Sourceanalyzer

The program *sourceanalyzer* analyzes other programs for vulnerabilities. This is a very brief explanation of its output. For more detail, see the documentation in */usr/local/FortifySoftware/SCAS-TE/Documentation*.

The Program

This C program copies a string into buffer and quits. It's clearly a demonstration program!

```
#define MAX_SIZE 128

void doMemCpy(char* buf, char* in, int chars) {
  memcpy(buf, in, chars);
}

int main() {
  char buf[64];
  char in[MAX_SIZE];
  int bytes;

  printf("Enter buffer contents:\n");
  read(0, in, MAX_SIZE-1);
  printf("Bytes to copy:\n");
  scanf("%d", &bytes);

  doMemCpy(buf, in, bytes);

  return 0;
}
```

It has a couple of security problems, were it to be installed setuid and set so anyone could run it. Can you find them before going any further?

The Analysis

We run the *sourceanalyzer* program over this program, as follows:

```
sourceanalyzer gcc stackbuffer.c
```

You will have to set your search path to look in the directory */usr/local/FortifySoftware/SCAS-TE*, of course. Here is the output:

```
[/usr/local/FortifySoftware/SCAS-TE/Samples/basic/stackbuffer]

[BB73F23E46159FBE5ED3C1968C046828 : low : Unchecked Return Value : semantic ]
stackbuffer.c(13) : read()

[EDACF5BD763B329C8EE8AA50F8C53D08 : high : Buffer Overflow : data flow ]
stackbuffer.c(4) :  ->memcpy(2)
    stackbuffer.c(17) :  ->doMemCpy(2)
    stackbuffer.c(15) : <- scanf(1)
```

The analyzer has identified two poor programming practices that may lead to security problems.

1. The function *read* at line 13 returns a value that is not checked. The danger from this is low. It is a semantic problem, that is, it results from the semantics of *read* returning a value.
2. On line 15 of the program, the function *scanf* reads something into its second argument (the first argument in a parameter list is argument 0, so argument 1 is the second one). The arrow "<-" means "input". This quantity is than passed to the function *doMemCpy* as argument 2, the call occurring on line 17. The arrow "->" means "passed to". This argument is then passed to the function *memcpy* on line 4, as the third argument. This means that an input number controls how many bytes *memcpy* copies, and if set incorrectly could cause a buffer overflow.

Potential Exploits

Given these problems, let's see how exploits might work.

1. ***Unchecked return value***
   This is marked "low", so it will be difficult to find a security flaw from it. Basically, it requires that the *read* system call on line 13 of *stackbuffer.c* either fail (hence returning #1) or fewer characters than typed. In that case, the number entered will be larger than the number of characters read, which could cause a problem. The word "semantic" means that the irregularity is from the semantics of the call (that is, no return value used).
2. ***Buffer overflow***
   This is marked "high" because the source code analyzer asserts it is easy to exploit. It This indicates that user input enters the program through the *scanf* call on line 15, which reads data into argument 1. (Arguments are 0-indexed, so argument 1 is the second argument to *scanf*, *&bytes*.) This data is then passed as argument 2 to *doMemCpy*, which in turn sends the data to argument 2 of *memcpy*. This allows a user to cause an arbitrary amount of data to be written to the 64-byte buffer buf, potentially overflowing that buffer.