Heap0

Moving on to the second day of tasks, I completed the heap0 challenge in the PicoCTF series, as it involves heap-based memory behavior.
The first step I took to start the challenge was to launch the challenge instance and establish a connection using nc from Kali Linux.
After successfully connecting, the program proceeded to print out the current state of the heap. From reading this output, I understood that two variables had been allocated on the heap: input_data, which I had access to, and safe_var, which I was not meant to modify according to the program's logic.
I attempted to overflow the buffer by writing to input_data. Initially, I tried to overflow the buffer by writing around 10 characters. This attempt was unsuccessful, and everything appeared to function normally, as if no overflow had occurred.
Later on, after printing the memory addresses of different heap locations, I realized that even though the buffer was defined as being five bytes in size, the heap allocator had rounded the allocation up to a larger size in memory.
Because the address difference between input_data and safe_var was much larger than expected, I realized I would need a significantly larger input size to fill the heap chunk. With this realization, I increased my input to around 40 characters, which successfully caused an overflow and overwrote safe_var.
After performing the safe_var manipulation, I selected the option to print the flag. Since safe_var was no longer holding the value the program expected, the program printed out the flag.
This challenge proved to be very important because it demonstrated that overflows are not limited to the stack. Additionally, when dealing with vulnerabilities on the heap, understanding heap allocation behavior is extremely important.