CISC870 Assignment

September 28, 2022

CISC870 Assignment1

Name: Donghao Wang
Student Number: 20119632

Partner:
Student Number:

2.1

With the srand() fucntion, the program outputs two lines of numbers. Every time the program runs, the first number above increases for a few and the other number on the bottom changes. When commenting out the line with srand() function, the program still output two lines of numbers, but no matter how the first number changes through out time, the latter number doesn't change.
Since the first number is a count of time and the second number is a generated random number, we can deduce that, in this program, the time() function is for producing a changing integer/long value as a seed for generating random number, and the srand() function should be the function takes an variable from time() and generate the pseudo random number using the variable as a key.

2.2

For starters, bob should loop through every possible key in each generation. So he just need to modify the original program provided in task1 and make it loop from time "2018-04-17 21:08:49" to "2018-04-17 23:08:49", which in seconds is 1524013729 to 1524020929. These values are acquired through the date -d command.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define KEYSIZE 16
void main()
{
    long int starttime;
    int i;
    for(starttime=1524013729;starttime<=1524020929;starttime++){
        char key[KEYSIZE];
        srand(starttime);
        for (i = 0; i< KEYSIZE; i++){
            key[i] = rand()%256;
            /*printf("%.2x", (unsigned char)key[i]);*/
        }
        printf("\n");
    }


}
```

The keys are stored in a file result1.txt by the command at bash ./task2 ¿ result1.txt After that a python program decrypt.py is used to find out the decryption key by trying the keys one by one.(this is from google colab)

```
import binascii
from Crypto.Cipher import AES

keyfile = open('result1.txt','r')
try_outs= keyfile.readlines()
keyfile.close()
for random_key in try_outs:
    random_key=random_key.rstrip()

    magiccrypt=AES.new(binascii.unhexlify(random_key.lower()),
                       AES.MODE_CBC, binascii.unhexlify('09080706050403020100A2B2C2D2E2F2'.lower()))
    cipher=magiccrypt.encrypt(binascii.unhexlify('255044462d312e350a25d0d4c5d80a34'.lower()))

    if(cipher==binascii.unhexlify('d06bf9d0dab8e8ef880660d2af65aa82'.lower())):
        print('The correct key is '+ random_key)
        break
```

2.3

```
$ watch -n .1 cat /proc/sys/kernel/random/entropy_avail
```

Among all the 5 activities, moving the mouse and typing increases the entropy for a few as the movement started and stop increasing entropy when stopped. Before the experiment, I expect the effect to go like this from small to big: clicking mouse, moving mouse, typing, reading large file, visiting website. The result would match most of the expectation I had. In my practice, clicking the mouse has the least effect on entropy. Moving mouse and typing has moderate effects. Reading large file or visiting a website would increase the entropy largely since it may involve a lot of mouse moving and clicking.

2.4

```
$ cat /dev/random | hexdump
```

If there is no movement on the device like moving the mouse, the command won't print anything. When there is random mouse movement, the command prints out random numbers start with a sorted 8-bit hexadecimal heading like 00000a0.
It would prove that the random number generation is related to the system entropy. To do DDOS attack, the attacker first must figure out how to exhaust system entropy. Requesting large amount of session id in a small period would achieve this, making the server unable to generate random number by /dev/random.

2.5

2.5.1

```
$ cat /dev/urandom | hexdump
```

It quickly generates random numbers and moving mouse had no affect on the speed.

2.5.2

```
$ head -c 1M /dev/urandom > output.bin
$ ent output.bin
```

The outcome produces a very low correlation coefficient, Arithmetic mean value=127.5919 very close to 127.5, monte carlo value for pi is 3.141621176(indicates zero error), Chi-square distribution of 215.06 and randomly would exceed this value 96.72 per cent of times. These values indicate the quality of the random number is good. There is nearly no correlation between the numbers and the randomness is ensured.

2.5.3

```c
#include <stdio.h>
#include <stdlib.h>
#define LEN 32 // 256 bits
void main()
{
    unsigned char *key = (unsigned char *) malloc(sizeof(unsigned char)*LEN);

    FILE* random = fopen("/dev/urandom", "r");

    fread(key, sizeof(unsigned char)*LEN, 1, random);

    int i;

    for (i = 0; i< LEN; i++){
            printf("%.2x", (unsigned char)key[i]);
        }
    printf("\n");
    fclose(random);


}
```

The change being made is to double the value of global variable Len.Here is the output of the program.

```
james@LAPTOP-7AJ4SITH:/mnt/c/james/870/A1$ gcc -o task5 task5.c
james@LAPTOP-7AJ4SITH:/mnt/c/james/870/A1$ ./task5
27b87a6276df7617cdf6c6f1b461d19318fab677e11d1b7ca9e6292a47cbcd0e
james@LAPTOP-7AJ4SITH:/mnt/c/james/870/A1$ ./task5
e88aa462a6e0d43beaa3e89601a4f71e768a66b7d8f13a7c58c829f60db6dc00
james@LAPTOP-7AJ4SITH:/mnt/c/james/870/A1$ ./task5
e1f33663496bbdb90ad6230437178131c865280bacc5f3dd7dff8d4ec2fb64cc
james@LAPTOP-7AJ4SITH:/mnt/c/james/870/A1$ ./task5
96895b9bcb6f055d66a990fc2b8c1d1b1641ddcc866c276cab62f7af81849183
```