# 竞态条件漏洞



S7.1. 下列 Set-UID 程序是否存在竞争条件漏洞？

```
if (!access("/etc/passwd", W_OK)) {
    /* the real user has the write permission*/
    f = open("/tmp/X", O_WRITE);
    write_to_file(f);
}
else {
   /* the real user does not have the write permission */
   fprintf(stderr, "Permission denied\n");
}
```

S7.2. 假设新增系统调用 faccess(int fd, int mode)，语义与 access() 相同，但通过文件描述符 fd 指定被检查文件。下列程序是否存在竞争条件？

```
int f = open("/tmp/x", O_WRITE);
if (!faccess(f, W_OK)) {
    write_to_file(f)
} else {
    close(f);
}
```

S7.3. 在下列程序中，攻击者必须赢得多少个竞争窗口？

```
int main()
{
   struct stat stat1, stat2;
   int fd1, fd2;

   if (access("/tmp/XYZ", O_RDWR)) {
      fprintf(stderr, "Permission denied\n");
      return -1;
   }
   else fd1 = open("/tmp/XYZ", O_RDWR);
```

```
   if (access("/tmp/XYZ", O_RDWR)) {
      fprintf(stderr, "Permission denied\n");
      return -1;
   }
   else fd2 = open("/tmp/XYZ", O_RDWR);

   The program then checks whether fd1 and fd2 refer to
   the same file, if so, the program will write to
   fd1 (or fd2). Otherwise, the program will do nothing
   and exit.
}
```

S7.4. open() 在真正打开文件前也要检查调用者是否有相应权限，看起来同样是“先检查、后使用”。这会产生竞争条件吗？

S7.5. 最小权限原则能有效缓解本章的竞争条件攻击。能否用同样方法防御缓冲区溢出：在调用易受攻击函数前暂时关闭 root 权限，函数返回后再恢复？为什么？

S7.6. 下列由 root 拥有的 Set-UID 程序需要写文件，但希望确保文件属于调用它的用户。程序用 stat() 取得所有者 ID，并与进程真实用户 ID 比较，不一致就退出。程序是否存在竞争条件？若存在，应如何利用？可查阅 stat() 手册。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int main()
{
  struct stat statbuf;
  uid_t real_uid;
  FILE* fp;

  fp = fopen("/tmp/XYZ", "a+");
  stat("/tmp/XYZ", &statbuf);

  printf("The file owner's user ID: %d\n", statbuf.st_uid);
  printf("The process's real user ID: %d\n", getuid());

  // Check whether the file belongs to the user
  if (statbuf.st_uid == getuid()) {
```

```
    printf("IDs match, continue to write to the file.\n");
    // write to the file ...
    if (fp) fclose(fp);
  } else {
    printf("IDs do not match, exit.\n");
    if (fp) fclose(fp);
    return -1;
  }

  return 0;
}
```

S7.7. 下列程序把 `stat()` 改为 `fstat()`，通过 `fileno(fp)` 取得文件描述符后检查所有者。它是否仍有竞争条件？若有，应如何利用？可查阅 `fstat()` 和 `fileno()` 手册。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int main()
{
  struct stat statbuf;
  uid_t real_uid;
  FILE* fp;

  fp = fopen("/tmp/XYZ", "a+");
  fstat(fileno(fp), &statbuf);

  printf("The file owner's user ID: %d\n", statbuf.st_uid);
  printf("The process's real user ID: %d\n", getuid());

  // Check whether the file belongs to the user
  if (statbuf.st_uid == getuid()) {
    printf("IDs match, continue to write to the file.\n");
    // write to the file ...
    if (fp) fclose(fp);
  } else {
    printf("IDs do not match, exit.\n");
```

```
    if (fp) fclose(fp);
    return -1;
  }

  return 0;
}
```

S7.8. 若能在检查到使用的整个窗口内锁住文件，其他进程就无法修改它，似乎可以消除竞争。为什么本章不采用文件锁来解决这类问题？

S7.9. 下列特权 Set-UID 程序是否存在竞争条件？若存在，攻击窗口在哪里？应如何利用？

```
1   filename = "/tmp/XYZ";
2   fd = open (filename, O_RDWR);
3   status = access (filename, W_OK);
    ...
    ... (code omitted) ...
    ...
10  if (status == ACCESS_ALLOWED) {
11     write_to_file(fd);
12  } else {
13     fprintf(stderr, "Permission denied\n");
14  }
```

S7.10. 使用最小权限原则修复下列程序中的竞争条件问题。

```
if (access("/tmp/XYZ", W_OK) == ACCESS_ALLOWED) {
    f = open("/tmp/XYZ", O_WRITE);
    write_to_file(f);
}
else {
   fprintf(stderr, "Permission denied\n");
}
```