伪随机数生成实验

1 概述

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，学生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）。他们将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：

- 伪随机数生成
- 随机数生成中的错误
- 加密密钥生成
- 设备文件 `/dev/random` 和 `/dev/urandom`

实验环境 本实验在我们预先构建好的 Ubuntu 20.04 VM（可以从我们的 SEED 网站当中下载）当中测试可行。

2 实验任务

2.1 任务 1: 用错误的方式生成加密密钥

要生成一个好的伪随机数，我们需要从一些随机的东西开始。否则，结果会是可预测的。下面的程序使用当前时间作为伪随机数生成器的种子。

Listing 1: ” 生成一个 128 bit 的加密密钥”

```
#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));       ①

  for (i = 0; i< KEYSIZE; i++){
     key[i] = rand()%256;
     printf("%.2x", (unsigned char)key[i]);
  }
  printf("\n");
}
```

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。运行上面的代码，并描述你的观察结果。然后，注释掉第 ① 行，再次运行该程序，并描述你观察到的结果。使用在两种情况下观察到的结果来解释代码中 `srand()` 和 `time()` 函数的用途。

2.2 任务 2: 猜测密钥

2018 年 4 月 17 日，Alice 完成了纳税申报表，并将报税表（PDF 文件）保存在磁盘上。为了保护文件，她使用任务 1 中描述的程序生成密钥对 PDF 文件进行了加密。她将密钥写在笔记本上，并安全地存储在保险箱中。几个月后，Bob 闯入她的计算机，并获得了加密的纳税申报表的副本。由于 Alice 是一家大公司的首席执行官，因此该文件非常有价值。

Bob 无法获得加密密钥，但是在 Alice 的计算机上，他看到了密钥生成程序，并怀疑 Alice 的加密密钥可能是由该程序生成的。他还注意到加密文件的时间戳是 `2018-04-17 23:08:49` 。他猜测密钥可能在创建文件之前的两个小时内生成。

由于该文件是 PDF 文件，因此有文件头。标头的开头部分始终是版本号。在创建文件期间，PDF-1.5 是最常见的版本，也就是说标头以 8 字节的 `%PDF-1.5` 开头。接下来的 8 个字节的数据也很容易预测。因此，鲍勃轻松获得了纯文本的前 16 个字节。根据加密文件的元数据，他知道使用 `aes-128-cbc` 对文件进行了加密。由于 AES 是 128 位密码，因此一个明文块由 16 字节的明文组成，即 Bob 知道一个明文块及其对应的密文。此外，鲍勃还从加密文件中知道初始向量（IV ）(IV 从未加密)。这是 Bob 知道的：

```
Plaintext:  255044462d312e350a25d0d4c5d80a34
Ciphertext: d06bf9d0dab8e8ef880660d2af65aa82
IV:         09080706050403020100A2B2C2D2E2F2
```

你的工作是帮助 Bob 找出 Alice 的加密密钥，以便解密整个文档。你应该编写一个程序来尝试所有可能的密钥。如果密钥是正确生成的，那么该任务是不可能完成的。但是，由于 Alice 使用 `time()` 作为她的随机数生成器的种子，因此你应该能够轻松地找到她的密钥。您可以使用 `date` 命令打印出指定时间和纪元之间的秒数，即 `1970-01-01 00:00:00 +0000 (UTC)`。请参见以下示例。

```
$ date -d "2018-04-15 15:00:00" +%s
1523818800
```

2.3 任务 3: 测量内核的熵

在虚拟世界中，创建随机性是困难的，只靠软件是很难生成随机数的。许多系统从物理世界中获得随机性。Linux 从下面的物理资源中获取随机性：

```
void add_keyboard_randomness(unsigned char scancode);
void add_mouse_randomness(__u32 mouse_data);
void add_interrupt_randomness(int irq);
void add_blkdev_randomness(int major);
```

前两个很容易理解：第一个使用按键之间的时间间隔；第二个使用鼠标移动和中断定时；第三个使用中断之间的时间间隔。当然，并非所有中断都是随机性的良好来源。例如，定时器中断不是一个很好的选择，因为它是可预测的。而磁盘中断是更好的措施。最后一个度量块设备请求的完成时间。

