**1 Task1: Generate Encryption Key in a Wrong Way**

2 Task2: Guessing the Key

OnApril17,2018,Aliceﬁnishedhertaxreturn,andshesavedthereturn(aPDFﬁle)onherdisk. Toprotect the ﬁle, she encrypted the PDF ﬁle using a key generated from the program described in Task 1. She wrote down the key in a notebook, which is securely stored in a safe. A few month later, Bob broke into her computer and gets a copy of the encrypted tax return. Since Alice is CEO of a big company, this ﬁle is very valuable. Bob cannot get the encryption key, but by looking around Alice’s computer, he saw the key-generation program, and suspected that Alice’s encryption key may be generated by the program. He also noticed the timestamp of the encrypted ﬁle, which is "2018-04-17 23:08:49". He guessed that the key may be generated within a two-hour window before the ﬁle was created. Since the ﬁle is a PDF ﬁle, which has a header. The beginning part of the header is always the version number. Aroundthetimewhentheﬁlewascreated,PDF-1.5wasthemostcommonversion,i.e.,theheader starts with %PDF-1.5, which is 8 bytes of data. The next 8 bytes of the data are quite easy to predict as well. Therefore, Bob easily got the ﬁrst 16 bytes of the plaintext. Based on the meta data of the encrypted ﬁle, he knows that the ﬁle is encrypted using aes-128-cbc. Since AES is a 128-bit cipher, the 16-byte plaintext consists of one block of plaintext, so Bob knows a block of plaintext and its matching ciphertext. Moreover, Bob also knows the Initial Vector (IV) from the encrypted ﬁle (IV is never encrypted). Here is what Bob knows:

Plaintext: 255044462d312e350a25d0d4c5d80a34 Ciphertext: d06bf9d0dab8e8ef880660d2af65aa82 IV: 09080706050403020100A2B2C2D2E2F2

Your job is to help Bob ﬁnd out Alice’s encryption key, so you can decrypt the entire document. You should write a program to try all the possible keys. If the key was generated correctly, this task will not be possible. However, since Alice used time() to seed her random number generator, you should be able to ﬁnd out her key easily. You can use the date command to print out the number of seconds between a speciﬁed time and the Epoch, 1970-01-01 00:00:00 +0000 (UTC). See the following example.

SEEDLabs–PseudoRandomNumberGenerationLab 3

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

2.3 Task3: MeasuretheEntropyofKernel Inthevirtualworld,itisdifﬁculttocreaterandomness,i.e.,softwarealoneishardtocreaterandomnumbers. 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 the timing between key presses; 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. Therandomnessismeasuredusingentropy,whichisdifferentfromthemeaningofentropyintheinformation 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

Let us monitor the change of the entropy by running the above command via watch, which executes a program periodically, showing the output in fullscreen. The following command runs the cat program every 0.1 second.

$ watch -n .1 cat /proc/sys/kernel/random/entropy\_avail

Please run the above command. While it is running, move your mouse, click your mouse, type somethings, read a large ﬁle, visit a website. What activities increases the entropy signiﬁcantly. Please describe your observation in your report.

2.4 Task4: GetPseudoRandomNumbersfrom /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,everytimearandomnumberisgivenoutbythisdevice,theentropyoftherandomnesspoolwillbe decreased. When the entropy reaches zero, /dev/random will block, until it gains enough randomness. Letusdesignanexperimenttoobservethebehaviorofthe/dev/randomdevice. Wewillusethecat commandtokeepreadingpseudorandomnumbersfrom /dev/random. Wepipetheoutputto hexdump for nice printing.

$ cat /dev/random | hexdump

Please run the above command and at the same time use the watch command to monitor the entropy. 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. Please describe and explain your observations.

SEEDLabs–PseudoRandomNumberGenerationLab 4

Question: If a server uses /dev/random to generate the random session key with a client. Please describe how you can launch a Denial-Of-Service (DOS) attack on such a server.

2.5 Task5: GetRandomNumbersfrom /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 willkeepgenerating newnumbers. Thinkofthedata inthepoolasthe “seed”, and aswe know, we can use a seed to generate as many pseudo random numbers as we want. Let us see the behavior of /dev/urandom. We again use cat to get pseudo random numbers from this device. Please run the following command, and the describe whether moving the mouse has any effect on the outcome.

$ cat /dev/urandom | hexdump

Letusmeasurethequalityoftherandomnumber. Wecanuseatoolcalledent,whichhasalreadybeen installedinourVM.Accordingtoitsmanual,“entappliesvariousteststosequencesofbytesstoredinﬁles andreportstheresultsofthosetests. Theprogramisusefulforevaluatingpseudo-randomnumbergenerators for encryption and statistical sampling applications, compression algorithms, and other applications where the information density of a ﬁle is of interest”. Let us ﬁrst generate 1 MB of pseudo random number from /dev/urandom and save them in a ﬁle. Then we run ent on the ﬁle. Please describe your outcome, and analyze whether the quality of the random numbers is good or not.

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

Theoretically speaking, the /dev/random device is more secure, but in practice, there is not much difference,becausethe“seed”usedby /dev/urandom israndomandnon-predictable(/dev/urandom does re-seed whenever new random data become available). A big problem of the blocking behavior of /dev/random isthatblockingcanleadtodenialofserviceattacks. Therefore, itisrecommendedthatwe use /dev/urandom togetrandomnumbers. Todothatinourprogram,wejustneedtoreaddirectlyfrom this device ﬁle. The following code snippet shows how.

#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);

Pleasemodifytheabovecodesnippettogeneratea256-bitencryptionkey. Pleasecompileandrunyour code; print out the numbers and include the screenshot in the report.