CE-248 Lab Race Condition

Vitor Pimenta dos Reis Arruda

Março de 2019

1 7.1

Não tem. Se o teste do **access()** passar, é porque o RUID já tem permissão para escrever em **/etc/passwd**, ou seja, já é um usuário privilegiado. Não é possível, sendo um usuário não-privilegiado, "enganar" o programa para que o teste de **access()** recaia sobre outro arquivo (já que a string está, como dizem, **hard-coded** no código-fonte do programa).

2 7.2

Reproduzimos abaixo o código da questão (note que está reformatado !) com numeração de linhas para facilitar referências (assumimos que **tmp/XYZ** é um erro de digitação e deveria ser **/tmp/XYZ**):

```
1  int main() {
2    struct stat stat1, stat2;
3    int fd1, fd2;
4    if (access("/tmp/XYZ", O_RDWR)) {
5      fprintf(stderr, "Permission denied\n");
6      return -1;
7    } else
8      fd1 = open("/tmp/XYZ", O_RDWR);
9    if (access("/tmp/XYZ", O_RDWR)) {
10     fprintf(stderr, "Permission denied\n");
11     return -1;
12   } else
13     fd2 = open("/tmp/XYZ", O_RDWR);
14   // Check whether fd1 and fd2 have the same inode.
15   fstat(fd1, &stat1);
16   fstat(fd2, &stat2);
17   if (stat1.st_ino == stat2.st_ino) {
18     write_to_file(fd1);
19   } else {
20     fprintf(stderr, "Race condition detected\n");
21     return -1;
22   }
23   return 0;
24 }
```

Premissoas para responder à questão são que **access()** usa o RUID (**real user id**) para testar permissão, enquanto **open()** usa o EUID para tal.

Assim sendo, há uma condição de corrida "óbvia" entre as linhas 4 e 8. Outra menos óbvia está entre as linhas 8 e 9: na linha 8, **/tmp/XYZ** deveria ser um symlink apontando para um arquivo privilegiado, enquanto, na linha 9, deveria apontar para um arquivo não-privilegiado. Finalmente, há outra condição de corrida "óbvia" entre as linhas 9 e 13. Ao todo, portanto, são 3 condições de corrida.

3 7.3

Não há condição de corrida, porque a chamada a **open()** é atômica. Isso significa que, uma vez invocada **open()**, o sistema operacional assegura a impossibilidade de outra instrução ser executada em meio ao tratamento de **open()**. Confira a seção 2.9.7 da especificação de "System Interfaces" do POSIX

4 7.4

Não é possível defender contra **buffer overflow** meramente desabilitando temporariamente os privilégios (ou seja, desabilitando de maneira que o privilégio seja recuperável pelo programa). Isso porque, quando um **buffer overflow** é bem sucedido e insere código arbitrário para ser executado, uma dessas instruções do atacante pode ser muito bem recuperar os privilégios !

5 7.5

Reproduzimos abaixo o código da questão (note que está reformatado !) com numeração de linhas para facilitar referências:

```
1  #include <stdio.h>
2  #include <sys/stat.h>
3  #include <sys/types.h>
4  #include <unistd.h>
5
6  int main() {
7    struct stat statbuf;
8    uid_t real_uid;
9    FILE* fp;
10   fp = fopen("/tmp/XYZ", "a+");
11   stat("/tmp/XYZ", &statbuf);
12   printf("The file owners user ID: %d\n", statbuf.st_uid);
13   printf("The processs real user ID: %d\n", getuid());
14   // Check whether the file belongs to the user
15   if (statbuf.st_uid == getuid()) {
16     printf("IDs match, continue to write to the file.\n");
17     // write to the file ...
18     if (fp) fclose(fp);
19   } else {
20     printf("IDs do not match, exit.\n");
```

```
21      if (fp) fclose(fp);
22      return -1;
23    }
24    return 0;
25  }
```

Existe situação de corrida, provocada pelas linhas 10 e 11 do código. Imagine a situação em que, antes da execução da linha 10, **/tmp/XYZ** é um symlink apontando para **/etc/passwd**, o que dá êxito à invocação de **fopen()** - já que EUID é **root**, o qual tem acesso a **/etc/passwd**.

Em seguida, depois da execução da linha 10, mas antes da execução da linha 11, o atacante redefine o symlink para que ele aponte para **/home/seed/somefile**. Supondo que o usuário **seed** seja o dono desse arquivo, a invocação de **stat()** vai preencher a variável **statbuf** com o UID do usuário **seed**, dando êxito ao teste executado na linha 15. Está feito o ataque.

6 7.6

Reproduzimos abaixo o código da questão (note que está reformatado !) com numeração de linhas para facilitar referências:

```
1  #include <stdio.h>
2  #include <sys/stat.h>
3  #include <sys/types.h>
4  #include <unistd.h>
5
6  int main() {
7    struct stat statbuf;
8    uid_t real_uid;
9    FILE* fp;
10   fp = fopen("/tmp/XYZ", "a+");
11   fstat(fileno(fp), &statbuf);
12   printf("The file owners user ID: %d\n", statbuf.st_uid);
13   printf("The processs real user ID: %d\n", getuid());
14   // Check whether the file belongs to the user
15   if (statbuf.st_uid == getuid()) {
16     printf("IDs match, continue to write to the file.\n");
17     // write to the file ...
18     if (fp) fclose(fp);
19   } else {
20     printf("IDs do not match, exit.\n");
21     if (fp) fclose(fp);
22     return -1;
23   }
24   return 0;
25 }
```

A princípio, parece que as linhas 10 e 11 introduzem condição de corrida, mas isso não é verdade.

Na execução de **fopen()**, mesmo que **/tmp/XYZ** seja um symlink, ele será seguido e o arquivo para o qual ele aponta será aberto, retornando um **FILE***. Esse **FILE*** é uma abstração por cima dos **file descriptors** do Linux (que são simplesmente do tipo **int**).

A função **fileno()** só faz aceitar um **FILE*** e retornar seu **file descriptor** original (de certa forma, essa função "desfaz" a abstração para revelar o **file descriptor** "cru" armazenado no **FILE***).

De qualquer modo, observe que, uma vez aberto, o **FILE*** (e, por extensão, o **file descriptor**) identificam unicamente o arquivo aberto, e nenhuma informação sobre **/tmp/XYZ** (o symlink do início) permanece. Em outras palavras, neste ponto não importa mais o fato de que o arquivo foi alcançado a partir de um symlink.

Assim, quando **fstat()** é invocada usando o **file descriptor** como argumento, ela preencherá a variável **statbuf** com as informações do arquivo aberto. Dessa forma, poderemos checar se o RUID do processo corresponde ao proprietário do arquivo aberto, não importando o que um atacante faça com o symlink **/tmp/XYZ** (o qual, nota-se, só é acessado uma única vez).