LEVEL NEW ORLEANS:

In this level, the function check_password compares the user's input byte by byte with the password created by create_password and stored at memory address 0x2400. The function uses relative addressing to iterate through the stored bytes. Here's how it works:

Let's assume that r15 is 0x4000.
Repeat until r14 is 0x8:

Iteration 1: r14 = 0x0. r13 = r15 + r14 (0x4000 + 0x0). It checks if the byte at address 0x4000 is equal to the byte at address 0x2400+0. If they match, r14 += 0x1, and it goes to Iteration 2.
Iteration 2: r14 = 0x1. r13 = 0x4000 + 0x1. It checks if the byte at address 0x4001 is equal to the byte at address 0x2400+1. If they match, r14 += 0x1, and it goes to Iteration 3.
...
Notice that 0x2400+1 is equivalent to 1 (0x2400) due to relative addressing.

The solution is simple: We need to provide input that exactly matches the 7 bytes (since the null terminating byte is implicit with gets) that create_password stores in memory.

The password is a sequence of 7 bytes: 0x66, 0x69, 0x73, 0x35, 0x61, 0x6b, and 0x29.

So, to unlock the level, you need to input this password exactly. You can use hexadecimal encoding for the input:

Hexadecimal: 66697335616b29

LEVEL SYDNEY:

In this level, we have two main functions: main and check_password. The main function takes user input and calls the check_password function to validate it.

The check_password function checks whether the input provided by the user matches a predefined password. It performs four comparisons on the user's input using relative addressing. The password is stored in little-endian format.

To solve the level:

1). Find the little-endian representation of the password: 0x2e78, 0x673d, 0x333d, 0x5e63.
2). Convert each pair of bytes to little-endian format: 782e, 3d67, 3d33, 635e.
2). Input the converted password in hexadecimal format (without spaces) to pass the level.

Now the machine reads this as Little-Endian, so you need to reverse the byte order to match the representation used in memory. Once you input the correct password, you will successfully solve the level (782e3d673d33635e).

LEVEL HANOI:

In this level, the main function calls the login function, which asks for a password input. The input is accepted using a function called getsn, which allows up to 28 characters to be stored in memory from address 0x2400 to 0x241B.

After inputting the password, the program calls the test_password_valid function, which takes the password as an argument and checks if it's valid. The test_password_valid function uses an interrupt (INT) with code 0x7D to validate the password and places a value at memory address 0x2410 based on the result.

The goal is to unlock the level, which requires having the value 0x91 at memory address 0x2410. To achieve this, we need to input 16 random characters (hex-encoded) followed by 0x91. The program will ignore the rest of the characters after the 0x91, and as long as the test_password_valid function sets r15 to zero, the level will be unlocked.

So, the solution is to input: 41414141414141414141414141414191

This input places 0x91 at memory address 0x2410, satisfying the condition for unlocking the level.

LEVEL CUSCO:

In this level, we encounter a buffer overflow vulnerability. The login function takes user input and stores it in a buffer of size 48 bytes. However, there is no bounds checking, and it is possible to write more data than the buffer can hold, leading to overwriting adjacent memory.

The vulnerability arises because the stack grows toward lower addresses, and memory writes happen toward higher addresses. This means that when a function is called, its return address is saved on the stack "above" the function's local variables and buffers. If we can overwrite the return address with a target address of our choice, we can manipulate the program's flow.

To exploit the vulnerability and solve the level:

Find the number of padding bytes needed to reach the return address of the login function. In this case, it's 16 bytes.
Determine the target address you want to jump to, which is the address of the unlock_door function. In this case, it's 0x4528.
Reverse the byte order of the target address to match little-endian format, resulting in 0x2845.
Create an input that consists of 16 padding bytes (e.g., "A" repeated 16 times) followed by the little-endian representation of the target address (0x2845).
So, the solving input will be (hex encoded): 41414141414141414141414141412845