[pipe] Crackme

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:

```
 1  #include <fcntl.h>
 2  #include <stdlib.h>
 3  #include <signal.h>
 4  #include <string.h>
 5
 6  #define PNAME1 "tmpPipeInput"
 7  #define PNAME2 "tmpPipeOutput"
 8  #define SUCCESSO "PIN corretto. Sei autenticato\n"
 9  #define FALLIMENTO "PIN errato\n"
10  // il PIN è stato oscurato!
11
12  void chiuditutto() {
13      unlink(PNAME1);         // rimuove la pipe
14      unlink(PNAME2);         // rimuove la pipe
15      exit(1);
16  }
17
18  // stampa l'errore e termina
19  void die(char *s) {
20      perror(s);
21      exit(EXIT_FAILURE);
22  }
23
24  main() {
25      int fdI,fdO;
26      int leggi;
27
28      signal(SIGINT,chiuditutto);
29
30      mkfifo(PNAME1,0666);    // crea la pipe, se esiste gia'
31      mkfifo(PNAME2,0666);    // crea la pipe, se esiste gia'
32
33      if ( (fdI = open (PNAME1,O_RDWR)) < 0 ) // apre la pipe
34          die("errore apertura pipe\n");
35
36      if ( (fdO = open (PNAME2,O_RDWR)) < 0 ) // apre la pipe
37          die("errore apertura pipe\n");
38
39      while ( read(fdI, &leggi, sizeof(int) ) ) {
40          // Controlla la correttezza del PIN
41          if (leggi == PINsegreto)
42              write(fdO, SUCCESSO, strlen(SUCCESSO)+1);
```

```
43                  // qui si ha accesso alle risorse ...
44              else
45                  write(fdO, FALLIMENTO, strlen(FALLIMENTO)+1);
46          }
47
48      }
```

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:

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

#define PNAME1 "tmpPipeInputChars"
#define PNAME2 "tmpPipeOutputChars"
#define SUCCESSO "PIN corretto. Sei autenticato\n"
#define FALLIMENTO "PIN errato\n"
#define PINsegreto 13495

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,r,pin;
    char leggi[6];

    signal(SIGINT,chiuditutto);

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

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

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

    while (1) {
        r=0;

        // legge il pin un carattere alla volta
        while ( r<6 && read(fdI, &leggi[r], 1 ) && leggi[r]
            r++;

        pin = atoi(leggi); // converte la stringa in intero

        // Controlla la correttezza del PIN
        if (pin == PINsegreto)
            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      }
```

Scaricare la variante di crackme.

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).

Comments: 8

Leave a reply »

Gaia
November 20th, 2012 at 18:34

noi abbiamo provato a fare così: va bene?

```
1   #include <fcntl.h>
2   #include <stdlib.h>
3   #include <signal.h>
4   #include <string.h>
5
6   #define POUT "tmpPipeInput"
7   #define PIN "tmpPipeOutput"
8
9   void die(char *s) {
10      perror(s);
11      exit(EXIT_FAILURE);
12  }
13
14  int main() {
```

```
15      int out;
16      int in;
17      int codice = 0;
18      char leggi[100];
19
20      mkfifo(POUT,0666);     // crea la pipe, se esiste gia' nc
21      mkfifo(PIN,0666);     // crea la pipe, se esiste gia' nor
22
23      if ( (out = open(POUT,O_RDWR)) < 0 )
24          die("errore apertura pipe\n");
25
26      if ( (in = open(PIN,O_RDWR)) < 0 )
27          die("errore apertura pipe\n");
28
29      while(codice < 99999){
30          write(out, &codice, sizeof(int));
31
32          int r=0;
33          while ( r<100 && read(in, &leggi[r], 1 ) && leggi[r]
34              r++;
35
36          if (leggi[4] == 'c'){
37              printf("il pin corretto è:%d\n", codice);
38              return 0;
39          }
40          codice++;
41      }
42
43      return 0;
44  }
```

Andrea Baesso
November 21st, 2012 at 01:43

```
1   #include <stdio.h>
2   #include <sys/types.h>
3   #include <fcntl.h>
4   #include <unistd.h>
5   #include <sys/stat.h>
6   #include <string.h>
7   #include <stdlib.h>
8
9   #define PNAME1 "tmpPipeInput"
10  #define PNAME2 "tmpPipeOutput"
11  #define SUCCESSO "PIN corretto. Sei autenticato\n"
12  #define FALLIMENTO "PIN errato\n"
13  int main(){
14
15      int fd1,fd2,i,dim,dimfal;
16      char *message;
17
18      message = (char *)malloc(sizeof(char) *(strlen(FALLIMENT
```

