HW2

**Problem 1** (15 Points): Hash functions: In class we discussed several desirable properties for hash functions, in particular one-wayness and collision-resistance. In this exercise, we’ll show that neither property implies the other. We can do this by counter-example:

1. a) First, define a function that is one-way, but not collision- resistant.

**Hint:** The answer will be a trivial function that you would never use as a cryptographic hash function.

F (x)= x %2

It is one-way. Because given the output, it is difficult to find the input that produced it, but easy to find a collision input.

1. b) Second, define a function that is collision-resistant, but not one-way.

**Hint:** Assume you have a collision-resistant hash-function H. Use that to build a hash function H’, which is still collision- resistant but not one-way.

H’(x) = H(x) concate x.

We can get the input at the end of output. So it is not one way.

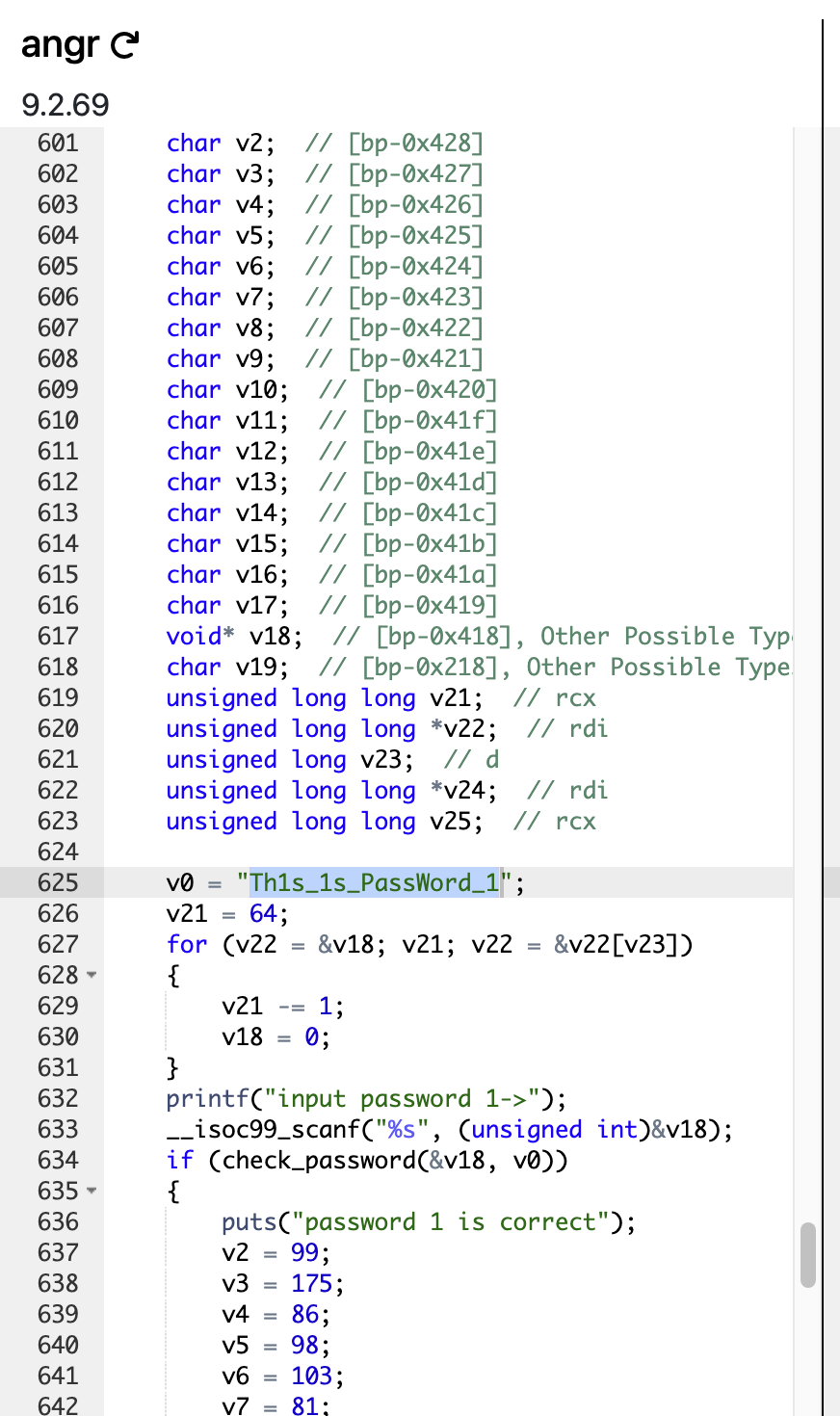
Because H(x) is collision resistant, so H’(x) is still collision-resistant.

