41900 – Security Fundamentals
PRNG - Basic Encryption Decryption

OVERVIEW

The learning objective of this lab is to get familiar with the process used to generate pseudo random numbers which are used to create encryption keys. The lab also gives an overview of the concept of encryption.

TASKS

Task 1: Pseudo Random Number Generation

Generating random numbers is a quite common task in security software. In many cases, encryption keys are not provided by users, but are instead generated inside the software. Their randomness is extremely important; otherwise, attackers can predict the encryption key, and thus defeat the purpose of encryption. Many developers know how to generate random numbers (e.g. for Monte Carlo simulation) from their prior experiences, so they use the similar methods to generate the random numbers for security purpose. Unfortunately, a sequence of random numbers may be good for Monte Carlo simulation, but they may be bad for encryption keys. Developers need to know how to generate secure random numbers, or they will make mistakes. Similar mistakes have been made in some well-known products, including Netscape and Kerberos.

In this task, students will learn a standard way to generate pseudo random numbers that are good for security purposes.

Task 1.1: Measure the Entropy of Kernel

To generate good pseudo random numbers, we need to start with something that is random; otherwise, the outcome will be quite predictable. Software (i.e. in the virtual world) is not good at creating randomness, so most systems resort to the physical world to gain the randomness. Linux gains the randomness from the following physical resources:

```
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);
```

The first two are quite straightforward to understand: the first one uses inter-keypress timing and scan code, and the second one uses mouse movement and interrupt timing. The third one gathers random numbers using the interrupt timing. Of course, not all interrupts are good sources of randomness. For example, the timer interrupt is not a good choice, because it is predictable. However, disk interrupts are a better measure. The last one measures the finishing time of block device requests.

41900 – Security Fundamentals
PRNG - Basic Encryption Decryption

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

- **`cat /proc/sys/kernel/random/entropy_avail`**

Please move and click your mouses, type somethings, and run the program again. Please describe your observation in your report.

Task 1.2: 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 have different behaviors. In this subtask, we study the **`/dev/random`** device.
You can use the following command to get 16 bytes of pseudo random numbers from **`/dev/random`**. We pipe the data to hexdump to print them out.

- **`head -c 16 /dev/random | hexdump`**

Please run the above command several times, and you will find out that at some point, the program will not print out anything, and instead, it will be waiting. Basically, every time a random number is given out by **`/dev/random`**, the entropy of the randomness pool will be decreased. When the entropy reaches zero, **`/dev/random`** will block, until it gains enough randomness. Please show us how you can get **`/dev/random`** to unblock and to print out random data.

Task 1.3: Get 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, even if the entropy of the pool runs low.

You can use the following command to get 1600 bytes of pseudo random numbers from **`/dev/urandom`**. You should run it several times, and report whether it will block or not.

- **`head -c 1600 /dev/urandom | hexdump`**

Both **`/dev/random`** and **`/dev/urandom`** use the random data from the pool to generate pseudo random numbers. When the entropy is not enough, **`/dev/random`** will pause, while **`/dev/urandom`** will keep generating new numbers. Think of the data in the pool as the “seed”, and as we know, you can use a seed to generate as many pseudo random numbers as you want. Theoretically speaking, the **`/dev/random`** device is more secure, but in practice, there is not much difference, because the "seed" is random and nonpredictable. **`/dev/random`** does re-seed whenever new random data become available. The fact that **`/dev/random`** blocks may lead to denial of service attacks.

Task 2: Basic encryption and decryption using OpenSSL

In this task, you will use various encryption algorithms and modes on the Text File and Image File provided. You are required to save each type of encrypted file separately and observe if each encryption technique gives the same or different results. Please take screenshots to show your observations (as the image file are large, only list your observations for it. The task is to be performed using a Terminal.

You can begin by reading the manual for openssl and enc by typing in

```
➢ man openssl enc
```

Use the PRNG to generate a key (size defined by the algorithm used) to be used for encryption. Use the same key and cryptographic method to decrypt the cipher text file to original plain text.

You can use the following command to encrypt/decrypt a file.

```
➢ openssl enc <cipher-type> -e/-d -in <> -out <> -K <> -iv <>
```

- **`-e`** `encrypt`
- **`-d`** `decrypt`
- **`-in`** `input file`
- **`-out`** `output file`
- **`-K`** `secret key (in hexadecimal)`
- **`-iv`** `initialization vector (in hexadecimal)`

`<cipher-type>` some types include:

- Cipher Block Chaining (**`CBC`**)
- Cipher Feedback (**`CFB`**)
- Electronic Codebook (**`ECB`**)
- Output Feedback (**`OFB`**)