

Applied Cryptography Lab-03

Manual

19 September 2022 22:09

Task 1: Generate Encryption Key in a Wrong Way

In this task we run the code provided in the lab description which generates 128 bit encryption key based on the random value (which is time in this case). Below is the code provided in the lab description

Step 1

Compile and run the following program and describe your observation

```
/* task1.c */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define KEYSIZE 16
void main() {
    int i;
    char key[KEYSIZE];
    printf("%lld\n", (long long) time(NULL));
    srand (time(NULL));
    for (i = 0; i < KEYSIZE; i++) {
        key[i] = rand()%256;
        printf("%.2x", (unsigned char)key[i]);
    }
    printf("\n");
}
```

Commands

```
$ gcc task1.c -o task1
$ ./task1
$ ./task1
$ ./task1
```

Step 2

Now comment out the statement `srand (time(NULL));` and then compile and run the program 2-3 times.

Give your observation with screenshot

Task2: Guessing the Key

In this task we as Bob need to break the encryption done by Alice by guessing the key she

used for encryption. Since Alice did not use safe and efficient way to generate the key we assume that Bob can guess the key. We do require some reconnaissance before we can generate the key, results of which are already provided in lab description.

Plaintext: 255044462d312e350a25d0d4c5d80a34
Ciphertext: d06bf9d0dab8e8ef880660d2af65aa82
IV: 09080706050403020100A2B2C2D2E2F2

Timestamp: 2018-04-17 23:08:49
Time range: 2018-04-17 21:08:49 - 2018-04-17 23:08:49

After looking at the source code of the key generation program used by Alice we can see that the `time()` function is used as seed value given to the `srand()` function. We also know that the `time()` returns the time elapsed in secs. Therefore for the malicious key generation program we need to give the same seed value as Alice gave while generating the key. From our reconnaissance we know the time range in which Alice generated the key, so we can brute force all the keys possible during the time range.

Step 1

We can use the date command as mentioned in the lab description to get the time elapsed in seconds.

Commands

```
$ date -d "2018-04-17 21:08:49" +%s  
<value1>  
$ date -d "2018-04-17 23:08:49" +%s  
<value2>
```

Give your observation with a screenshot. Make a note of value1 and value2

Step 2

Modify the program used by Alice to generate all the keys in this time range (value1 – value2). Write all the keys to the file called keys.txt

```
/* task2.c */  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#define KEYSIZE 16  
void main() {
```

```

int i, j;
FILE *f;
char key[KEYSIZE];
int value1, value2;
/* use the output of the previous step as value1 and value2 respectively*/
// value1 = output of date -d "2018-04-17 21:08:49" +%s ;
// value2 = output of date -d "2018-04-17 23:08:49" +%s ;
f = fopen("keys.txt", "w");
for (j = value1; j <= value2; j++) {
    srand (j);
    for (i = 0; i < KEYSIZE; i++) {
        key[i] = rand()%256;
        fprintf(f, "%.2x", (unsigned char)key[i]);
    }
    fprintf(f, "\n");
}
}

```

Compile and run the above program

Commands

```

$ gcc task2.c -o task2
$ ./task2

```

Give your observation with a screenshot

Step 3

All the possible keys generated are written to file "keys.txt". Compile and run the following python script that will encrypt the plaintext with keys in "keys.txt" and compares it with cipher text.

The following code compares each key to find the one used by Alice. However, it requires the cryptography module. Install it using the following steps

```

$ sudo apt-get update
$ sudo apt install python3-pip
$ python3 -m pip install pycryptodome

```

```

# decrypt.py
from Crypto import Random
from Crypto.Cipher import AES
file = open("keys.txt", "r")
ciphertext = "d06bf9d0dab8e8ef880660d2af65aa82"
for i in range(0,7200):
    str = file.readline()

    key = bytes.fromhex(str[:-1]).decode("hex")
    IV = bytes.fromhex("09080706050403020100A2B2C2D2E2F2".lower())
    plaintext1 = bytes.fromhex("255044462d312e350a25d0d4c5d80a34")
    cipher = AES.new(key, AES.MODE_CBC, IV)
    encrypted = cipher.encrypt(plaintext1)
    #print("Encrypted: ",encrypted.hex())
    if ciphertext == encrypted.hex():#.encode("hex")[0:32]:
        print("")

```

```
print("Match found")
print("key: "+str[:-1])
print("Ciphertext: " + ciphertext)
print("Encrypted: " + (encrypted).hex())
print("")
```

Commands

```
$ python3 decrypt.py
```

Task 3: Measure the Entropy of Kernel

The randomness is measured using entropy, which is different from the meaning of entropy in information theory. Here, it simply means how many bits of random numbers the system currently has. Find out how much entropy the kernel has at the current moment.

Commands

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

Observe the output on the screen by moving the mouse and pressing keys

Give your observation with screen shot.

Task 4: Get Pseudo Random Numbers from /dev/random

Linux stores the random data collected from the physical resources into a random pool, and then uses two devices to turn the randomness into pseudo random numbers.

These two devices are /dev/random and /dev/urandom. They have different behaviors.

The /dev/random device is a blocking device. Namely, every time a random number is given out by this device, the entropy of the randomness pool will be decreased. When the entropy reaches zero, /dev/random will block, until it gains enough randomness.

Step 1

Execute both the commands simultaneously

Commands

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

What happens if you do not move your mouse or type anything. Then, randomly move your mouse and see whether you can observe any difference.

Give your observation with a screenshot

Task 5: Get Pseudo Random Numbers from /dev/urandom

Linux provides another way to access the random pool via the /dev/urandom device, except that this device will not block. Both /dev/random and /dev/urandom use the random data from the pool to generate pseudo random numbers. When the entropy is not sufficient, /dev/random will pause, while /dev/urandom will keep generating new numbers.

Step 1

Execute both the commands simultaneously

Commands

```
$ watch -n .1 cat /proc/sys/kernel/random/entropy_avail  
$ cat /dev/urandom | hexdump
```

Give your observation with screenshot

Step 2

Measure the quality of the random number using a tool called ent.

Installing the package: `sudo apt install ent`

Please note that when we run /dev/urandom, a lot of random numbers are generated and since it is a non-blocking process the numbers will keep generating and may not be able to observe the effect of cursor movement immediately. For that we can write random numbers to file and check its quality using ent command.

Commands

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

Step 3

The program from Task 1 can be modified to write a new 128 bit

key using /dev/urandom

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define KEYSIZE 16
void main() {
    int i;
    FILE *random;
    unsigned char *key = (unsigned char *) malloc (sizeof (unsigned char) * KEYSIZE);

    random = fopen("/dev/urandom", "r");
    for (i = 0; i < KEYSIZE; i++) {
        fread(key, sizeof(unsigned char) * KEYSIZE, 1, random);
        printf("%.2x", *key);
    }
    printf("\n");
    fclose(random);
}
```

Commands

```
$ gcc task5.c -o task5
$ ./task5
$ ./task5
```

Give your observation with screenshot

Submission

You need to submit a detailed lab report to describe what you have done and what you have observed, including screenshots and code snippets. You also need to provide explanation to the observations that are interesting or surprising. You are encouraged to pursue further investigation, beyond what is required by the lab description. Please submit in word or PDF format only.