Assignment #4

CPEN 442

November 2, 2016

Weijun Qiu #42701136
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

I. Problem #1

Key: 9786
CPU time used: 0.008s
Code: pwCracker.c

First I guessed that the type of hash is SHA1, since the length of the string given is 22, including a 2-character salt. Then, I wrote a program to brute force this problem. Basically, try all possible keys in order until I find one with the same hash.

II. Problem #2

Key: w$c_##
CPU time used: 5h0m51.728s
Code: pwCracker2.c

I guessed it is hash with SHA1 as I did in question1. I modified the program I wrote in question1 to specialize it in brute forcing this problem. The real time elapsed is about half of the recorded CPU time, since I was using 2 threads. The reason I wrote my program is because I did not know we are allowed to use third party tools in this assignment. Afterwards, I also tried using JohnTheRipper to solve this problem, which was also successful but with a slightly longer time.

III. Problem #3

Key: 2K4-T%V_ZZ_^FvE)R
Patch: 42701136.program1.diff

I solved this problem using IDA. The program flow graph it generated was very helpful for me to identify that the program was doing string comparison by comparing byte by byte. Var_14 stores the index of the for-loop, and [42D000 + var_14 + 3Ah] was reading a byte from the correct password, which is stored as a global variable. Also, I figured out that the length of the password is 17 bytes, since before the program enters the for-loop it checks that if the entered string has a length of 17. Given the above facts, It is clear that the key is the 17 bytes starting from the offset 42D000+3A. No computation time is need in get the password.

The program normally checks if the entered password and the stored password are the same byte by byte, and if it sees a byte is different, it put a 0 into var_D and exit the loop. Value in var_D will later be used in a conditional jump to jump to either the success or fail branches. The patch basically changes the value that is put into var_D from 0 to 1, and user will always be able to login.

IV. Using the Template

Key: 9m7R2L
CPU time used: 44m15s
Patch: 42701136.program2.diff
Program: pwSetter.c

The program flow is almost identical to the one in problem 3, except there is an extra function call to Libeay32_501, which is a openssl library. Thus, I was able to guess that the password entered will be hashed and compared to the hash stored internally. The target hash is 65CD30D51BF1D197AD61163AC0C3103B4A6D5A20 and I used JohnTheRipper to find a key that has the same SHA1 hash.

The reasoning of the patch is exactly the same as the one in problem 3.

The program I wrote will take a user input and hash it using SHA1, and write the resulting hash over the stored hash in the exe file. The program is expected to be run on macOS. An example output file is 42701136.program2.modified.exe, which has a password of 123.