

Cache Me Outside

Description

While being super relevant with my meme references, I wrote a program to see how much you understand heap allocations. `nc mercury.picoctf.net 17612`

Attempts

Attempt 1: Decompiling using Ghidra

Since the compiled file is provided (`heapedit`), we can decompile it and have a look at the code.

We see that the flag is loaded from a file on the server (`flag.txt`) and then read into a variable (`char flag [72]`)

We can also see that the size of the flag string (`char[]`) is 72, which is 9bytes, however, only 8 bytes (`0x40 = 64`) are being read from the file (`fgets(flag, 0x40, flagfile)`). Maybe this is important, maybe not, who knows (I don't).

Then a huge bunch of pointer-arithmetic-shit is being done in a weird-ass loop, we won't bother with this one for now.

We can then also see `strcat((char *)local_98,flag)`

man strcat `strcat(char *restrict dst, const char *restrict src)`

appends the content of `src` to the content of `dst`, basically in-place concatenating the `dst`-string with the `src`-string.

Now, the thing is, this stuff happens every loop iteration, we can see in the loop that this happens exactly 7 times. So the contents of `local_98` now also contain our flag. Since `local_98` gets assigned a new address space in the heap (first line in the for-loop `local_98 = (undefined8*) malloc(0x80)`) I am assuming that we technically only care about the contents of `local_98` on the very last iteration, since this is then kept after the loop. We can also see a bunch of weird-ass addresses being written to this variable, don't know what to do with this info tho.

On line 48 we have a `free(local_98)`, which iirc only marks this address-space as free, but the contents of it are still very much there, since nothing else seems to be directly overriding it nor did the contents of `local_98` get moved/copied to any other variable I assume that we could still read the flag from that address-space.

free()

The `free()` function frees the memory space pointed to by `ptr`, which must have been returned by a previous call to `malloc()` or related functions. Otherwise, or if `ptr` has already been freed, undefined behavior occurs. If `ptr` is `NULL`, no operation is performed.

```
Decompile: main - (heapedit)
1
2 undefined8 main(void)
3
4 {
5     long in_FS_OFFSET;
6     undefined local_a9;
7     int local_a8;
8     int local_a4;
9     undefined8 *local_a0;
10    undefined8 *local_98;
11    FILE *flagfile;
12    undefined8 *local_88;
13    void *local_80;
14    undefined8 local_78;
15    undefined8 local_70;
16    undefined8 local_68;
17    undefined local_60;
18    char flag [72];
19    long local_10;
20
21    local_10 = *(long *) (in_FS_OFFSET + 0x28);
22    setbuf(stdout, (char *) 0x0);
23    flagfile = fopen("flag.txt", "r");
24    fgets(flag, 0x40, flagfile);
25    local_78 = 0x2073692073696874;
26    local_70 = 0x6d6f646e61722061;
27    local_68 = 0x2e676e6972747320;
28    local_60 = 0;
29    local_a0 = (undefined8 *) 0x0;
30    for (local_a4 = 0; local_a4 < 7; local_a4 = local_a4 + 1) {
31        local_98 = (undefined8 *) malloc(0x80);
32        if (local_a0 == (undefined8 *) 0x0) {
33            local_a0 = local_98;
34        }
35        *local_98 = 0x73746172676e6f43;
36        local_98[1] = 0x662072756f592021;
37        local_98[2] = 0x203a73692067616c;
38        *(undefined *) (local_98 + 3) = 0;
39        strcat((char *) local_98, flag);
40    }
41    local_88 = (undefined8 *) malloc(0x80);
42    *local_88 = 0x5420217972726f53;
43    local_88[1] = 0x276e6f7720736968;
44    local_88[2] = 0x7920706c65682074;
45    *(undefined4 *) (local_88 + 3) = 0x203a756f;
46    *(undefined *) ((long) local_88 + 0x1c) = 0;
47    strcat((char *) local_88, (char *) &local_78);
48    free(local_98);
49    free(local_88);
50    local_a8 = 0;
51    local_a9 = 0;
```

Figure 1: decompiled code

More Coming Soon

Binary Exploitation is great and all but this is too much guesswork for me, starting now - my bachelor's degree in Binary Exploitation with pwn.com as recommended by this roadmap

