PRN_LabReport

Task1: Generate Encryption Key in Wrong Way

- How can we make an outcome unpredictable?

In comparison with the pseudo-random number generator (PRNG), the true random number generator (TRNG) relies on gadgets that promise to create really random numbers.

The gadgets are unpredictable rather than human-defined patterns like an algorithm. For really random numbers, the computer must employ an unpredictable external physical variable, such as radioactive decay of isotopes or airwave static, rather than.

- What is the purpose of srand() and time()?

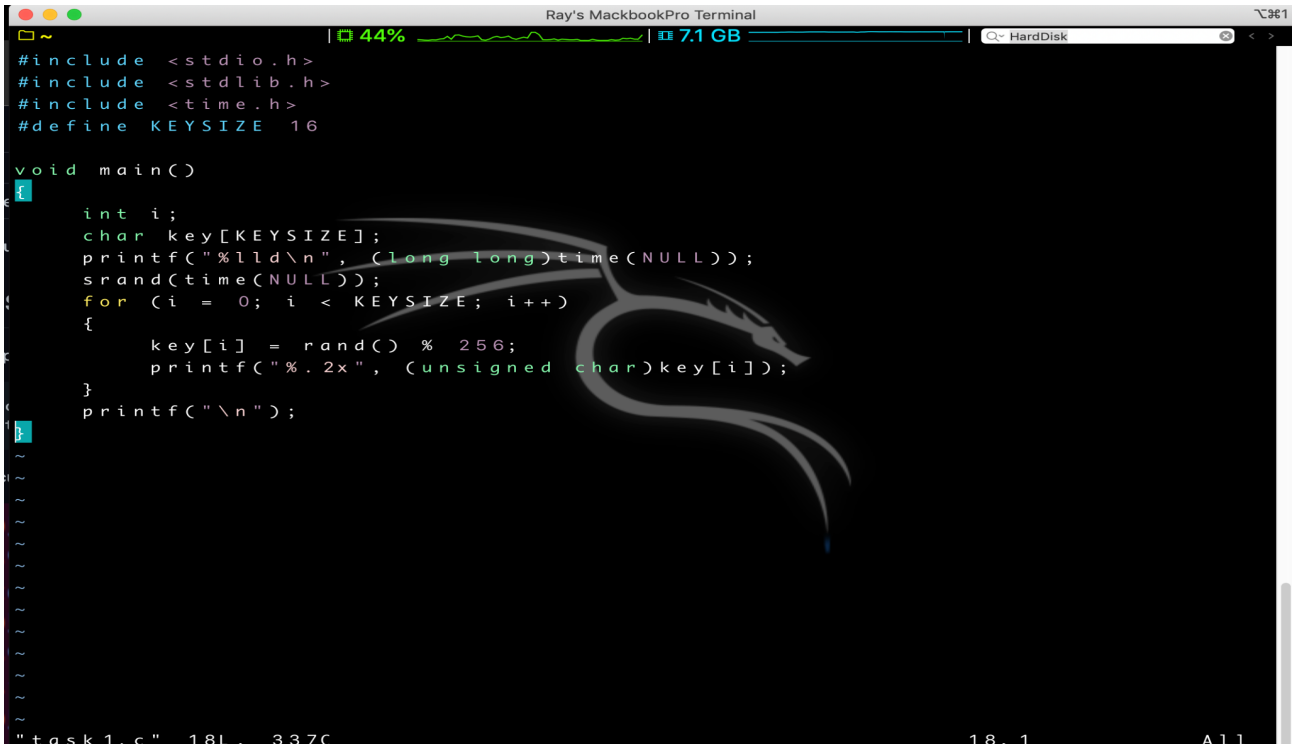
The purpose of srand() is that pass the argument seed to start the pseudo-random number generator. And the time() function means generating a time-based input as an argument for the seed.

- What is the result of commenting out the srand()?

The procedure will not be involved with creating an initial input for generating pseudo-random numbers if srand() is commented out. So, every time the program runs, it will generate an identical series of integers.