**Problem 2** (15 Points): Consider the following C code. What is the problem? Explain.

int a, b;  
int sum=a+b;  
cout<<"Enter two numbers to add: ";  
cin>>a;  
cin>>b;  
cout<<"The sum is: "<<sum;

1. Sign mismatch. Two int-type number additions may be out of the int range. It can cause overflow. For example, INT\_MAX + 1 = INT\_MIN
2. Input happens after the sum calculation. So, even if the user enters values, they won't be used in the **sum** calculation because it occurred before the input.
3. if you read a string using **cin** into a fixed-size character array and the input is longer than int length, it can overwrite adjacent memory, leading to undefined behavior and potential security vulnerabilities. To mitigate this, you should use functions like **getline** or consider dynamic memory allocation for input that may vary in size.
4. **Type Mismatch**: **cin** expects input to match the data type of the variable it's reading into. If the user enters data of the wrong type, it can lead to unexpected behavior or errors in the program. For instance, if you're expecting an integer, but the user enters a string, it may result in input failure and possible security risks if the program doesn't handle this gracefully.

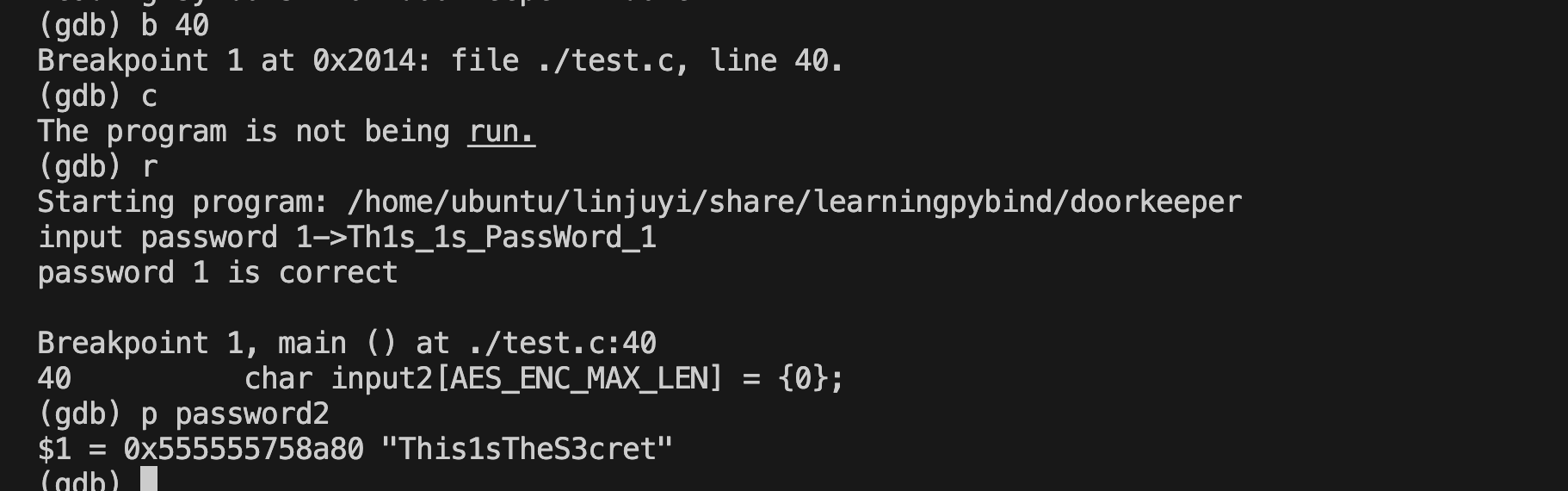
**Problem 3** (30 Points): Simple Reverse Engineering Course. There’s a **Linux** binary file called **doorkeeper**, which has two password checks. Try to crack the two passwords. Show your cracking progress by screenshot step by step. To simplify the process, the debug information is reserved, and the optimization is disabled.