随机性使用*熵* 来度量，这与信息论中熵的含义不同。在这里，熵仅仅意味着系统当前有多少 bit 的随机数。你可以使用下面的命令找出在当前时刻内核的熵是多少。

```
$ cat /proc/sys/kernel/random/entropy_avail
```

watch 可以周期性地执行一个程序。我们使用 watch 运行上面的命令来监控熵的变化。下面的命令每 0.1 秒就执行一次 cat 程序。

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

请执行上面的命令。在它运行时，移动你的鼠标，点击鼠标，输入什么东西，读取一个大文件，访问一个网站。哪些活动会显著地使熵变大？请在报告中描述你的发现。

2.4 任务 4: 从 /dev/random 中获取伪随机数

Linux 将从物理资源收集的随机数据存储到一个随机池中，然后使用两个设备将随机源转换为伪随机数。这两个设备是 /dev/random 和 /dev/urandom 。它们有不同的行为。/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。

让我们设计一个实验来观察 /dev/random 设备的行为。我们将使用 cat 命令持续从 /dev/random 中读取伪随机数。我们将输出通过管道传递到 hexdump 以便获得良好的输出。

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

请运行上面的命令，同时使用 watch 命令来监视熵。如果不移动鼠标，也不键入任何内容，将会发生什么。然后，随机移动鼠标，看看是否可以观察到任何差异。请描述并解释你观察到的现象。

问题 假设一个服务器使用 /dev/random 与客户端生成随机会话密钥。请描述你将如何对这样的一个服务器发起拒绝服务（DoS）攻击。

2.5 任务 5: 从 /dev/urandom 获取随机数

Linux 提供了另一种方式,可以通过 /dev/urandom 设备访问随机池。/dev/random 和 /dev/urandom 都可以使用随机池中的数据生成伪随机数。当熵不足时，/dev/random 将会暂停，而 /dev/urandom 会继续生成新的数。将随机池中的数据视作“种子”，我们可以使用种子想生成多少随机数就生成多少。

我们来看看 /dev/urandom 的行为。我们再次使用 cat 从设备中获取伪随机数。请运行下面的命令，描述移动鼠标是否会影响结果。

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

让我们测量随机数的质量。我们可以使用一个名为 ent 的工具，该工具已经安装在我们的 VM 中。根据其手册，“ ent 对存储在文件中的字节序列进行各种测试，并报告这些测试的结果。该程序对于评估加密和统计采样应用程序，压缩算法以及其他文件信息密度受关注的其他应用程序的伪随机数生成器很有用”。让我们首先从 /dev/urandom 生成 1 MB 的伪随机数，并将其保存在文件中。然后，在文件上运行 ent。请描述你的结果，并分析随机数的质量是否良好。

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

理论上讲，/dev/random 设备更加安全，但是实践上并没有很大的差异，因为 /dev/urandom 使用的“种子”是随机的和不可预测的（当有新的随机数据输入时 /dev/urandom 会重新设置种子）。一个比较大的问题是 /dev/random 的阻塞行为可能导致拒绝服务攻击。因此，比较推荐使用 /dev/urandom 来获取随机数。要在我们的程序中这样做，只需要直接从这个设备文件中读取。下面的代码片段展示了如何实现。

```
#define LEN 16  // 128 bits

unsigned char *key = (unsigned char *) malloc(sizeof(unsigned char)*LEN);
FILE* random = fopen("/dev/urandom", "r");
fread(key, sizeof(unsigned char)*LEN, 1, random);
fclose(random);
```

请修改上面的代码片段来生成一个 256 bit 的加密密钥。请编译并运行你的代码，输出这些数并将屏幕截图放在报告里。

3 Submission

你需要提交一份带有截图的详细实验报告来描述你所做的工作和你观察到的现象。你还需要对一些有趣或令人惊讶的观察结果进行解释。请同时列出重要的代码段并附上解释。只是简单地附上代码不加以解释不会获得学分。