Esercitazione su pipe

Vediamo un esercizio sulle pipe (è una delle verifiche degli scorsi anni).

Crackme

Il programma crackme chiede un Personal Identification Number (PIN) di 5 cifre e ne verifica la correttezza. Si limita a stampare un messaggio ma si può pensare che solo nel caso il PIN sia corretto il programma dia accesso a risorse protette.

Una volta lanciato, crackme utilizza due pipe per interagire con altri processi. Su una pipe viene letto il PIN e sull'altra viene inviato il risultato della verifica in forma di stringa.

Il sorgente del programma (senza il PIN segreto) è il seguente:

```c
#include <fcntl.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>

#define PNAME1 "tmpPipeInput"
#define PNAME2 "tmpPipeOutput"
#define SUCCESSO "PIN corretto. Sei autenticato\n"
#define FALLIMENTO "PIN errato\n"
// il PIN è stato oscurato!

void chiuditutto() {
    unlink(PNAME1);        // rimuove la pipe
    unlink(PNAME2);        // rimuove la pipe
    exit(1);
}

// stampa l'errore e termina
void die(char *s) {
    perror(s);
    exit(EXIT_FAILURE);
}

main() {
    int fdI,fdO;
    int leggi;

    signal(SIGINT,chiuditutto);

    mkfifo(PNAME1,0666);    // crea la pipe, se esiste gia' non fa nulla
    mkfifo(PNAME2,0666);    // crea la pipe, se esiste gia' non fa nulla

    if ( (fdI = open (PNAME1,O_RDWR)) < 0 ) // apre la pipe per la lettura
        die("errore apertura pipe\n");

    if ( (fdO = open (PNAME2,O_RDWR)) < 0 ) // apre la pipe per la scrittura
        die("errore apertura pipe\n");

     while ( read(fdI, &leggi, sizeof(int) ) ) {
         // Controlla la correttezza del PIN
         if (leggi == PINsegreto)
             write(fdO, SUCCESSO, strlen(SUCCESSO)+1);
             // qui si ha accesso alle risorse ...
         else
             write(fdO, FALLIMENTO, strlen(FALLIMENTO)+
     }

}
```

Scaricare crackme qui.

Scrivere un programma crack.c che scopra il PIN segreto e lo stampi a video. Il programma deve interagire con crackme solo utilizzando le pipe (scoprire il PIN tramite debugging, anche se utile e divertente, non è considerata una soluzione, visto che l'esercizio è sull'uso delle pipe).

Notare che, per semplicità, gli interi vengono ricevuti direttamente nella loro rappresentazione binaria utilizzando la seguente istruzione:

```
read(fdI, &leggi, sizeof(int) )
```

Nel caso di interi a 32 bit, ad esempio, verranno letti direttamente i 4 byte che rappresentano il numero intero.

NOTA: È possibile interagire con le pipe di crackme da terminale ma si deve tener presente che gli interi sono rappresentati in little-endian, con il byte meno significativo al primo posto. Ad esempio il PIN 21664 che in esadecimale è `0x54a0` quando è rappresentato in 4 byte little-endian diventa `0xa0 0x54 0x00 0x00`. Per mandarlo sulla pipe si può usare il comando `echo` con le opzioni `-ne` che tolgono l'a-capo finale (-n) e interpretano le sequenze `\xa0` come byte (-a).

```
$ echo -ne "\xa0\x54\x00\x00" > tmpPipeInput
$ cat tmpPipeOutput
PIN errato
```

Possiamo anche inviare due PIN consecutivi

```
$ echo -ne "\xa0\x54\x00\x00\xa1\x54\x00\x00" > tmpPipeInput
$ cat tmpPipeOutput
PIN errato
PIN errato
```

Come si può notare crackme si aspetta i PIN codificati in 4 byte consecutivi senza separatore. Per inviarli da C **non serve fare alcuna conversione**: se il PIN è in una variabile intera sarà già rappresentato in little-endian.

Crackme con stringhe

La soluzione di inviare direttamente un intero nella sua rappresentazione interna può creare problemi di portabilità. Di solito si preferisce usare una rappresentazione che non dipenda dall'architettura o dal linguaggio utilizzato.

