Lab 10

Chetan Sahrudhai Kimidi - ckimidi

***Abstract*—Patching and bypassing the password, of a CRACKME, using binary analysis on Ghidra**

I. Introduction

In this lab, I will crack a binary, crackme.bin, using Ghidra.

II. Ghidra

Ghidra is a free and open source reverse engineering tool developed by the National Security Agency of the United States. It works by decompiling the object code back to source code, for instance, in this case, from C language syntax to assembly language instructions such as MOV, JMP, ADD, JNZ etc. Basically, it is a disassembly tool, which facilitates such reverse engineering, without tampering the file system integrity.

III. Step-1: Finding the logic

After proper installation of Ghidra on the VM, I also downloaded the binary which I needed to analyze, from the given link. Then, I opened up Ghidra through command line, using the "./GhidraRun" command.

Following the startup and launch of Ghidra, I got familiar with the interface first. I understood that I had to load/open the binary file, to reverse engineer it and view the assembly code. I opened the crackme.bin on Ghidra, and then, in the left panels, I opened the Functions dropdown in the Symbol Tree. From that list of functions, I clicked on the "main" function, which led me to its respective C code, in the right view, and the assembly code, in the left view.

First, I carefully read and understood the whole logic of the main function. Then, I also ran the crackme.bin a couple times to test out some assumptions, which proved to be right. The basic structure/flow of the binary file when executed is such that, there is a prompt for a username first, which is limited to 8 letters in this lab, following which there is a prompt for a digit in the range of 1 to 9. Any digit smaller or larger than the respective range limits i.e. 1 and 9, are erroneous/invalid inputs. After a valid digit is given, then the password entry shows up finally. Upon correct password entry, the output is "you are successfully logged in", whereas upon wrong password entry, it shows "wrong password.".

What I observed and understood from the C code is that, there are respective data structures to store both the username and the input digit, and based upon that, the correct password is computed in such a way that, each letter of the password is equivalent to a letter obtained by adding the digit to each letter of the given username.

For instance, if abc was the username and 6 was the input digit, the correct password would be (a+6)(b+6)(c+6), which would be, ghi. I understood this from the operation of the indefinite while loop present in the actual else-if condition statement, of the given binary file. So, I wanted to make sure whether that is the actual logic behind the password computation, and did the tests, as seen in Fig. 1 and Fig. 2.

After finding the variables for the username and the digit, I was able to understand the whole flow of the main function and hence retrieve those two and compute the actual password. My proper C code for the binary's main function was as follows, after modifications and reasonable variable renaming:

My C code

```c
#include <stdio.h>
#include <string.h>
void main(void) {
    int NumInput, limit, InputLen;
    char Input[32], CorrectPWD[32], GivenPWD[24];
    printf("Enter Username: ");
    scanf("%s", Input);
    printf("\nType a number between 1 and 9: ");
    scanf("%d", & NumInput);
    if ((NumInput < 1)){
        printf("\nPLEASE Enter a number between 1
    and 9");}
    if ((NumInput >10)){
        printf("\nPLEASE Enter a number between 1
    and 9");
    }
    else
    {
        limit = 0;
        while (1) {
            InputLen = strlen(Input);
            if (InputLen<= limit) break;
            CorrectPWD [limit] = (char) NumInput +
    Input [limit];
            limit ++;
        }
        printf("\n Enter Password: ");
        scanf("%s", GivenPWD);
        int string_compare= strcmp(CorrectPWD,
    GivenPWD);
        if (string_compare != 0) {
            printf ("\nWrong password");
            CorrectPWD [limit] = '\0';
            printf("\nThe correct password is: ");
            printf("%s", CorrectPWD);
        } else {
            printf ("\nYou are successfully logged
    in");
        }
    }
}
```

IV. Step-2: Patching and bypassing the password

Now, the main task was to display a successful login, in-spite of a wrong password entry. So, I started by thinking what would control the output of a login. I understood the if-else conditioning/validation is the key to displaying the appropriate output. So, I checked the code for string comparisons, since to

validate the password, there needs to be some sort of checking like that. Then, I found the usage of strcmp function in one line of the code, following which, there was an if statement, stating that if the output is zero, then display the successful login, else do not. I thought that if I change this particular if condition, then irrespective of the comparison's result, the successful login attempt would be displayed.