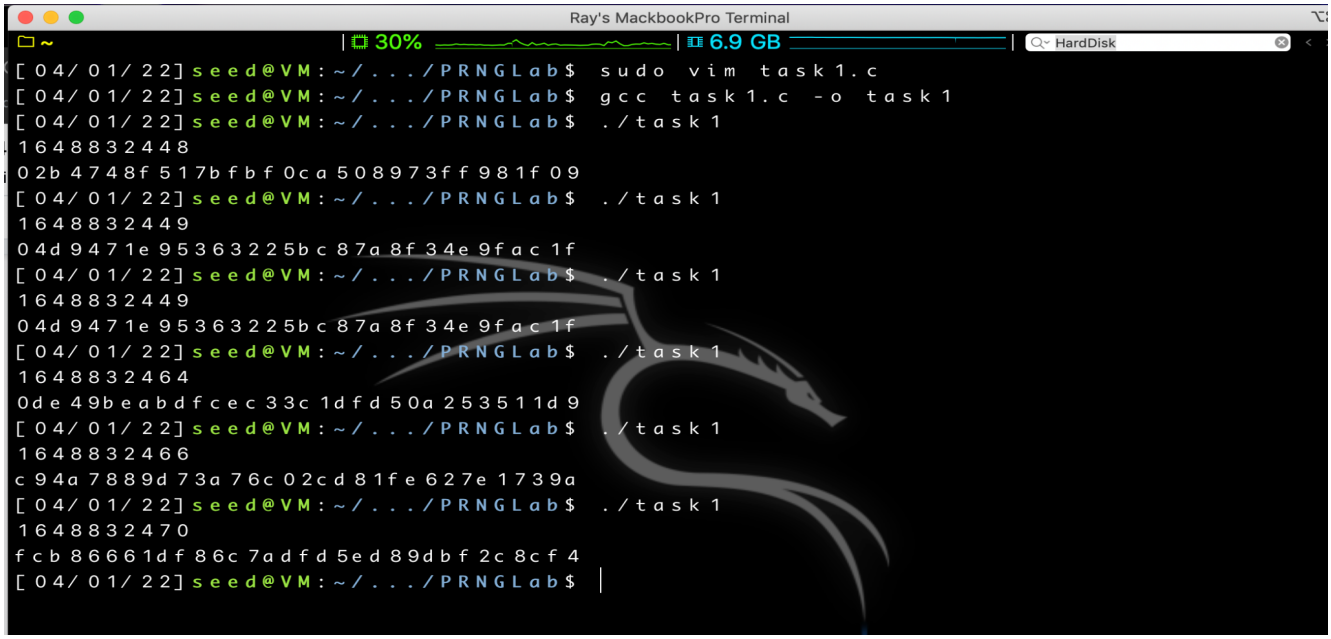
Task 1 Screenshots and Observations

Generate random numbers **with srand() function**

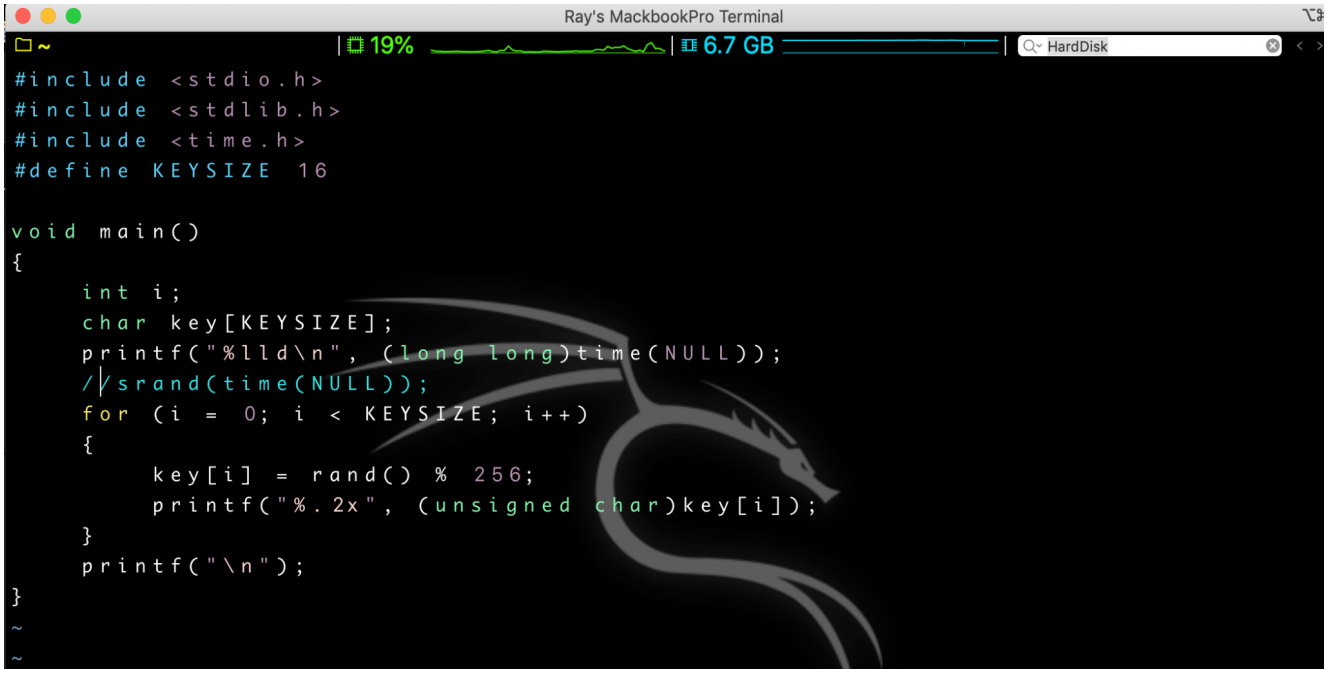


Observation:

Because it generates random numbers using the current time as a random seed, the seed is always different in each run.



Generate random numbers **without srand() function**



Observation:

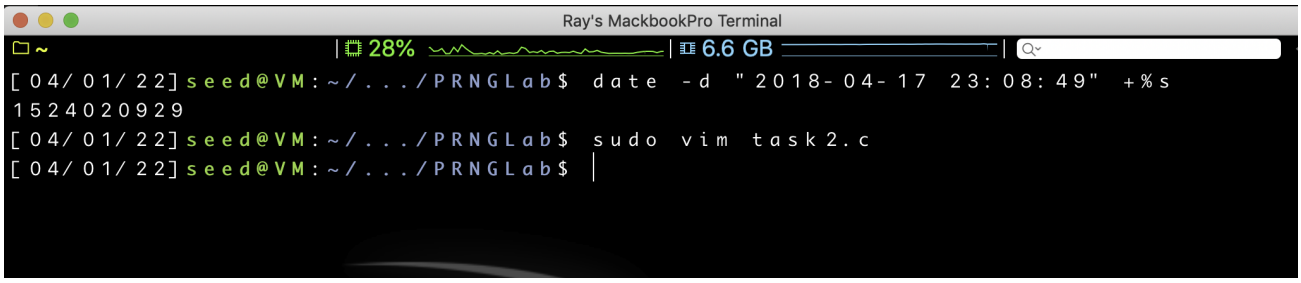
Regarding commenting out srand() function, the program does not use the current time as a seed to generate random numbers, the random numbers are always the same in each run.

Task2: Guessing the Key

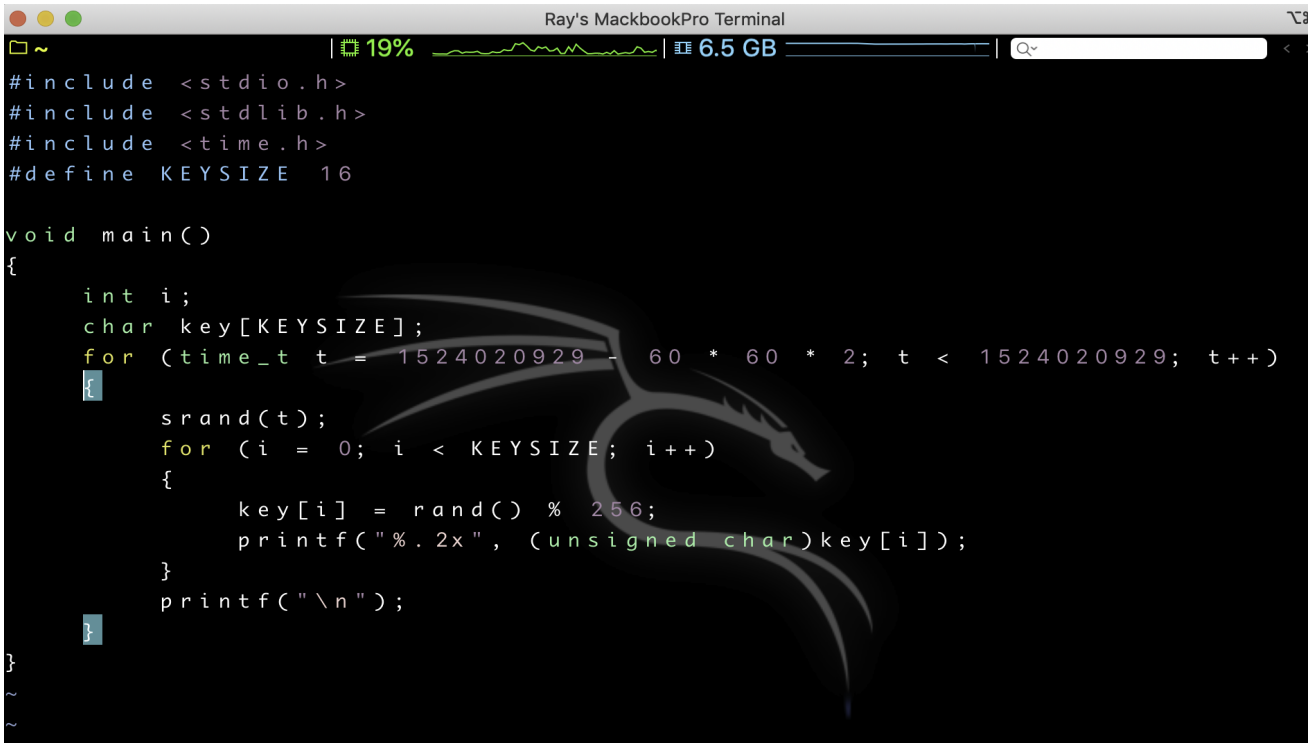
The purpose of Task 2 is to decrypt the full document using the encryption key. Bob surmised that the key would be generated during a two-hour period before the file was created.

Task 2 Screenshots and Observations

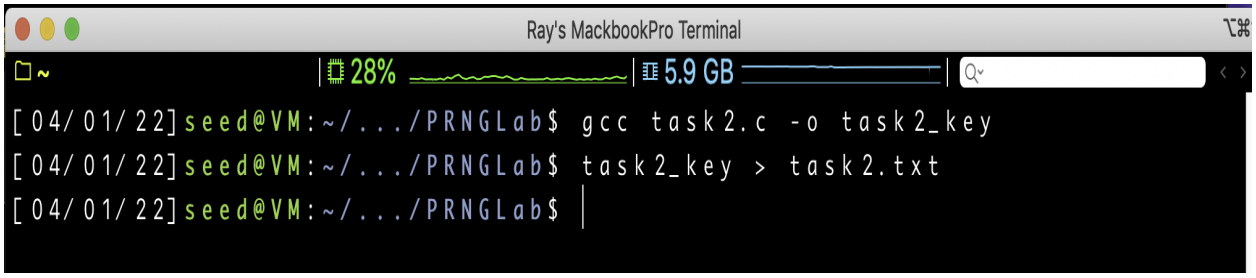
The timestamp of the encrypted file is "2018-04-17 23:08:49" so calculate its number of seconds.



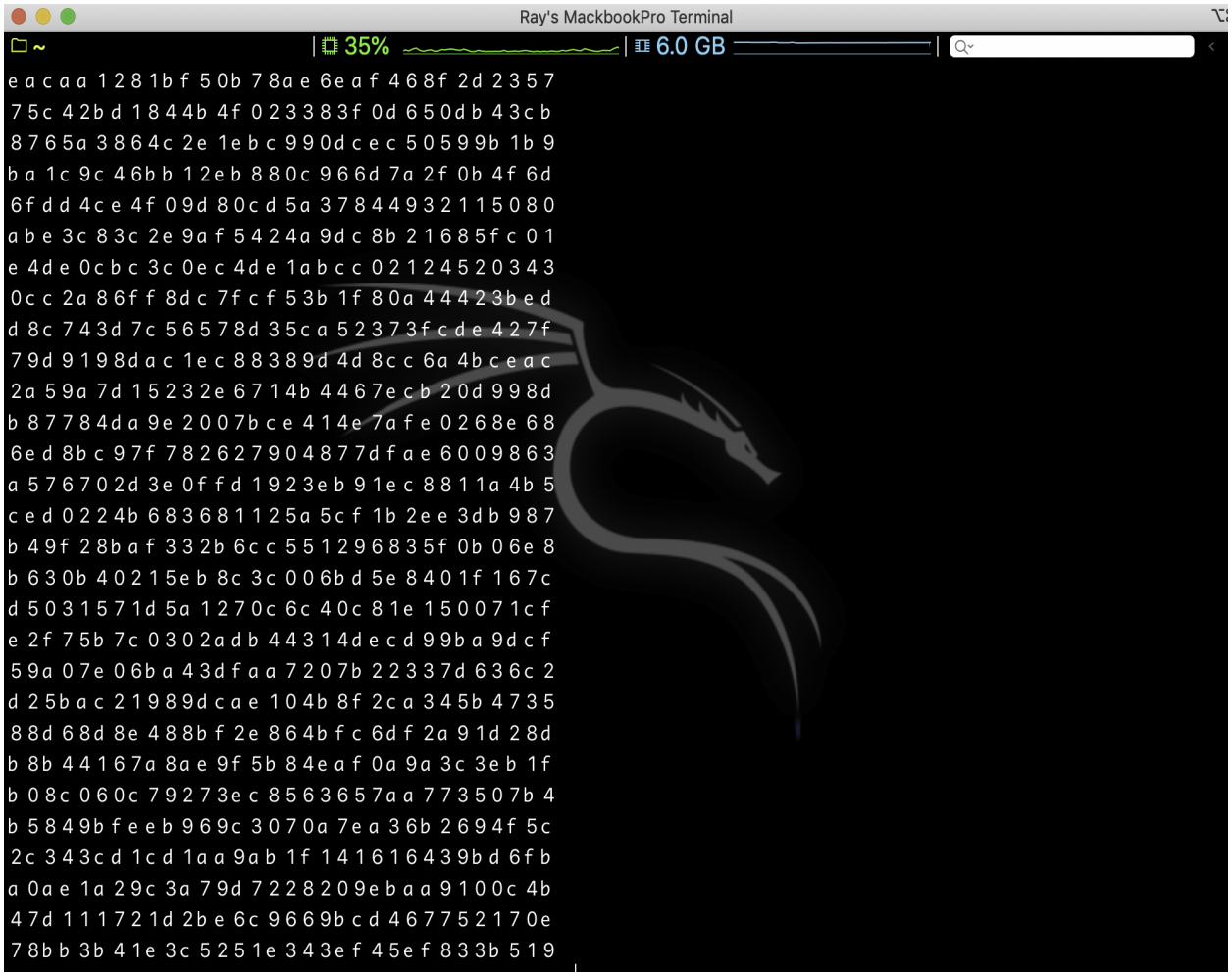
The below program can list the possible pseudo-random generator numbers. And the file was created during a two-hour window so the program needs to minus the number of seconds in 2 hours total.