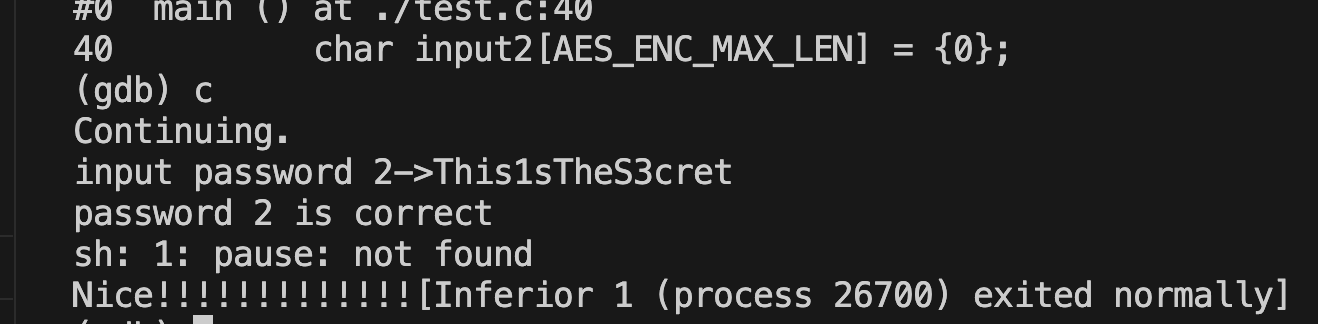


We can find first password is Th1s\_1s\_PassWord\_1

We should read

char\* password2



So the second password is This1sTheS3cret

Question4

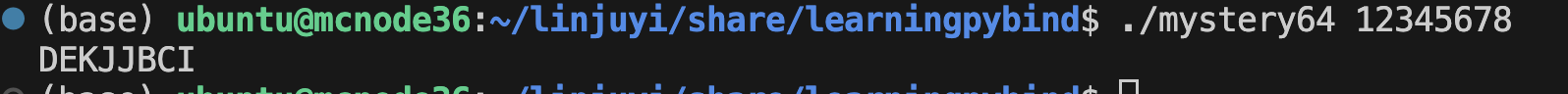
**Main course:** You are a 1337 H4x0r, after breaching the security perimeter of a high profile target you were able to compromise one of their devices, find some **encrypted** passwords and exfiltrate the **password generation binary**.

First, run the provided **mystery** binary (./mystery32 or ./macMystery) to find the inputs that will produce the following encrypted passwords: (15 points)

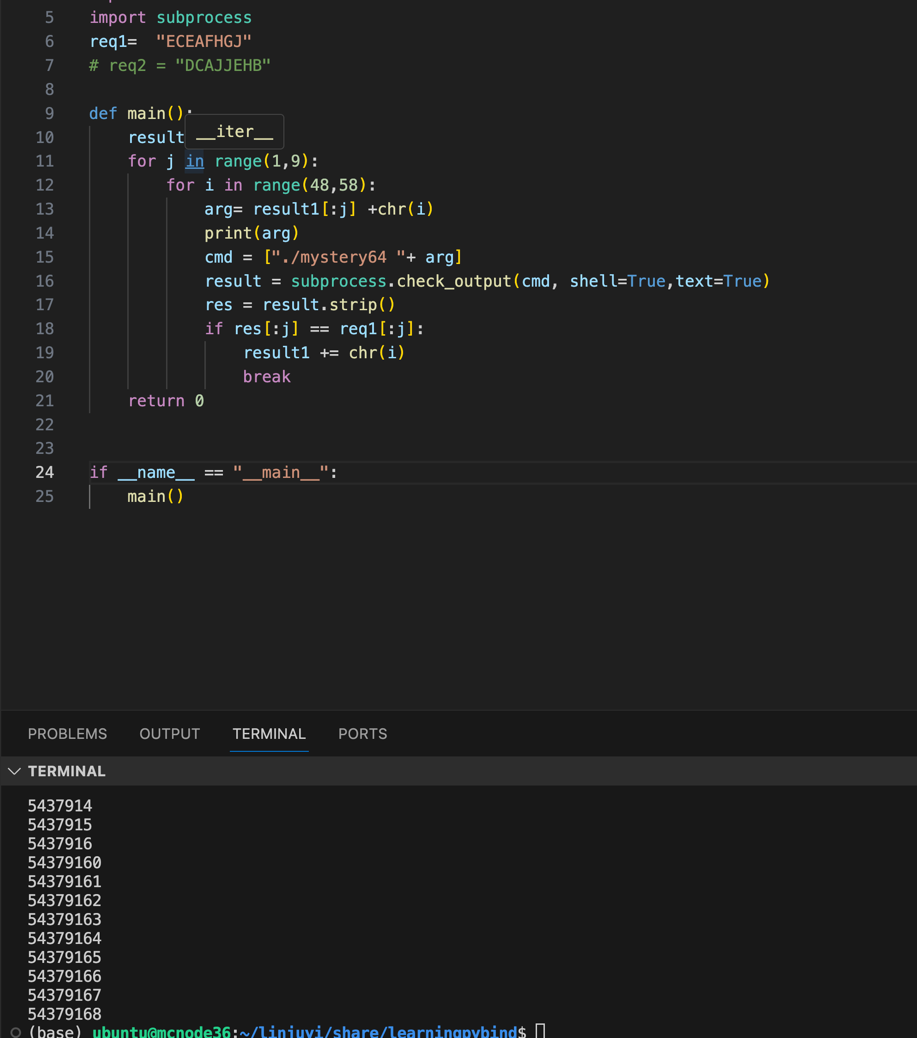
We can not brute force 0-99999999 all numbers. It takes too much time.