```
19
20      mkfifo(PNAME1,0666);// crea la pipe, se esiste gia' non
21      mkfifo(PNAME2,0666);// crea la pipe, se esiste gia' non
22
23      if((fd1=open(PNAME1,O_RDWR))<0){//apro la pipe
24          perror("Non sono riuscito ad aprire la pipe1");
25          exit(1);
26      }
27      if((fd2=open(PNAME2,O_RDWR))<0){//apro la pipe
28          perror("Non sono riuscito ad aprire la pipe2");
29          exit(1);
30      }
31
32      for(i=0;i<100000;i++){
33        write(fd1,&i,sizeof(int));//scrivo nella pipe1 il valo
34          dim=read(fd2,message,strlen(FALLIMENTO)+1);//leggo l
35        if(strcmp(message,FALLIMENTO)) {//controllo la rispost
36          printf("Il pin è %d\n",i);
37              break;}
38      }
39
40      close(fd1);//chiudo la pipe
41      close(fd2);//chiudo la pipe
42      return 0;
43  }
```

Andrea Baesso
November 21st, 2012 at 01:44

Scusate, il codice è relativo alla soluzione per il file crackme.c

riccardo
November 21st, 2012 at 02:17

@Gaia: ottimo codice conciso. Vedo che avete utilizzato la lettura a singolo byte. Ricordatevi però di mettere commenti in modo da rendere la soluzione leggibile da tutti. In particolare invito tutti a vedere bene il while che fa la read perché non è immediato. Notate anche l'uso di && alla 'C': se una condizione booleana è falsa quelle successive non vengono eseguite. Ad esempio se la read ritorna 0 (EOF) non faccio il check sul valore di leggi[r] (che conterrebbe sporco di memoria)

@Andrea: Il codice funziona ma ci sono un paio di osservazioni importanti da fare (1 e 3) e un errore (2):

1) leggi tutta la stringa fino alla lunghezza di FALLIMENTO+1. Questo funziona perché è sufficiente per svuotare la pipe. La scrittura è atomica e ogni lettura svuota tutta la pipe quindi non rischi di leggere messaggi successivi accodati. Fate attenzione perché in altre applicazioni può accadere di leggere con una read due write (c'era un esempio nella lezione sulle pipe);

2) quando arriva la stringa SUCCESSO ne leggi solo un pezzo ma questo ti basta per accorgertene

e uscire. La pipe però non è vuota! Infatti se esegui il programma due volte la seconda fallisce (perché ci sono dati rimasti sulla pipe). La soluzione corretta è di leggere sempre la dimensione della stringa più lunga (SUCCESSO+1 in questo caso). Ricordatevi che la read se ci sono meno byte di quelli specificati legge solo quelli presenti, non ci sono problemi;

3) strcmp ritorna 0 quando le stringhe sono uguali e -1 o 1 se una e' maggiore dell'altra. In "C" la if fallisce se il risultato è 0 quindi if(strcmp(…)) ha successo solo se le stringhe sono diverse.

Andrea Baesso
November 21st, 2012 at 18:38

2)Quindi mi conviene sempre leggere la stringa più lunga da pipe?

Posto una possibile soluzione del secondo esercizio, in realtà quì non utilizzo i caratteri 0×00 ma il pin lo trova. E' dovuto al fatto che le stringhe hanno il carattere di terminazione?

```
#include <fcntl.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <stdio.h>
#define PNAME1 "tmpPipeInputChars"
#define PNAME2 "tmpPipeOutputChars"
#define FALLIMENTO "PIN errato\n"
#define SUCCESSO "PIN corretto. Sei autenticato\n"
void die(char *s) {
    perror(s);
    exit(EXIT_FAILURE);
}
main() {
    int fd0,fd1;
    int i;
    char *leggi;
    char scrivi[10];
    //alloco la memoria per il lettore
    leggi = (char *)malloc(sizeof(char) * strlen(SUCCESSO));

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

    if ( (fd0 = open(PNAME1,O_RDWR)) < 0 )//apro la pipe fd0
        die("errore apertura pipe\n");

    if ( (fd1 = open(PNAME2,O_RDWR)) < 0 )//apro la pipe fd1
        die("errore apertura pipe\n");

     for(i=0;i<100000;i++){
        sprintf(scrivi,"%d",i);//converto il valore di i nel
        write(fd0,scrivi,strlen(scrivi)+1);//scrivo 'scrivi'
        if(read(fd1,leggi,strlen(SUCCESSO)+1)>0){//leggo da
            if(strcmp(SUCCESSO,leggi)==0){//se leggi è ugual
```

```
36                      printf("OK! PIN : %d\n",i);
37                      close(fd0);
38                      close(fd1);
39                      return 0;
40                  }
41              }
42              else{
43                  perror("errore in lettura da pipe\n");
44                  exit(1);
45              }
46          }
47          close(fd0);
48          close(fd1);
49  }
```

riccardo
November 23rd, 2012 at 01:30

Sì come dicevo sopra, il numero di byte che metti nella read è un limite superiore. Se ci sono meno byte la read li legge e ritorna il numero di byte letti. Quindi se, come in questo caso, si vuole sempre svuotare la pipe va bene leggere usando la dimensione della stringa più lunga.

La soluzione al secondo esercizio va bene! Quando scrivi con strlen(scrivi)+1 in effetti mandi il byte di terminazione di stringa che è proprio 0×00.

riccardo
December 4th, 2012 at 15:29

Per la verifica della settimana prossima, vi consiglio di provare a risolvere anche la verifica dello scorso anno. Postate le vostre soluzioni!

Francesco
December 10th, 2012 at 20:22

Se qualcuno magari con ubuntu a 64 bit non riesce ad eseguire crackme e gli da file non esistente la soluzione è scrivere da terminale questo:

sudo apt-get install ia32-libs

Leave a Reply

Name *

Mail * (will not be published)

Website

Comment

Submit Comment