Questa variante di crackme legge i numeri interi come stringhe terminate da 0×00 e li converte, successivamente, in numeri interi per il confronto con il PIN.

Ecco il sorgente:

```
1  #include <fcntl.h>
2  #include <stdlib.h>
3  #include <signal.h>
4  #include <string.h>
5  #include <stdio.h>
6
7  #define PNAME1 "tmpPipeInputChars"
8  #define PNAME2 "tmpPipeOutputChars"
9  #define SUCCESSO "PIN corretto. Sei autenticato\n"
10 #define FALLIMENTO "PIN errato\n"
11 #define PINsegreto 13495
12
13 void chiuditutto() {
14     unlink(PNAME1);        // rimuove la pipe
15     unlink(PNAME2);        // rimuove la pipe
16     exit(1);
17 }
18
19 // stampa l'errore e termina
20 void die(char *s) {
21     perror(s);
22     exit(EXIT_FAILURE);
23 }
24
25 main() {
26     int fdI,fdO,r,pin;
27     char leggi[6];
28
29     signal(SIGINT,chiuditutto);
```

```
29      signal(SIGINT,chiuditutto);
30
31      mkfifo(PNAME1,0666);    // crea la pipe, se esiste gia' non fa nulla
32      mkfifo(PNAME2,0666);    // crea la pipe, se esiste gia' non fa nulla
33
34      if ( (fdI = open (PNAME1,O_RDWR)) < 0 ) // apre la pipe per la lettura
35          die("errore apertura pipe\n");
36
37      if ( (fdO = open (PNAME2,O_RDWR)) < 0 ) // apre la pipe per la scrittura
38          die("errore apertura pipe\n");
39
40      while (1) {
41          r=0;
42
43          // legge il pin un carattere alla volta
44          while ( r<6 && read(fdI, &leggi[r], 1 ) && leggi[r] != 0 )
45              r++;
46
47          pin = atoi(leggi); // converte la stringa in intero
48
49          // Controlla la correttezza del PIN
50          if (pin == PINsegreto)
51              write(fdO, SUCCESSO, strlen(SUCCESSO)+1);
52              // qui si ha accesso alle risorse ...
53          else
54              write(fdO, FALLIMENTO, strlen(FALLIMENTO)+1);
55
56      }
57  }
```

Provare a scrivere il un programma crack-chars.c che interagisca con crackme-chars e scopra il PIN segreto.

NOTA: anche in questo caso si può interagire con il programma da linea di comando ma i PIN saranno stringhe terminate dal byte `0x00`:

```
$ echo -ne "21664\x00" > tmpPipeInputChars
$ cat tmpPipeOutputChars
PIN errato
^C
$ echo -ne "21664\x0021666\x00" > tmpPipeInputChars
$ cat tmpPipeOutputChars
PIN errato
PIN errato
```

In questo caso da C è necessario effettuare una conversione da interi a stringhe utilizzando, ad esempio, `sprintf` (vedere il manuale).

SOLUZIONE GAIA :

```c
#include <fcntl.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>

#define POUT "tmpPipeInput"
#define PIN "tmpPipeOutput"

void die(char *s) {
    perror(s);
    exit(EXIT_FAILURE);
}

int main() {
    int out;
    int in;
    int codice = 0;
    char leggi[100];

    mkfifo(POUT,0666);    // crea la pipe, se esiste gia' non fa nulla
    mkfifo(PIN,0666);    // crea la pipe, se esiste gia' non fa nulla

    if ( (out = open(POUT,O_RDWR)) < 0 )
        die("errore apertura pipe\n");

    if ( (in = open(PIN,O_RDWR)) < 0 )
        die("errore apertura pipe\n");

    while(codice < 99999){
        write(out, &codice, sizeof(int));

        int r=0;
        while ( r<100 && read(in, &leggi[r], 1 ) && leggi[r] != 0 )
            r++;

        if (leggi[4] == 'c'){
            printf("il pin corretto è:%d\n", codice);
            return 0;
         }
         codice++;
     }

     return 0;
 }
```