i. ECEAFHGJ 54379168

ii. DCAJJEHB 15935728

iii. DEKJJBCI 12345678

**Hint:** Time, automating the process will save

Write a program, if a[0] == E, then judge a[0]a[1] == EC then judge a[0]- 3 == ECE

# write a program, iterate all numbers from 0 to 99999999.

# For example , we run ./mystery64 0, then ./mystery64 1

import subprocess

# req1= "ECEAFHGJ"

req1 = "DCAJJEHB"

def main():

result1= ""

for j in range(1,9):

for i in range(48,58):

arg= result1[:j] +chr(i)

print(arg)

cmd = ["./mystery64 "+ arg]

result = subprocess.check\_output(cmd, shell=True,text=True)

res = result.strip()

if res[:j] == req1[:j]:

result1 += chr(i)

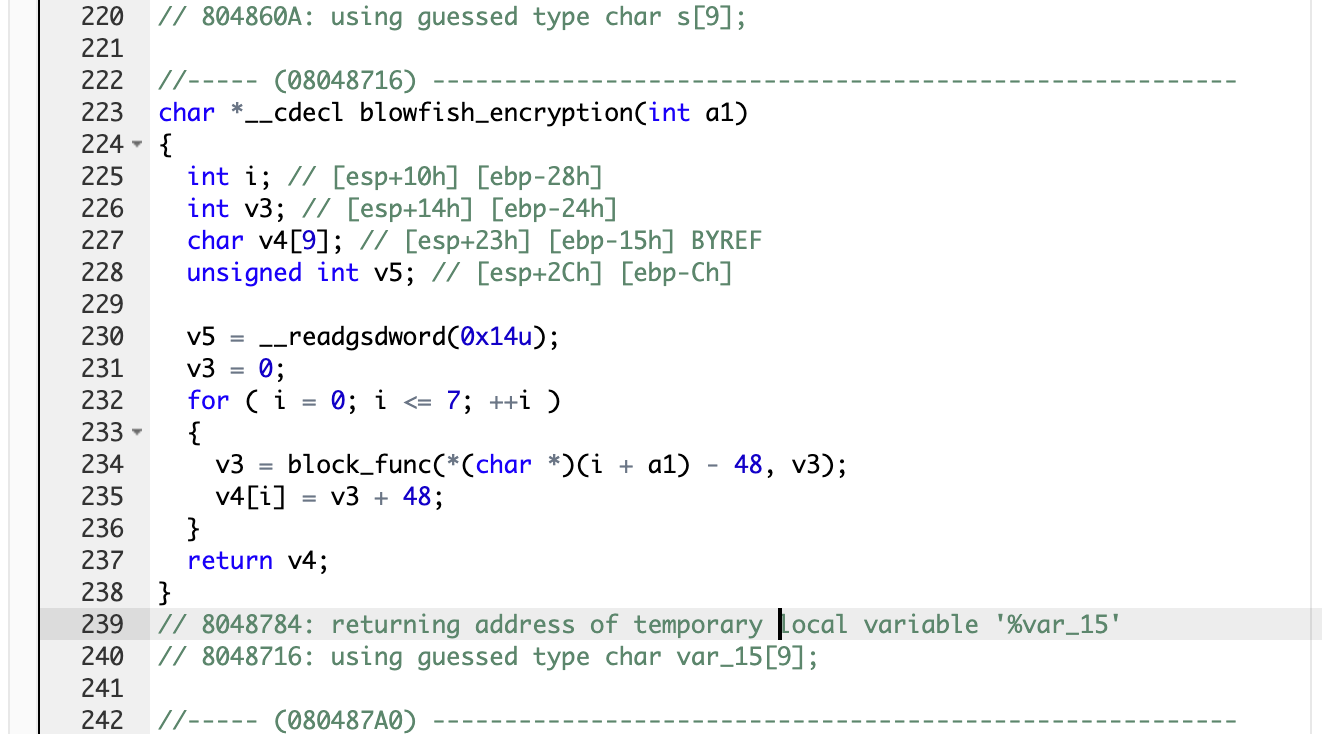
break

return 0

if \_\_name\_\_ == "\_\_main\_\_":

main()

1. b)  Second, **decompile** the provided binary and **identify the functions** (e.g., names, input-output parameters, etc.) used for the encrypted password generation. (25 points)  
   **Hint:** For function symbols, you shall look



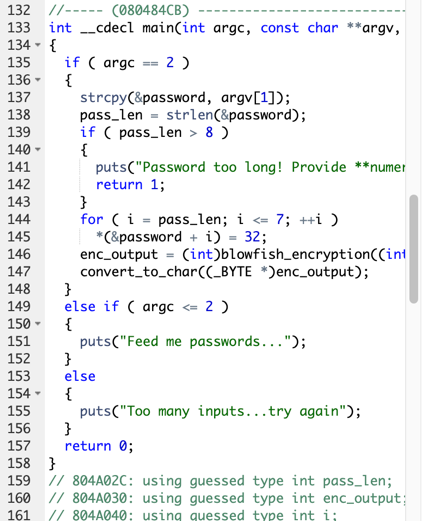
