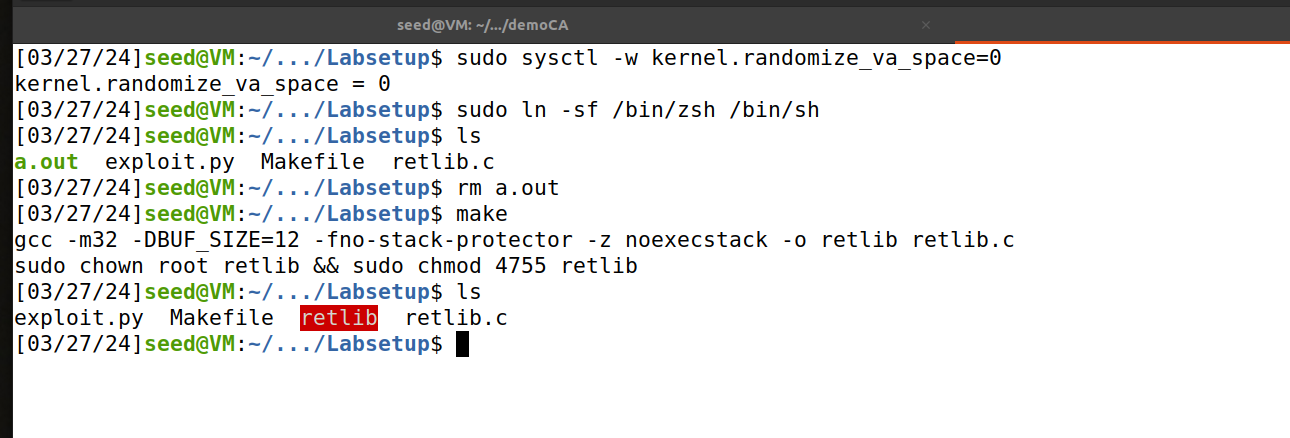
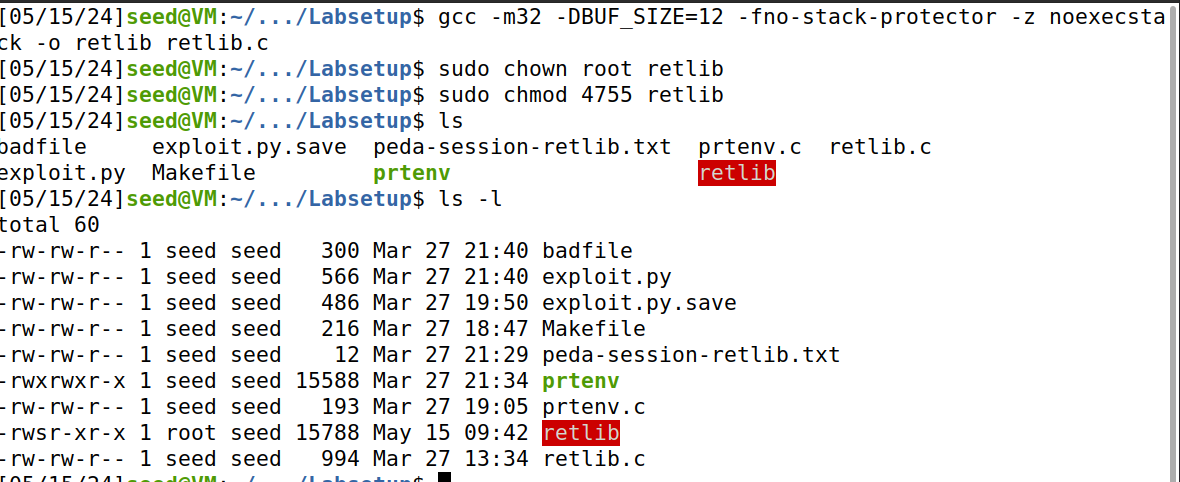
**Name:** Henry Salgado