GCC compiles C program and redirects stdout to in a .txt file as a dictionary



The below screenshot shows the possible pseudo-random generator numbers which are used to encrypt the file.



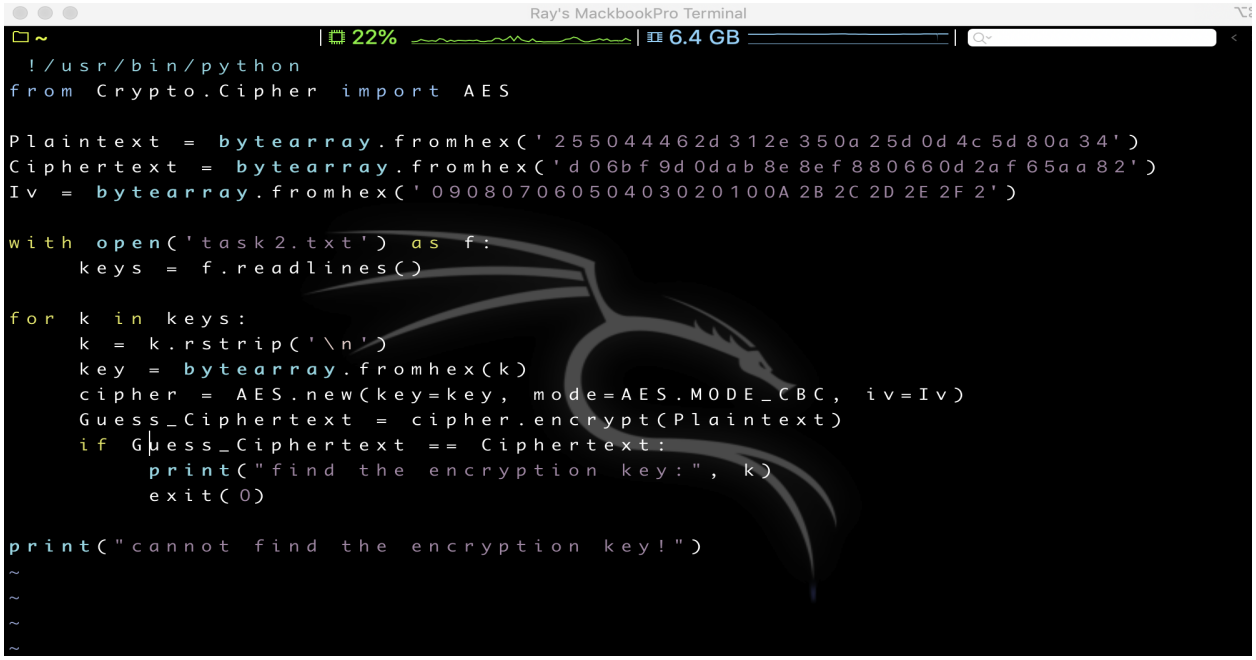
As given the information as belows,

Plaintext: 255044462d312e350a25d0d4c5d80a34

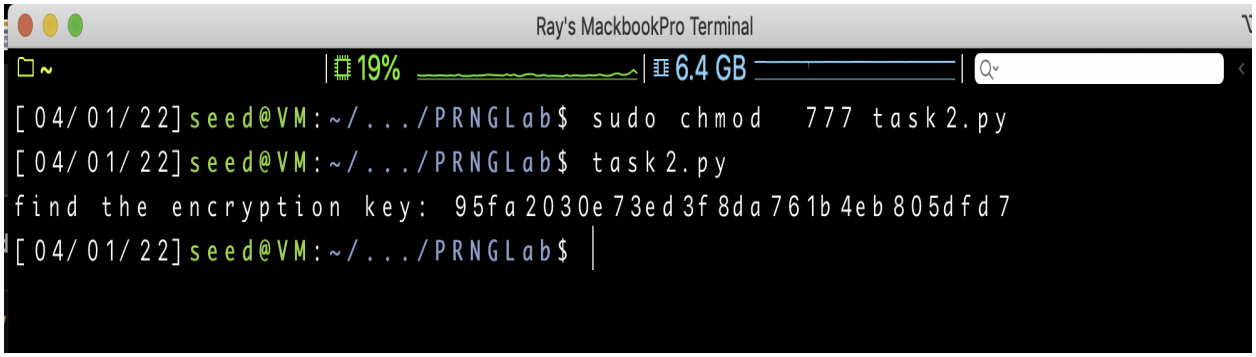
Ciphertext: d06bf9d0dab8e8ef880660d2af65aa82

IV: 09080706050403020100A2B2C2D2E2F2

To crack the key from task2.txt, utilize a brute-force approach (dictionary approach).



Find the encryption key: **95fa2030e73ed3f8da761b4eb805dfd7**



Observation:

Utilize the given date converting to the number of seconds to generate the possible pseudo-random numbers, and store these numbers in a dictionary file. Then, guess the encryption key by brute force way with the given plaintext, ciphertext, IV and encryption algorithm.

Task 3: Measure the entropy of the Kernel

- From what resources does Linux get randomness? (name 3 and explain each one).

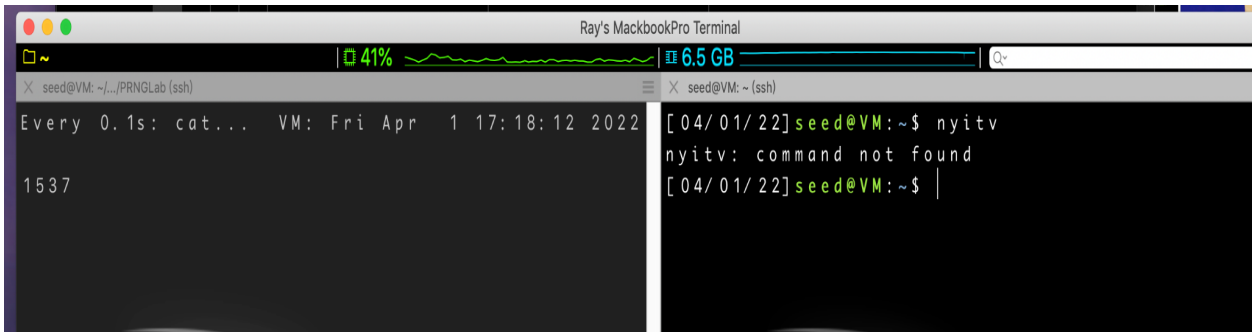
1. input randomness: This is usually involved in games, it is when something random happens before the player has any input into the game.
2. device randomness: an evenly-distributed integer random number generator that generates non-deterministic random numbers
3. environmental noise randomness: This refers to an external source of randomness, such as the user's interactions with the keyboard and mouse, as well as the signal on an audio input line.

