UGA

Grenoble INP

M2 CySEC - Software Security

Lab Session 2

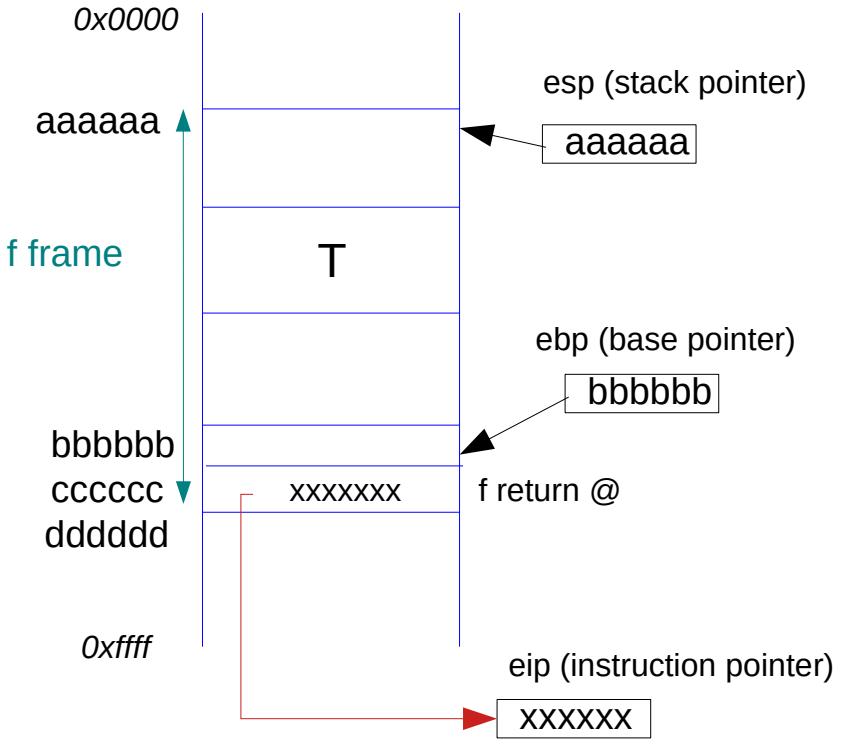
Cooking a simple

buffer overflow (BoF) exploit

Part 1

classical BoF attack

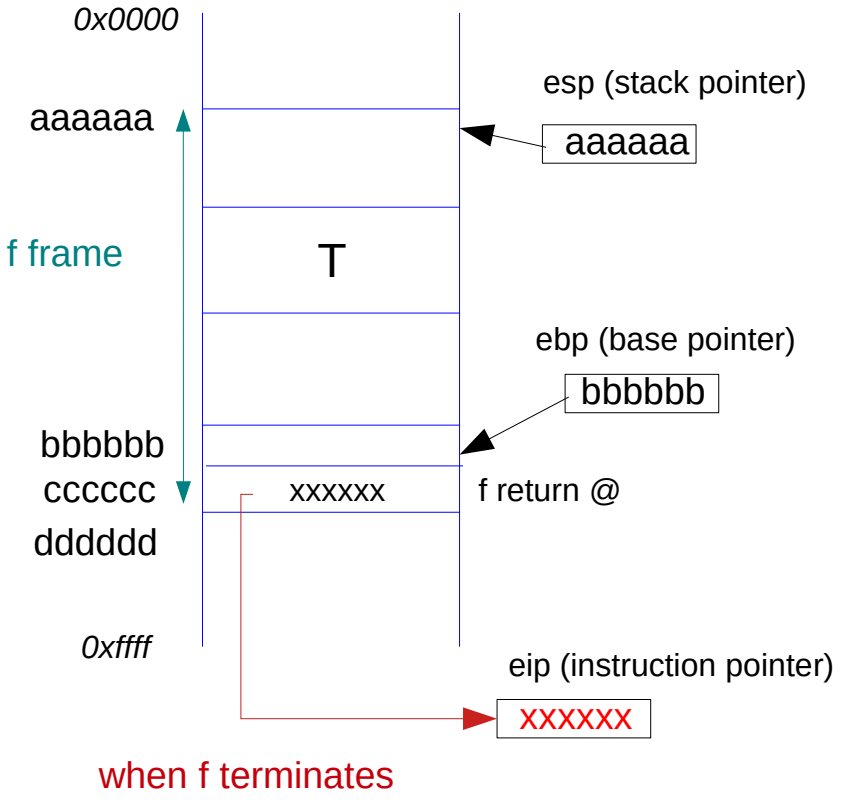
A vulnerable function f and its corresponding stack frame :




```
void f (…..) {

    char T[N]
        ….

     // T is filled with user-controlled input

            ….
}
```