names: blowfish\_encryption

input-output parameters

input: int

output: char\*

1. c)  Third, identify the steps of the password generation algorithm. Provide clear explanations on how the input is transformed to the output in a step-by-step fashion. (40 points)  
   **Hint:** A chain, everything is



Input is a numeric password of up to 8 digits.

Then, we do a padding.

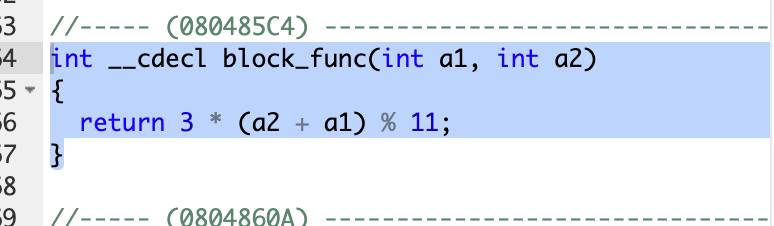
for ( i = pass\_len; i <= 7; ++i )

\*(&password + i) = 32;

means if length <=8, append space to 8 length.

Then, we use blowfish\_encryption. From decompiling, we can get some information. See question b

It invokes block\_func



First it calculate the first number, then calculate the second number.

For example, We input 12345678, get

./mystery64 12345678

DEKJJBCI

V3 = blockfunc( 1, 0 )

V3 = 3

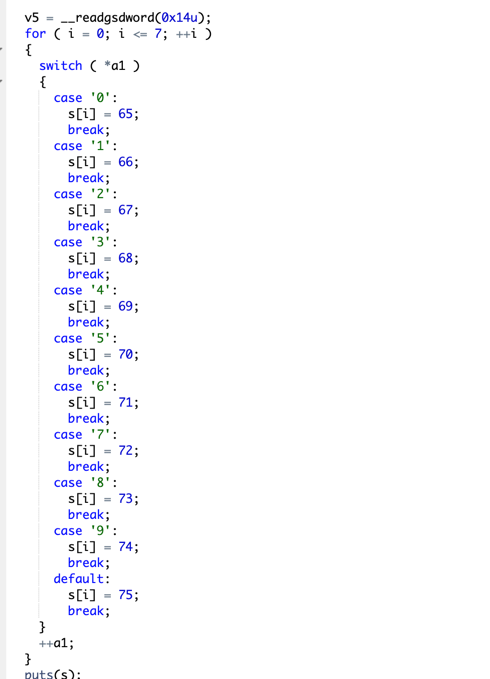
V4[0] = 3

V3 = blockfunc( 2, 3 )

V3 = 4

V4[1] = 4

Finally, it convert number to character.

so v4[0] = 3 - >E , V4[1] = 4 ->E