ID: 80509684

**Return-to-libc Attack Class Assignment**

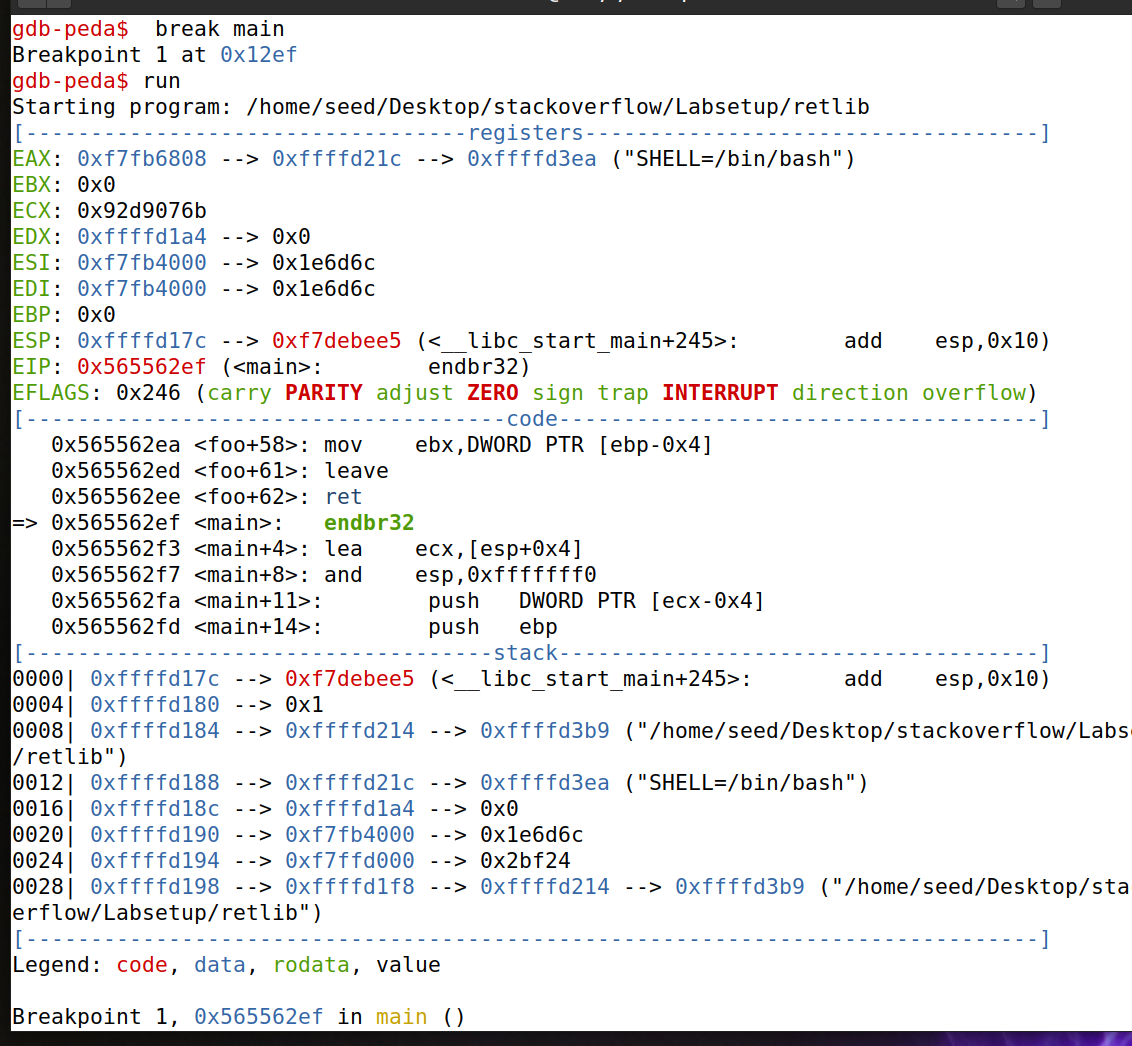
**Step 1: Turning off countermeasures, configuring /bin/sh and compiling code (after turning it to a set uid by chmod 4755 command)**