So, I clicked on the if statement, viewed its equivalent assembly code, to find a JNZ (Jump if Not Zero) instruction, as seen in Fig. 3. I simply edited this using patch instruction feature of Ghidra and changed it to JZ (Jump if Zero), such that it displays a successful login for a wrong password.

After these changes, I tested the new crackme binary, by intentionally giving the wrong password, and it successfully displayed the "logged in" output, as seen in Fig. 4. Later, I thought that this would give rise to a new situation since, if the actually correct password is entered, then it would be shown as wrong password. so, we can use the unconditional jump instruction, JMP, to show the successful login, for every password.

Also, the code for the Bonus Task, to show the correct password, is included above in the code listing.

V. Conclusion

To conclude this lab exercise, I learnt about Ghidra interface, found out the logic behind the correct password computation, and finally patched and bypassed the password, such that the output is shown successful, even for wrong passwords.

References

[1] Ghidra
[2] x86 Instructions
[3] Code Block LaTeX Formatting

Figure 1. Wrong when random password given!

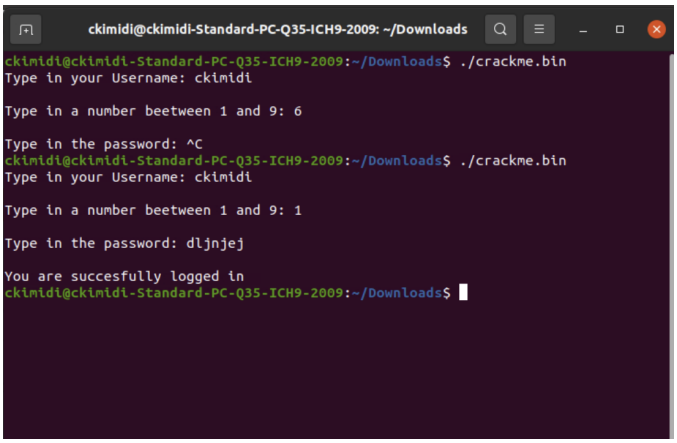

Figure 2. correct password with assumed C code logic

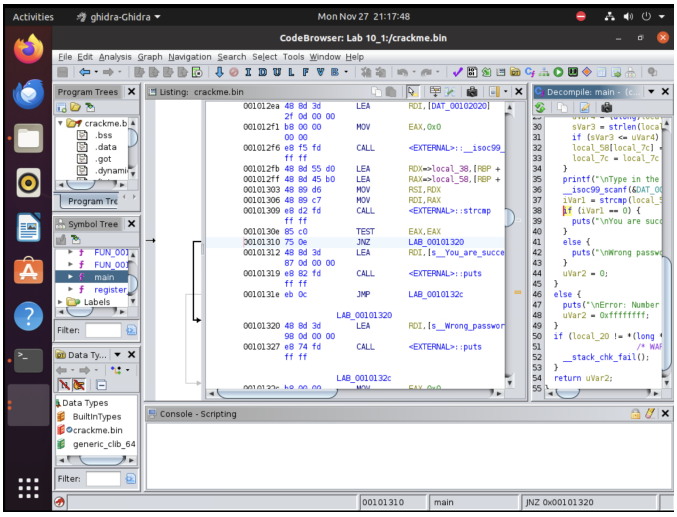

Figure 3. JNZ instruction found (highlighted in blue)

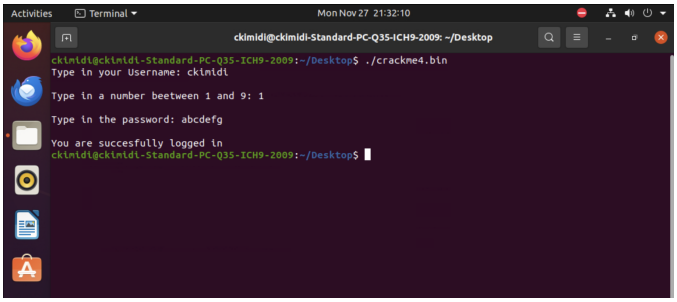

Figure 4. Patched and Bypassed!!!