

软件安全实验报告

2022 年 12 月 14 日

课程名称:	软件安全
成绩评定:	
实验名称:	伪随机数生成实验
实验编号:	实验 6
指导教师:	刁文瑞
姓 名:	李晨漪
学 号:	202000210103
学 院:	山东大学网络空间安全学院
专 业:	网络空间安全

目录

1	实验目的	3
2	实验步骤与结果	3
2.1	Task 1: 用错误的方式生成加密密钥	3
2.2	Task 2: 猜测密钥	4
2.3	Task 3: 测量内核的熵	5
2.4	任务 4: 从 /dev/random 中获取伪随机数	6
2.5	任务 5: 从 /dev/urandom 获取随机数	6

1 实验目的

生成随机数是安全软件中非常常见的任务。在许多情况下，加密密钥不是由用户提供的，而是在软件内部生成的。它们的随机性非常重要。否则，攻击者可以预测加密密钥，从而达到破坏加密目的。许多开发人员从其先前的经验中知道如何生成随机数（例如用于蒙特卡洛模拟），因此他们使用类似的方法生成用于安全目的的随机数。不幸的是，随机数序列对于蒙特卡洛模拟可能是好的，但对于加密密钥则可能是不好的。开发人员需要知道如何生成安全的随机数，否则就会犯错。在一些著名的产品（包括 Netscape 和 Kerberos）中也犯过类似的错误。在本实验中，我们将学习为什么典型的随机数生成方法不适用于生成秘密（例如加密密钥）。进一步学习生成用于安全目的的伪随机数的标准方法。本实验涵盖以下主题：

1. 伪随机数生成
2. 随机数生成中的错误
3. 加密密钥生成
4. 设备文件 `/dev/random` 和 `/dev/urandom`

2 实验步骤与结果

2.1 Task 1: 用错误的方式生成加密密钥

case 1: 运行程序，不注释 `srand`:

```
[12/09/22] seed@VM:~/.../Lab$ ./task1
1670574385
19139c5cbeed3c7a12ab5cf91349095b
[12/09/22] seed@VM:~/.../Lab$ ./task1
1670574405
13fb28cab2416bc49dc827cd61820c35
[12/09/22] seed@VM:~/.../Lab$ ./task1
1670574410
ceffa74c563bd124ae6a7b9cb2f5564b
```

图 1:

case 2: 运行程序，注释 `srand`:

```
[12/09/22] seed@VM:~/.../Lab$ ./task1
1670576324
67c6697351ff4aec29cdbaabf2fbe346
[12/09/22] seed@VM:~/.../Lab$ ./task1
1670576330
67c6697351ff4aec29cdbaabf2fbe346
[12/09/22] seed@VM:~/.../Lab$ ./task1
1670576341
67c6697351ff4aec29cdbaabf2fbe346
```

图 2:

case 1: 所产生的密钥（随机值）随当前时间的变化而变化。

case 2: 所产生的随机值不变化，始终为默认产生的随机值。

`srand()`: `srand()` 函数为 `rand()` 函数提供随机数种子，`rand` 函数产生伪随机数，该方法有最大周期 M ，即该方法在一定的范围内生成一串数字，不过这个 M 通常比较大，所以正常使用时一般

看起来就是产生了随机的数，不过这个函数由于默认种子是 1，所以每次产生的随机数都是相同的，也就是说每次运行程序，产生的随机数都是一样的，失去了部分随机的意义，所以一般会配合 `srand` 函数使用。

`time()`：库函数 `time()` 以从纪元 1970-01-01 00:00:00 +0000 (UTC) 起的秒数的形式返回当前时间。

2.2 Task 2: 猜测密钥

Step1: 根据加密文件时间戳计算两小时内秒数范围。

```
[12/13/22]seed@VM:~$ date -d "2018-04-17 23:06:49" +%s
1524020809
[12/13/22]seed@VM:~$ date -d "2018-04-17 23:08:49" +%s
1524020929
[12/13/22]seed@VM:~$
```

图 3:

Step2: 编写代码获得所有可能的密钥。

```
1 //task2.c
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <time.h>
5 #define KEYSIZE 16
6
7 void main()
8 {
9     int i;
10    char key[KEYSIZE];
11
12    long long j;
13    for (j = 1524020929 - 60 * 60 * 2; j < 1524020929; j++){
14        // printf("%lld\n", j);
15        srand (j);
16
17        for (i = 0; i < KEYSIZE; i++){
18            key[i] = rand() % 256;
19            printf("%.2x", (unsigned char)key[i]);
20        }
21        printf("\n");
22    }
23    printf("\n");
24 }
```

Step3: 编译运行获得所有可能的密钥，并将结果输出到 `key.txt` 文件中。

```
[12/13/22] seed@VM:~/.../Lab$ gcc task2.c -o task2
[12/13/22] seed@VM:~/.../Lab$ ./task2 > key.txt
[12/13/22] seed@VM:~/.../Lab$
```

图 4:

Step4: 暴力枚举法利用 key.txt 中密钥对已知明文、IV 进行 AES-CBC 加密，与密文相比较，得到匹配成功的密钥。

```
1 from Crypto.Cipher import AES
2
3 data = bytearray.fromhex('255044462d312e350a25d0d4c5d80a34')
4 ciphertext = bytearray.fromhex('d06bf9d0dab8e8ef880660d2af65aa82')
5 iv = bytearray.fromhex('09080706050403020100A2B2C2D2E2F2')
6
7 with open('key.txt') as f:
8     keys = f.readlines()
9
10 for k in keys:
11     k = k.rstrip('\n')
12     key = bytearray.fromhex(k)
13     cipher = AES.new(key=key, mode=AES.MODE_CBC, iv=iv)
14     guess = cipher.encrypt(data)
15     if guess == ciphertext:
16         print("the key is: ", k)
17         break
```

密钥为: 95fa2030e73ed3f8da761bb4eb805dfd7。

2.3 Task 3: 测量内核的熵

查找在当前时刻内核的熵，watch 命令来监控熵的变化。

```
[12/09/22] seed@VM:~$ cat /proc/sys/kernel/random/entropy_avail
3171
[12/09/22] seed@VM:~$ watch -n .1 cat /proc/sys/kernel/random/entropy_avail
```

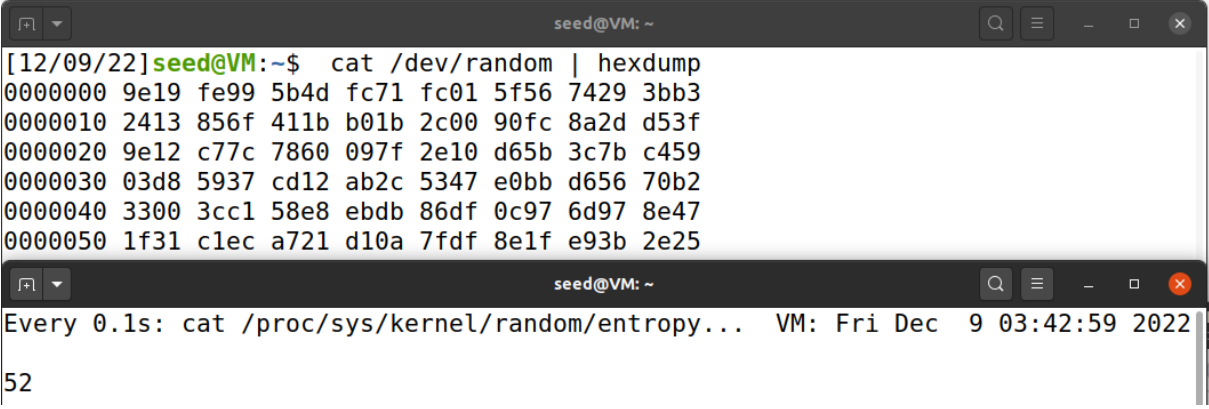
图 5:

• 现象

移动鼠标、点击鼠标、输入、读取一个大文件、访问一个网站都会使熵增大。

而输入、访问网站、读取文件显著地使熵变大, 可能因为这些活动操作更为复杂, 频率更高。移动鼠标、点击鼠标这样相对单一的操作使熵增大的速度稍慢一点。

2.4 任务 4: 从 /dev/random 中获取伪随机数



```
seed@VM: ~  
[12/09/22]seed@VM:~$ cat /dev/random | hexdump  
00000000 9e19 fe99 5b4d fc71 fc01 5f56 7429 3bb3  
00000100 2413 856f 411b b01b 2c00 90fc 8a2d d53f  
00000200 9e12 c77c 7860 097f 2e10 d65b 3c7b c459  
00000300 03d8 5937 cd12 ab2c 5347 e0bb d656 70b2  
00000400 3300 3cc1 58e8 ebdb 86df 0c97 6d97 8e47  
00000500 1f31 c1ec a721 d10a 7fdf 8e1f e93b 2e25  
  
Every 0.1s: cat /proc/sys/kernel/random/entropy... VM: Fri Dec 9 03:42:59 2022  
52
```

图 6:

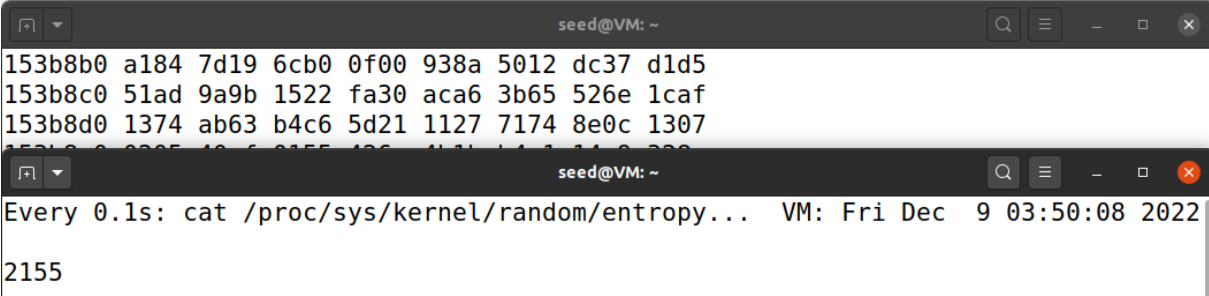
当鼠标不动时随机数产生速度缓慢的增加；当鼠标移动时，随机数产生速度会先增加（此时熵值大约在 100-200 之间）然后又马上减小。这是因为当该设备给出随机数时，随机池的熵将减小。当熵达到零时，/dev/random 将阻塞，直到获得足够的随机性为止。

- 问题

/dev/random 设备是阻塞设备。即，每当该设备给出随机数时，随机池的熵将减小。当熵达到零时，/dev/random 将阻塞，直到获得足够的随机性为止。

因此，如果攻击者不断请求连接建立，就会使得/dev/random 的可用熵被耗尽，从而导致随机数生成器不能正常使用。

2.5 任务 5: 从 /dev/urandom 获取随机数



```
seed@VM: ~  
153b8b0 a184 7d19 6cb0 0f00 938a 5012 dc37 d1d5  
153b8c0 51ad 9a9b 1522 fa30 aca6 3b65 526e 1caf  
153b8d0 1374 ab63 b4c6 5d21 1127 7174 8e0c 1307  
153b8e0 0305 1016 0155 126 1111 1111 1111 0305  
  
Every 0.1s: cat /proc/sys/kernel/random/entropy... VM: Fri Dec 9 03:50:08 2022  
2155
```

图 7:

移动鼠标不会影响结果。

```

[12/09/22]seed@VM:~$ head -c 1M /dev/urandom > output.bin
[12/09/22]seed@VM:~$ ent output.bin
Entropy = 7.999823 bits per byte.

Optimum compression would reduce the size
of this 1048576 byte file by 0 percent.

Chi square distribution for 1048576 samples is 257.23, and randomly
would exceed this value 44.91 percent of the times.

Arithmetic mean value of data bytes is 127.5221 (127.5 = random).
Monte Carlo value for Pi is 3.140385210 (error 0.04 percent).
Serial correlation coefficient is -0.001314 (totally uncorrelated = 0.0).

```

图 8:

熵的质量评价:

熵 = 7.999823 位/字节。

...

数据字节的算术平均值为 127.5221 (127.5 = 随机)。

Pi 的蒙特卡洛值为 3.140385210 (误差 0.04%)。

序列相关系数为 -0.001314 (完全不相关 = 0.00)。

可认为产生的熵质量很好。

生成一个 256 bit 的加密密钥:

```

1  \\task5.c
2  #include<stdio.h>
3  #include<stdlib.h>
4  #define LEN 32 // 256 bits
5  void main()
6  {
7  unsigned char *key = (unsigned char *) malloc(sizeof(unsigned char)*
      LEN);
8  FILE* random = fopen("/dev/urandom", "r");
9  fread(key, sizeof(unsigned char)*LEN, 1, random);
10 fclose(random);
11 int i;
12 for(i=0;i<32;i++)
13 {
14 printf("%.2x", (unsigned char)key[i]);
15 }
16 printf("\n");
17 }

```

```
[12/09/22] seed@VM:~/.../Lab$ cc task5.c -o task5
[12/09/22] seed@VM:~/.../Lab$ ./task5
901247bc9055641b1d042f6bb5625bc25ca67491e9cdd989befde7d330599cee
```

图 9:

参考文献

- [1] 杜文亮. 计算机安全导论：深度实践 [M]. 北京：高等教育出版社,2020.4.