- What does entropy mean?

The meaning of entropy is the randomness gathered by a system for use in algorithms that require random data, which simply refers to how many random number bits the system currently possesses. In other words, entropy is a measure of the unpredictability or variety of a data-generating function in cyber security. A lack of good entropy might leave a cryptosystem insecure and unable to encrypt data securely.

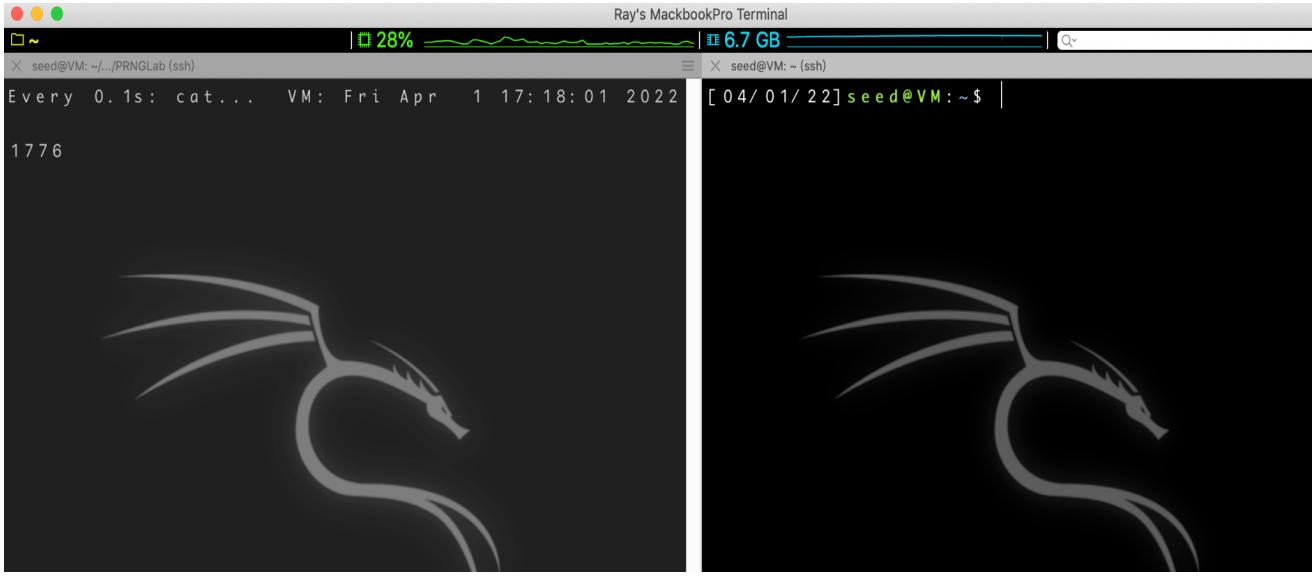
Task 3 Screenshots and Observations

When I input short words, it appears that the entropy figures climb slowly.



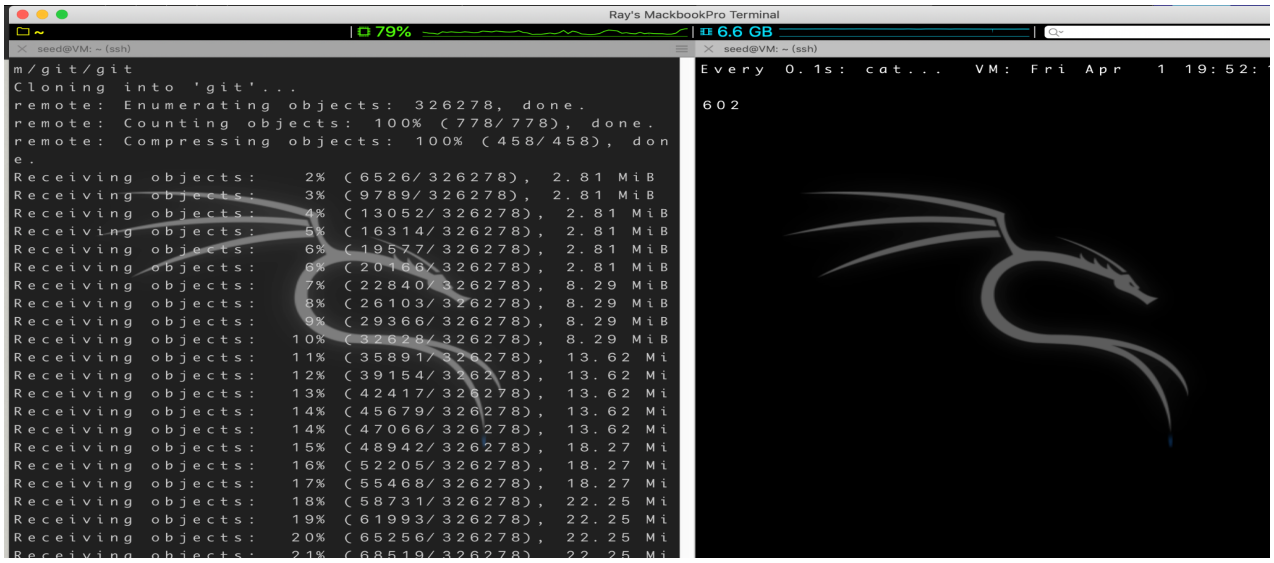
The below screenshots show that printing the content of a 76 bytes file via the cat command, increases the entropy number quickly compared to typing some words and moving the mouse.

Furthermore, if I move the mouse quickly, the entropy number also climbs tediously.



The below screenshot show when I open a new monitor window and clone a file from Github,

The number of entropy increases rapidly which is the fastest in this experiment such as moving the mouse, typing some words and printing the content of the file.



Overall, according to the observations, it might say that **/dev/random** utilizes the available entropy generated by user interactions to generate new random numbers.

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

- Where does Linux store the collected random data?

The Linux kernel gathers noise from device drivers and other sources and stores them in its entropy pool. The generator also maintains track of the number of noise bits in the entropy pool. The random numbers are generated from this entropy pool.

- What are “/dev/random” and “/dev/urandom”?