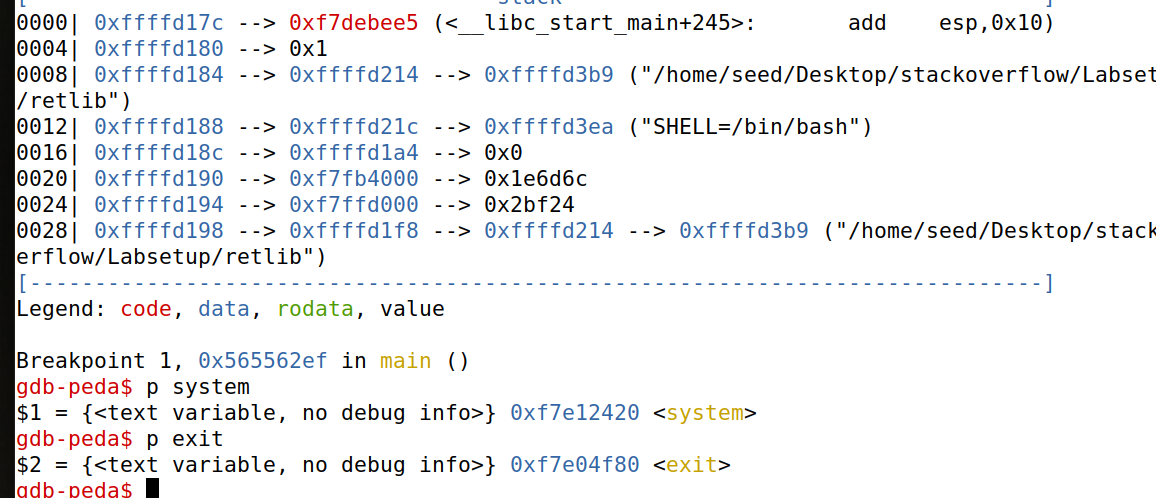




**Observation:** We can see retlib now has “S”, therefore a set-uid program.

**Task 1: Running the dbg and obtaining addresses**



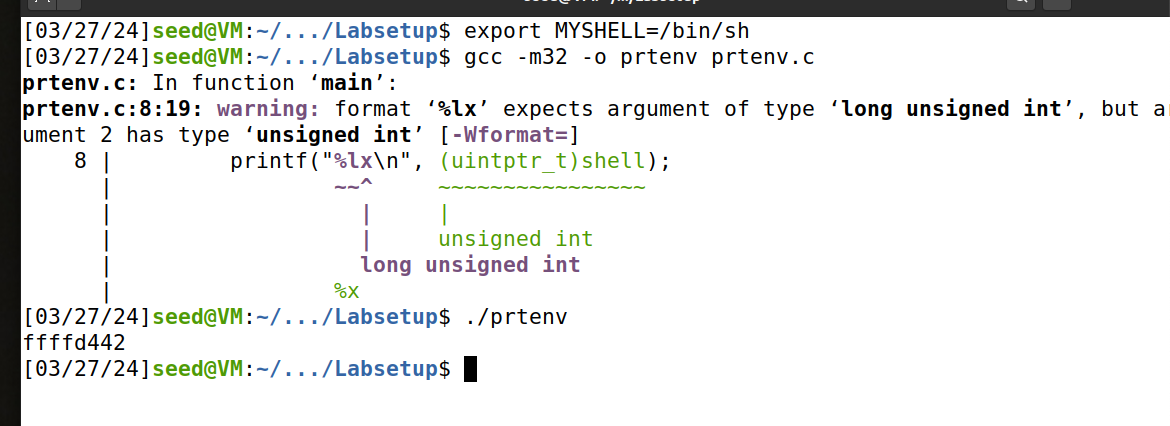


**Why system() and exit() are Needed:**

* system() function: This function executes a command specified in a string by invoking the system's command processor (e.g., a shell). By redirecting execution to system(), we can execute arbitrary commands, like spawning a shell.
* exit() function: After executing the desired command with system(), it's important to ensure that the program terminates cleanly. Invoking exit() helps to gracefully exit the program, avoiding crashes or other undefined behavior that might alert defenders or interfere with the attack's success.

