Lab #3: crack them:

CYBR 570 : Reverse Engineering
Spring 2025

Submitted By: Lilly Carlascio
Submission Date: 4/23/25

good_job: h4x0r

1. Running strings on this binary gave me some intersting strings like 'Enter the passwrod:', 'Good job!', 'Bad job'. But I didnt see the names of any functions so I can 'nm good_job' and found out that the binary was stripped, and I wouldn't find any important symbols or that information.

2. I then opened the file in Ghidra and walked through each of the functions until I found what looked like the main program, and the one the had the puts statements and asking for the password and checking.

3. The program scans for user input then performs this input validation check:

```
if((local_118 == 0x30783468) && (local_114 == 0x72))
```

   This is comparing the input string (local_118) to the raw bytes, which can be flipped in little-endian to be shown as: 68 34 78 30. These can be represented as ASCII characters 'h', '4', 'x', '0'. And then the last byte (local_114) can be represented as 'r' in ASCII.

4. This spells out the password of 'h4x0r' which when ran with the binary gives the success message.

5. For this binary I wrote a c program called 'input_good_job.c' that creates a password of 6 characters since that is what I was able to decode from the assembly and decompiled code in ghidra. The value '0x30783468' is hardcoded in and used in memcpy which takes the little endian of that value of bytes and copies it into the allocated space of the password. The 'r' character is added at the end to complete the password as well as the null terminating character. The password is then printed out and it can be copied and used with the executed binary. (see ReadMe for execution instructions)

guess_the_password: 1abc9defgh

1. I first ran 'strings guess_the_password' and found these interesting strings: 'What is the password:', 'Correct', 'Incorrect'. I was also able to see important functions like main and check.

2. I then opened the binary in Ghidra to look for the functions. In the main function the program prompts the user for input of what the password would be and the passes the input to the check function.

3. In the check function, there are 3 input validation that the function looks for. It checks to see if the string is 10 characters long, the 0th character is a '1', and the 4th character is a '9'.

4. This shows that there can be many different inputs to make this binary output "Correct!" since there are only 3 simple requirements. Some examples of correct answers could be: '1abc9defgh' or '1aaa9zzzzz' or really any combination of characters as long as the 3 input criteria are met.

5. This regular expression shows all of the valid input: ∧1,{3}9,{5}$

6. I have added a input txt file with 5 different solution inputs and when running './guess_the_password ¡ valid_input.txt' It will output 'Correct'.

7. For this binary, I wrote a c file called 'input_guess_password.c' that basically hard-codes the 0th and 4th index of the password and randomly uses characters for the other remaining 8 characters in the 10 charcter password, also adding a null terminating character at the end. Each time the program is run a new password is printed, that can then be copied and pasted when the guess_password binary is executed. (see ReadMe for execution instructions)

task4: Kj]UeETf

1. I first ran 'strings task4' to find any interesting strings. The results from that command gave me strings like "Enter your serial", "Access Granted! Welcome!", "Access Denied! Invalid Serial.". It also showed me some function names like main, check_serial, and transform.

2. I then opened the binary in Ghidra to look for the functions. Looking at the main function, it scans for the user input and then passes the scanned string into the check_serial function.

3. Going into the check_serial function, it loads values into an array. Converting the values into ASCII, the target array reads out: 'Y', 'o', 'a', 'l', '', 'V', 'f', 'u'. The function then checks to make sure the input from the user is exactly 8 characters, it then transforms each of the input characters to match the target array as shown above.

4. This happens in the transform function which adds 7 to the character's index, XOR's the character with that result, adds 13, and then keeps the lower 7 bits, which is shown by this code line:

```
int transform(char c, int index) {
    return ((c ^ (index + 7)) + 13) & 0x7F;
}
```

5. From there I had to write a function to reverse the transform, to find out what input would satisfy the condition and match the serial (revserse_transform.c). This program hard-codes the target ASCII values that are being transformed as found from Ghidra and then does a reverse of the transform and returns the serial. After running the program, it gave me the output of 'Kj]UeETf', which gave me the success message. (see ReadMe for execution instructions)