The meaning of /dev/random provides a maximum of the number of bits of randomness stored in the entropy pool. Also, /dev/random should be suited for applications that require very high-quality randomization. When the entropy pool is reduced, reads from /dev/random will be blocked until more environmental noise is obtained.

And, the meaning of /dev/unrandom is reading from /dev/urandom will not be interrupted while waiting for more entropy. As a result, if the entropy pool is reduced, the returned values are theoretically open to a cryptographic attack on the driver's algorithms.

- What is the difference between “/dev/random” and “/dev/urandom”?

The difference between “/dev/random” and “/dev/urandom” is that “/dev/random’’ may necessitate some waiting for an entropy pool, where random data may not be accessible at the time while ‘’/dev/urandom” only responses the number of bytes requested by the user, making it less unpredictable than /dev/random.

- What is your observation for this task if you do not move your mouse or type anything?

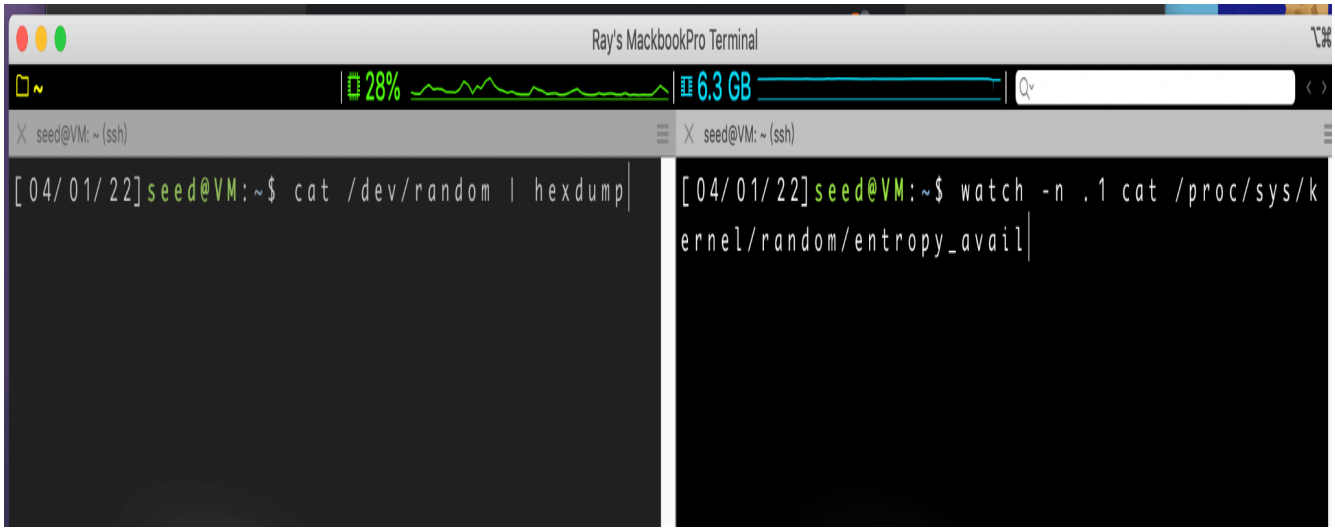
When I remain motionless, the number of entropy generates slowly and almost becomes stuck.

- What is your observation for this task if you move your mouse or type anything?

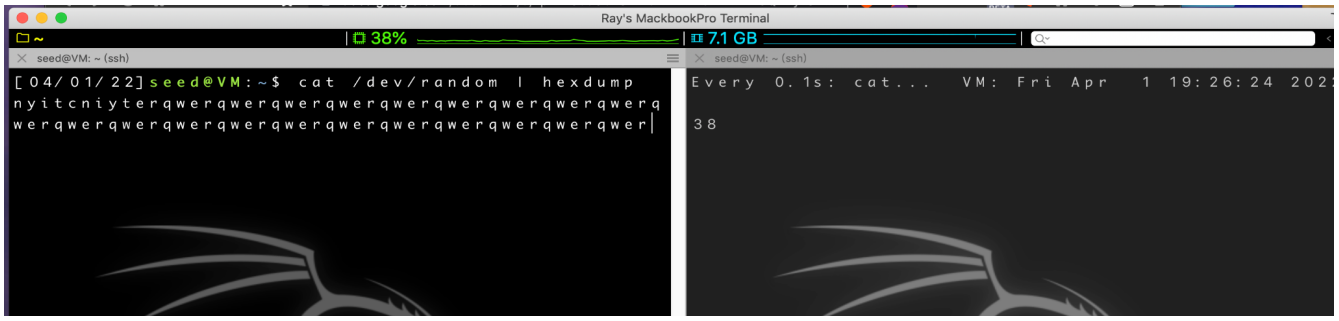
When typing some words on the left window and moving the mouse, it is indicated that the number of entropy increases on the right monitor window.

Task 4 Screenshots and Observations

The below screenshots show the /dev/random entropy usage and monitor simultaneously.



When typing some words on the left window and moving the mouse, the number of entropy increases.



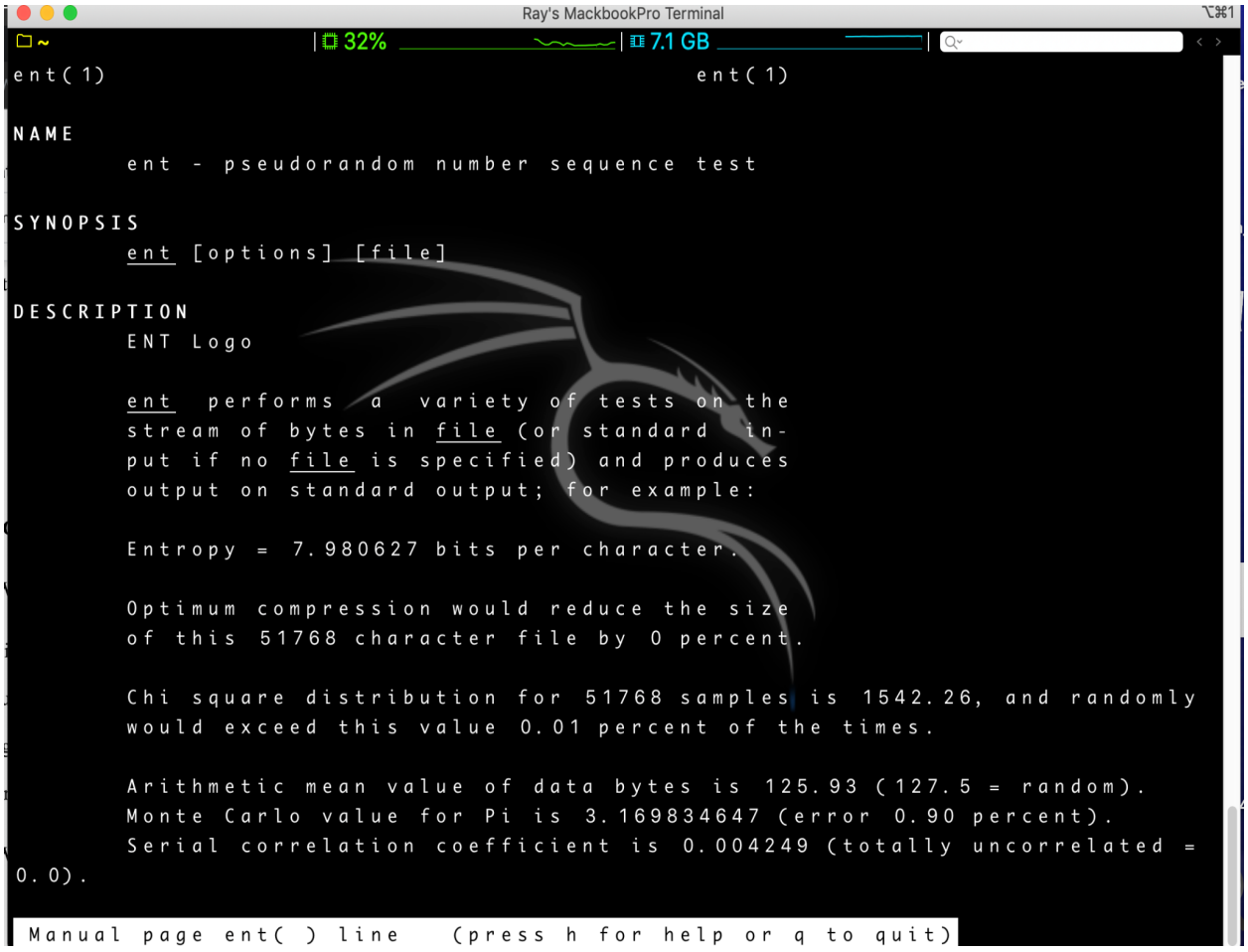
When I stop typing some words on the left window and stop moving the mouse, the number of entropy seems stuck and if the number of entropy decreases, some new line of random numbers appears.

Task 5: Get Random Numbers from /dev/urandom

- What is ent tool?

Ent tool is a good command-line utility for calculating entropy in a simple and quick manner and does a number of tests on the file's bytes stream. It can be used to evaluate pseudo-random number generators for encryption and statistical sampling applications, compression methods, and other applications where the information density of a file is important.

The below screenshot shows manual ent via the “man ent” command.



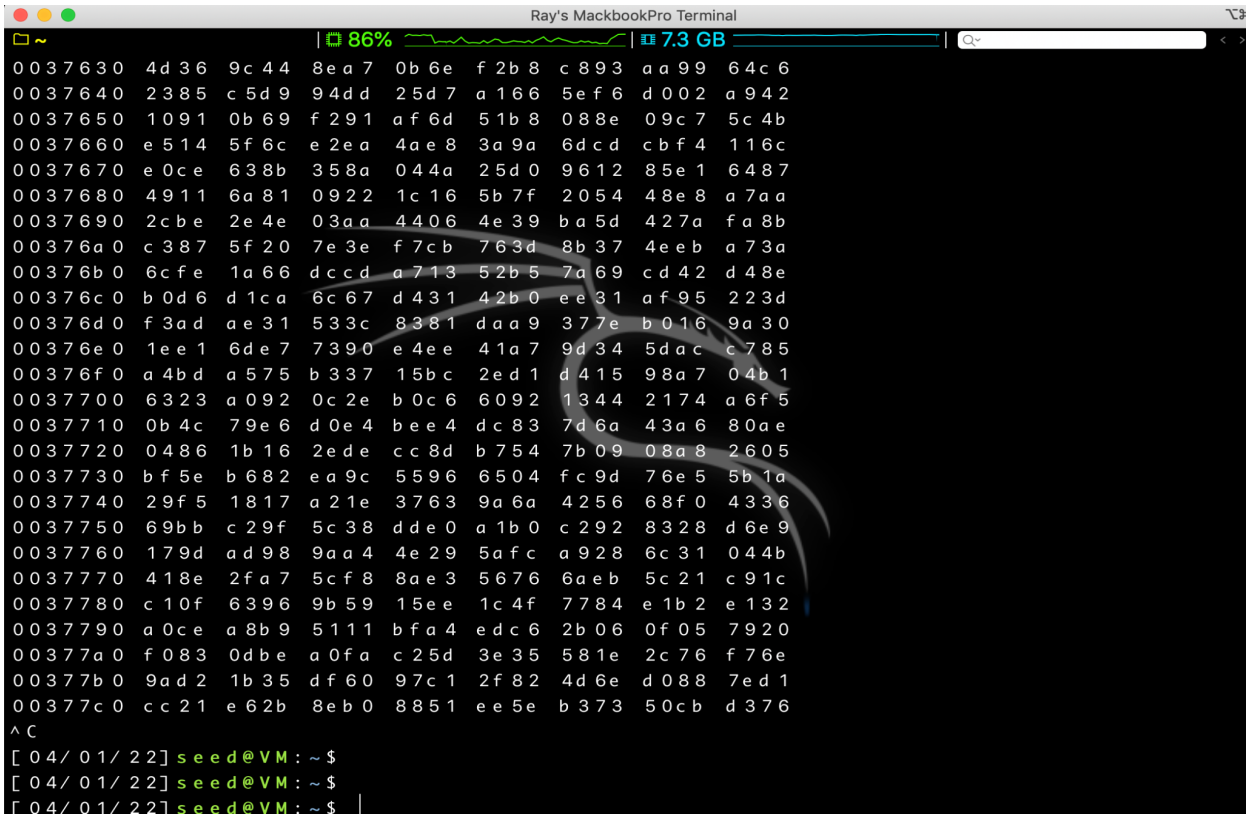
- Which one is more secure “/dev/urandom” or “/dev/random”?

The /dev/random is more secure than the dev/urandom.

The fundamental distinction between /dev/random and /dev/urandom is that /dev/random blocks if the entropy does not indicate adequate randomness, whereas /dev/urandom never blocks, even if the pseudo-random number generator is not fully seeded when booted. In short, /dev/random is the safest, followed by /dev/urandom, which is the least safe.