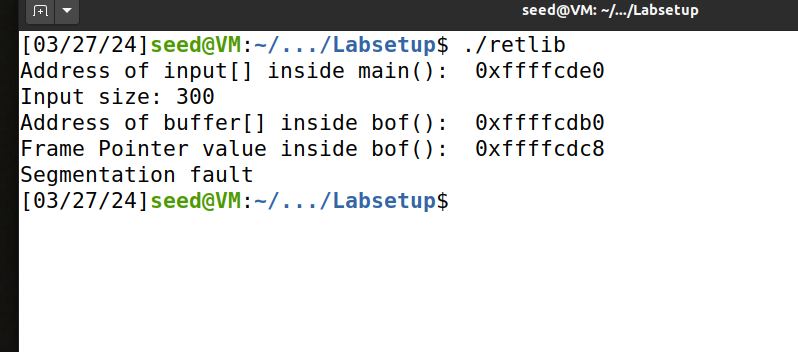
**Task 2: Putting the shell string in the memory**

Step: Setting Environment Variable and getting the address of Environment Variable



**Task 3: Launching the Attack**

**First failed attempt (by adding 4 to x, y, z)**



**Observation:** I get a segmentation fault.

**Successful attempt and Explanation:**

**First, I calculate the offset:**

0xffffcdb0 – 0xffffcdc8 = 24

The distance between %ebp and buffer is 24 bytes.

Therefore, if we try values, by adding 4 to x,y,z we can try to spawn a shell



**Success!**

**Values for success Z= 24+4, Y = 24+8 , X = 24+12**

1. **Z = 24 + 4**:
   * The value for Z (the address of **"/bin/sh"**) is placed closest to the return address in the stack because it needs to be the argument for the **system()** call.
   * Placing it 4 bytes after the base offset (28 bytes from the buffer start) aligns it to be the first argument after the return address.
2. **Y = 24 + 8**:
   * The value for Y (the address of **exit()**) is placed slightly further down the stack. After **system()** is called, the next function to execute (if needed) is typically **exit()**.
   * Placing it 8 bytes after the base offset (32 bytes from the buffer start) ensures it is in the correct position to be called after **system()** finishes.
3. **X = 24 + 12**:
   * The value for X (the address of **system()**) is placed furthest down the stack. When the buffer overflow occurs, the overwritten return address should jump to **system()**.
   * Placing it 12 bytes after the base offset (36 bytes from the buffer start) ensures it replaces the intended return address of the overflowed function.

**Attack Variation 1: Omitting the exit() Function**



Is exit necessary?

* Including exit() in the payload avoids this issue by providing a valid return path, ensuring the attack completes successfully.

**Observation:** Without exit, the attack yields a segmentation fault. The failure of the attack without the exit() function could be due to the way the stack and control flow are handled after the system() call.

**Possible explanation:** When the system() function is called, it spawns a new process to execute the given command (in this case, /bin/sh). After the command executes, the control needs to return to the calling function. Without a proper return address or a cleanup function like exit(), the program's stack might become corrupted, leading to undefined behavior and potential segmentation faults.

**Attack Variation 2:**



**Observation:** Again, the attack was not successful and yielded a segmentation dump.

**Possible explanation:** Changing the program name can change the memory addresses of environment variables due to the different lengths of the name. Since the attack relies on specific memory addresses, any change in the address of /bin/sh might cause the attack to fail.