simplified version of their code went something like this:

```
1   /*
2   * Illustration of code vulnerability similar to that found in
3   * FreeBSD's implementation of getpeername()
4   */
5
6   /* Declaration of library function memcpy */
7   void *memcpy(void *dest, void *src, size_t n);
8
9   /* Kernel memory region holding user-accessible data */
10  #define KSIZE 1024
11  char kbuf[KSIZE];
12
13   /* Copy at most maxlen bytes from kernel region to user buffer */
14   int copy_from_kernel(void *user_dest, int maxlen) {
15      /* Byte count len is minimum of buffer size and maxlen */
16      int len = KSIZE < maxlen ? KSIZE : maxlen;
17      memcpy(user_dest, kbuf, len);
18      return len;
19   }
```

In this code, we show the prototype for library function memcpy on line 7, which is designed to copy a specified number of bytes *n* from one region of memory to another.

The function *copy_from_kernel*, starting at line 14, is designed to copy some of the data maintained

by the operating system kernel to a designated region of memory accessible to the user. Most of the data structures maintained by the kernel should not be readable by a user, since they may contain sensitive information about other users and about other jobs running on the system, but the region shown as *kbuf* was intended to be one that the user could read. The parameter maxlen is intended to be the length of the buffer allocated by the user and indicated by argument *user_dest*. The computation at line 16 then makes sure that no more bytes are copied than are available in either the source or the destination buffer.

Suppose, however, that some malicious programmer writes code that calls *copy_from_kernel* with a negative value of *maxlen*. Then the minimum computation on line 16 will compute this value for *len*, which will then be passed as the parameter *n* to *memcpy*. Note, however, that parameter *n* is declared as having data type *size_t*. This data type is declared (via *typedef*) in the library file *stdio.h*. Typically it is defined to be *unsigned int* on 32-bit machines. Since argument *n* is unsigned, *memcpy* will treat it as a very large, positive number and attempt to copy that many bytes from the kernel region to the user's buffer. Copying that many bytes (at least 2^{31}) will not actually work, because the program will encounter invalid addresses in the process, but the program could read regions of the kernel memory for which it is not authorized.

We can see that this problem arises due to the mismatch between data types: in one place the length parameter is signed; in another place it is unsigned. Such mismatches can be a source of bugs and, as this example shows, can even lead to security vulnerabilities. Fortunately, there were no reported cases where a programmer had exploited the vulnerability in FreeBSD. They issued a security advisory, "FreeBSD-SA-02:38.signed-error," advising system administrators on how to apply a patch that would remove the vulnerability. The bug can be fixed by declaring parameter *maxlen* to *copy_from_kernel* to be of type size_t, to be consistent with parameter *n* of *memcpy*. We should also declare local variable *len* and the return value to be of type size_t.

Security vulnerability in the XDR library

In 2002, it was discovered that code supplied by Sun Microsystems to implement the XDR library, a widely used facility for sharing data structures between programs, had a security vulnerability arising from the fact that multiplication can overflow without any notice being given to the program.

Code similar to that containing the vulnerability is shown below:

```
1  /*
2  * Illustration of code vulnerability similar to that found in
3  * Sun's XDR library.
4  */
5  void* copy_elements(void *ele_src[], int ele_cnt, size_t ele_size) {
6        /*
7        * Allocate buffer for ele_cnt objects, each of ele_size bytes
8        * and copy from locations designated by ele_src
```

```
9       */
10      void *result = malloc(ele_cnt * ele_size);
11      if (result == NULL)
12              /* malloc failed */
13              return NULL;
14      void *next = result;
15      int i;
16      for (i =0;i<ele_cnt;i++){
17              /* Copy object i to destination */
18              memcpy(next, ele_src[i], ele_size);
19              /* Move pointer to next memory region */
20              next += ele_size;
21      }
22      return result;
23  }
```

The function *copy_elements* is designed to copy *ele_cnt* data structures, each consisting of *ele_size* bytes into a buffer allocated by the function on line 10. The number of bytes required is computed as *ele_cnt* * *ele_size*.

Imagine, however, that a malicious programmer calls this function with *ele_cnt* being 1,048,577 (2^{20} + 1) and *ele_size* being 4,096 (2^{12}). Then the multiplication on line 10 will overflow, causing only 4096 bytes to be allocated, rather than the 4,294,971,392 bytes required to hold that much data. The loop starting at line 16 will attempt to copy all of those bytes, overrunning the end of the allocated buffer, and therefore corrupting other data structures. This could cause the program to crash or otherwise misbehave.

The Sun code was used by almost every operating system, and in such widely used programs as Internet Explorer and the Kerberos authentication system. The Computer Emergency Response Team (CERT), an organization run by the Carnegie Mellon Software Engineering Institute to track security vulnerabilities and breaches, issued advisory "CA-2002-25," and many companies rushed to patch their code. Fortunately, there were no reported security breaches caused by this vulnerability.

A similar vulnerability existed in many implementations of the library function *calloc*. These have since been patched.