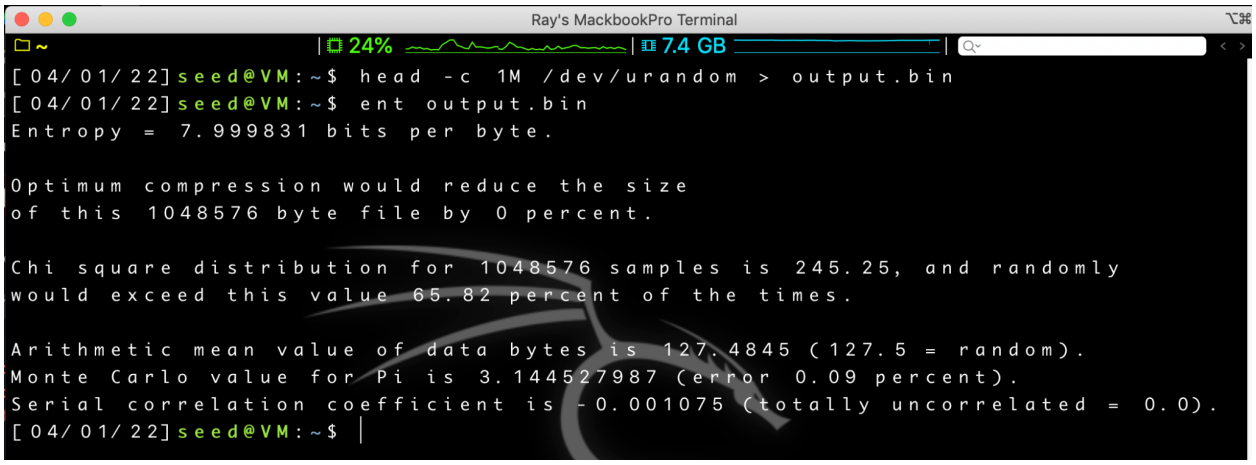
Task 5 Screenshots and Observations

The below screenshots shows the result of "cat /dev/urandom | hexdump", which indicates that it continues to generate random integers.

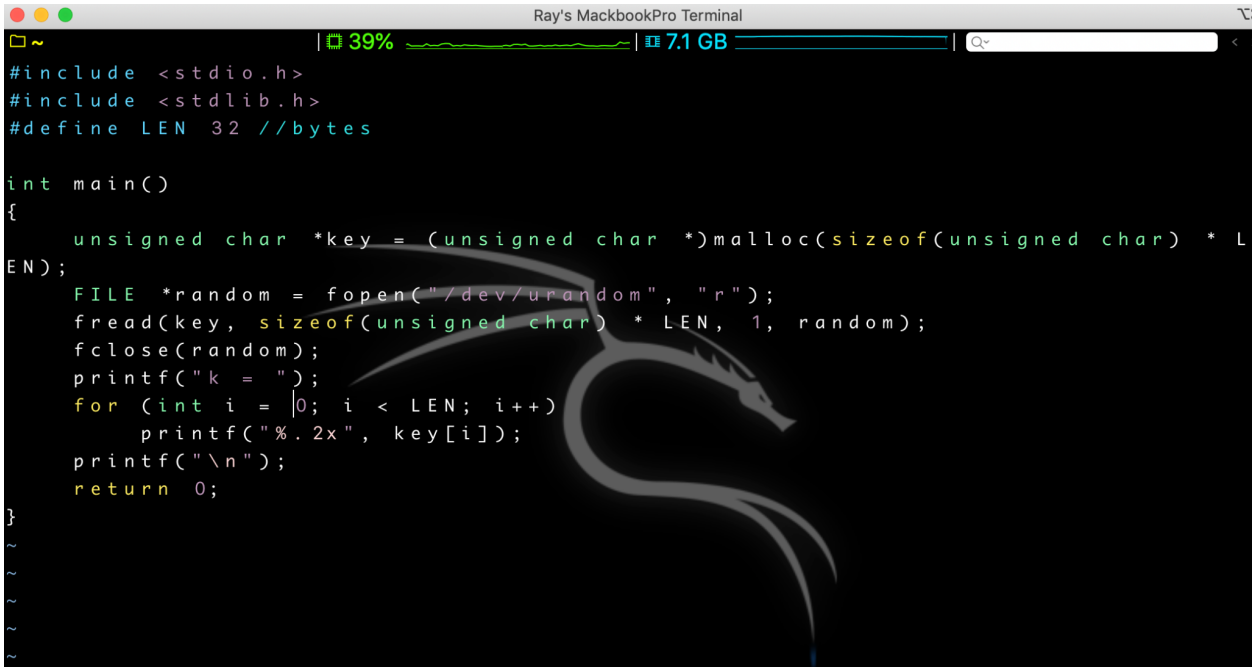


Because of continually generating random integers, use head to generate 1 MB of pseudo-random numbers from /dev/urandom and save them to output.bin.

And analyzing the result via “ent” tool.

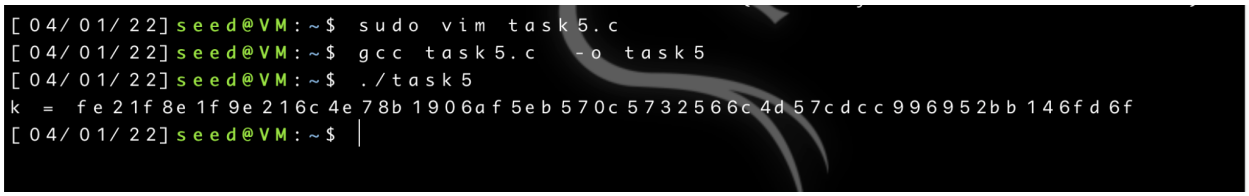


One major issue with /dev/random its nature is that it can lead to denial of service attacks. As a result, the below program utilizes /dev/urandom to generate 256 bits of random numbers.



After the running program, the result is

fe21f8e1f9e216c4e78b1906af5eb570c5732566c4d57cdcc996952bb146fd6f



References

Understanding and Managing Entropy (n.d.). Whitewood Encryption Systems. https://www.blackhat.com/docs/us-15/materials/us-15-Potter-Understanding-And-Managing-Entropy-Usage-wp.pdf

Stack overflow. What keeps draining entropy?. https://unix.stackexchange.com/questions/96847/what-keeps-draining-entropy

random add_input_randomness.

https://docs.huihoo.com/doxygen/linux/kernel/3.7/drivers_2char_2random_8c.html

/dev/random vs /dev/urandom and are they secure?. https://linuxhint.com/dev_random_vs_dev_urandom/