Assignment #4

CPEN 422

November 1, 2015

butterCHICKEN
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

I. Password Cracking

A. Problem #1

Recovered password: 6845
Computation time: 2 seconds (approx.)
Password entropy: 13 bits

$$e = log_2 10^4 = 13.29 \approx 13$$

This password was recovered using a brute force approach. Given that the keyspace is small, the only thing left to do after identifying the correct hash function is to look through all possible 4-digit combinations until one of them matches the original hashed value. The original hashed value is in hex notation, except for the first two characters, so I assumed the first two characters were either a diversion or they were used as a salt. Without the first two characters, the hashed value is 40 characters (160 bits) long, so I found the hash functions that output 160-bit long hashes and wrote a python script to compute and compare hash values. I did not find any matches until I used the first two characters as a salt with the SHA1 hash function.

B. Problem #2

Recovered password: ekjCO7
Computation time: 30 minutes (approx.)
Password entropy: 37 bits

$$e = log_2 76^6 = 37.49 \approx 37$$

I attempted to write another script for a brute force recovery, similar to problem 1. I assumed that, like Problem 1, SHA1 had been used to hash the password and that the two characters preceeding the hashed value were the salt. My script was not fast enough. I tried improving time efficiency by using threads, but even with threads, at the rate my script was running, it would have taken about 60 days, on average, to find the password (120 days to search through all possibile passwords), so instead of my script, I used a tool called hashcat to recover the password. I ran the hashcat executable with the following arguments:

-m 100 -a 3 -1 (alphabet) Ly?1?1?1?1?1?1 hash.txt
where the *m* flag defines the hash type (SHA1), *a* represents the attack mode (brute-force), *1* represents the 76-character password alphabet, and the salt is appended to the beginning of the password. *Hash.txt* is the name of the file where I saved the hashed value (without the salt). This command effectively tells hashcat to look for an 8-character password for which the first two characters are known. Hashcat recovered the password after 30 minutes, by which time 12% of the keyspcae had been covered.

II. Reverse engineering apps

A. Problem #3

Recovered password: 4x@InWPM4DfB(cvW
Method for password recovery:
I started by running the program on its own, and then running it repeatedly with the IDA pro debugger. I set breakpoints to execute the program one instruction at a time and noticed how the register values changed accordingly. The program takes two steps in checking for the correct password: first, it compares the length of the input to the length of the correct password. If the length of the input is not equal to 16 charaters (10 in hexadecimal), the program outputs an *Access Denied* error message. If the length of the input is correct, the program starts comparing input characters to correct password characters one at a time.
Since the characters that are being compared are loaded into registers, my initial strategy for recovering the password was to run the program 16 times, recovering one correct character at a time, but I ended up finding the location in memory in which the password is saved. I did this by looking at the instructions preceeding the input-chracter to correct-character comparison, and looking at the memory contents in that area. I recognized some of the password characters that I had already identified because they were saved as plaintext, so recovering the rest of the password was straightforward.

Program patch that will accept any password:
The patch requires two changes corressponding to the check steps described above. The first change has to be made in case the input is not 16 characters long. In the original program, if the length check fails, the value *0* is saved into

register *edx* and the program jumps to a location in which *edx* is checked, and an error massage is displayed if *edx* has the value *0*. Therefore, this part can be patched by replacing the value *0* with some other value (I chose the value *1*).
Another change is required in case the input is 16 characters long and it does not match the correct password. In this case, the program also stores *0* into *edx*, so the patch changes the instruction to save a value other than *0* into *edx*. These two small changes guarantee that any input will be accepted as the correct password.
The diff file generated by IDA for this patch has been attached to this submission (filename 29093119.program1.dif)

B. Problem #4

Recovered password: #z3u#
Method for password recovery:
As in the previous problem, I analyzed this prohgram with the IDA debugger, and I was able to find the location of the correct password in memory. In this program, however, the correct password is not stored in plaintext. The IDA debugger shows that a subroutine named *SHA1* is called before the input and correct password comparison begins, so I assumed the correct password's SHA1 hash value is being stored in memory and that the program hashes the inputs before comparing to the password's hash value. I could not find the expected length of the correct password, so I used the hashcat tool to find the corerct password. I ran hashcat with a similar command as problem #2, but I used the increment flag to begin the search at 2 characters up to a maximum of 20 characters. I also added to the alphabet some special characters that were not part of the alphabet for problem #2. Hashcat found a match in 2.5 hours.

Program patch that will accept any password:
Unlike problem #3, this program does not check input length, but it has similar behaviour when a mismatch between the input hash and the password hash is detected; the value *0* is moved into a specific address as denoted by the follofwing assembly code:

```
mov [ebp + var_29], 0
```

The patch consists of writing another value into this address (I chose the value *1*).
The diff file generated by IDA for this patch has been attached to this submission (filename 29093119.program2.dif)

Program patch to replace the pasword expected by the program: