Universität des Saarlandes
Lehrstuhl für Softwaretechnik
Prof. Andreas Zeller, Stephan Neuhaus
Design of Secure Software Systems
Exercise 7

Total number of points obtainable: 12, plus 4 bonus points.

1 Attacking `access()`

You can do the following exercise only if you have access to the root account on a Linux (or Unix) machine. Since not everybody has access to a Linux root account, it's an *optional* exercise that doesn't count for the final exam *at all*, not even with bonus points. Nevertheless, it's a fun thing to try.

The following code can be downloaded at `http://www.st.c.uni-sb.de/edu/secdesign/races.c`. Compile the program and install it suid root. See if you can make it print `/etc/shadow` or any other file to which you normally have no access.

```c
/* Program with a race condition for you to exploit.
 *
 * Save as "race-2.c" and compile with "gcc -O -Wall -o race-2 race-2.c".
 * Become root, then do "chown root race-2; chmod 4755 race-2".
 *
 * Happy cracking!
 *
 * Stephan Neuhaus
 */
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, const char* argv[]) {
  if (argc == 1) {
    fprintf(stderr, "usage: %s file\n", argv[0]);
    return 1;
  } else if (access(argv[1], R_OK) == 0) {
    int fd = open(argv[1], O_RDONLY); /* Can't fail, we checked */
    unsigned char* buf = malloc(1024);

    if (buf != 0) {
      ssize_t bytes_read = read(fd, buf, 1024);
      while (bytes_read > 0) {
        write(1, buf, bytes_read);
        bytes_read = read(fd, buf, 1024);
      }
      return 0;
    } else {
      fprintf(stderr, "%s: can't malloc\n", argv[0]);
      perror(argv[0]);
      return 1;
    }
  } else {
    fprintf(stderr, "%s: you can't access \"%s\"\n", argv[0], argv[1]);
    perror(argv[0]);
    return 1;
  }
}
```

Mode	Umask	Actual Bits
0644	0077	0600
0644	0022	0644
0777	0077	0700
0777	0022	0755

Table 1: Effects of various umask settings.

2 The Umask

The *umask*(2) system call sets a process's *file creation mask*. The umask is thus an attribute of a process's state. When a process uses a system call like *open*(2) or *creat*(2) to create a file, the call takes the file's access mode bits (see the lecture on security mechanisms) as an argument. From this argument, all the bits that are set in the process's umask are turned off before the file is created. See Table 1 for some examples.

The umask is inherited from the parent process. You can set this value in your shell by using the *umask*(1) shell built-in command or the *umask*(2) system call.

- **(2 points)** Explain why the lecture recommends a umask of 0066 when using *fopen*(3).
- **(4 points)** Implement the strategy for creating temporary files that was outlined in the lecture.
- **(3 points)** In the temporary file creation strategy, is the use of *unlink*(2) a possible security hole? If so, outline an exploit. If not, argue why an exploit would fail. (**Hint:** Consider the permissions on `/tmp`, as described in the lecture on security mechanisms.)

3 Waiting Times

(3 points) In the lecture, we said in the context of lockfiles that if a process fails to acquire a resource, it should try again after waiting for some time. This exercise explores some consequences of this scheme.

A resource allocation scheme is called *fair* if every process that wants a resource eventually gets it. Assume that processes acquire resources by using lock files. Also assume that all processes are programmed to use the resource only for a short period of time and that the lockfiles are immediately released. Is this scheme necessarily fair? If it is, prove it. If it is not, give a counterexample.

4 Elevated Privileges

(4 bonus points) The very first lecture in this course had a code example from the Unix `lpr` utility. This utility had a '`-r`' switch which caused the file to be removed after printing. The `lpr` utility ran suid root and used *access*(2) to check whether the user was allowed to remove the file and removed it only when the check succeeded. We said then that this was susceptible to a TOCTOU attack. Now you know why. Suggest a way to fix the problem.