when f terminates : execution resumes on the f caller's code at address xxxxxx

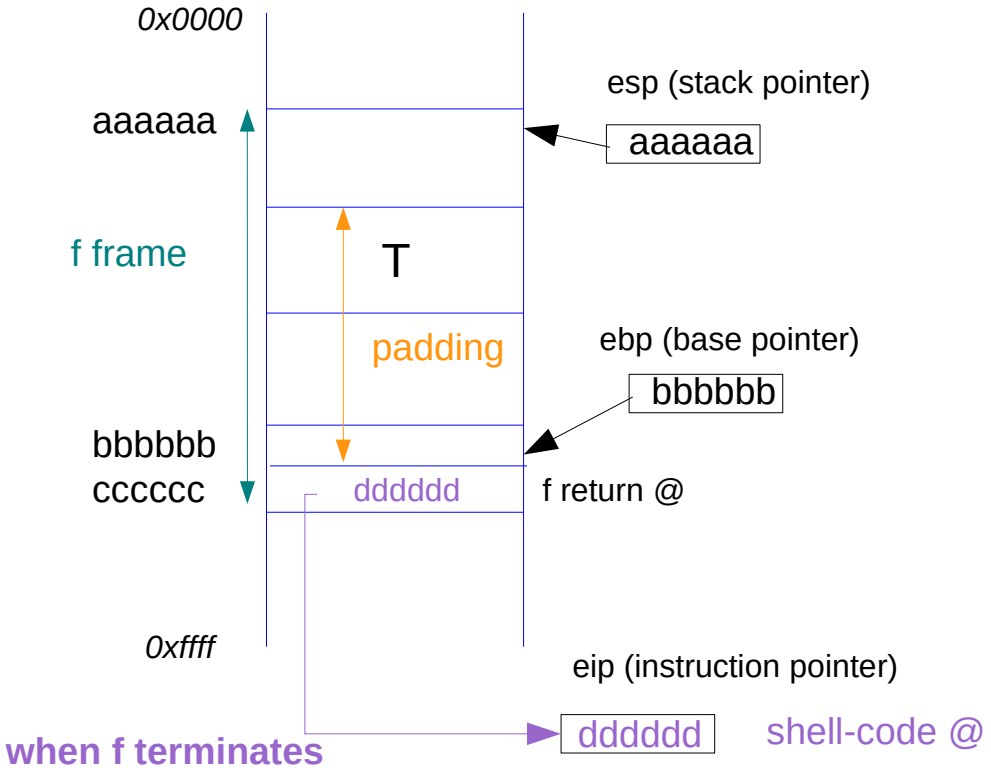
Rewriting f return address to hijack the control flow :




```
void f (…..) {

    char T[N]
        ….

     // T is filled with user-controlled input

            ….
}
```

Attacker goal : rewrite address xxxxxx with a **controlled shellcode address** in order to **hijack** the normal pgm execution ...

Building the payload (part 1)

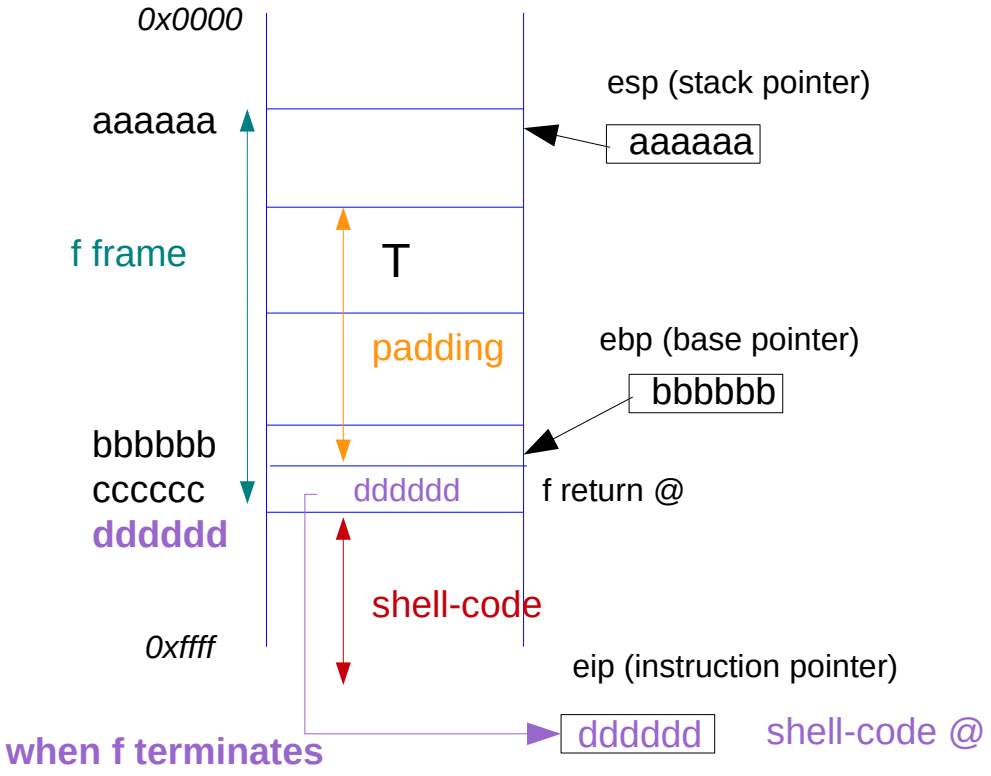



```
void f (…..) {

   char T[N]
   ….


}
```

T should be filled with the following input : <padding> <dddddd>

… the attacker needs to know the padding size … and find a shellcode address dddddd

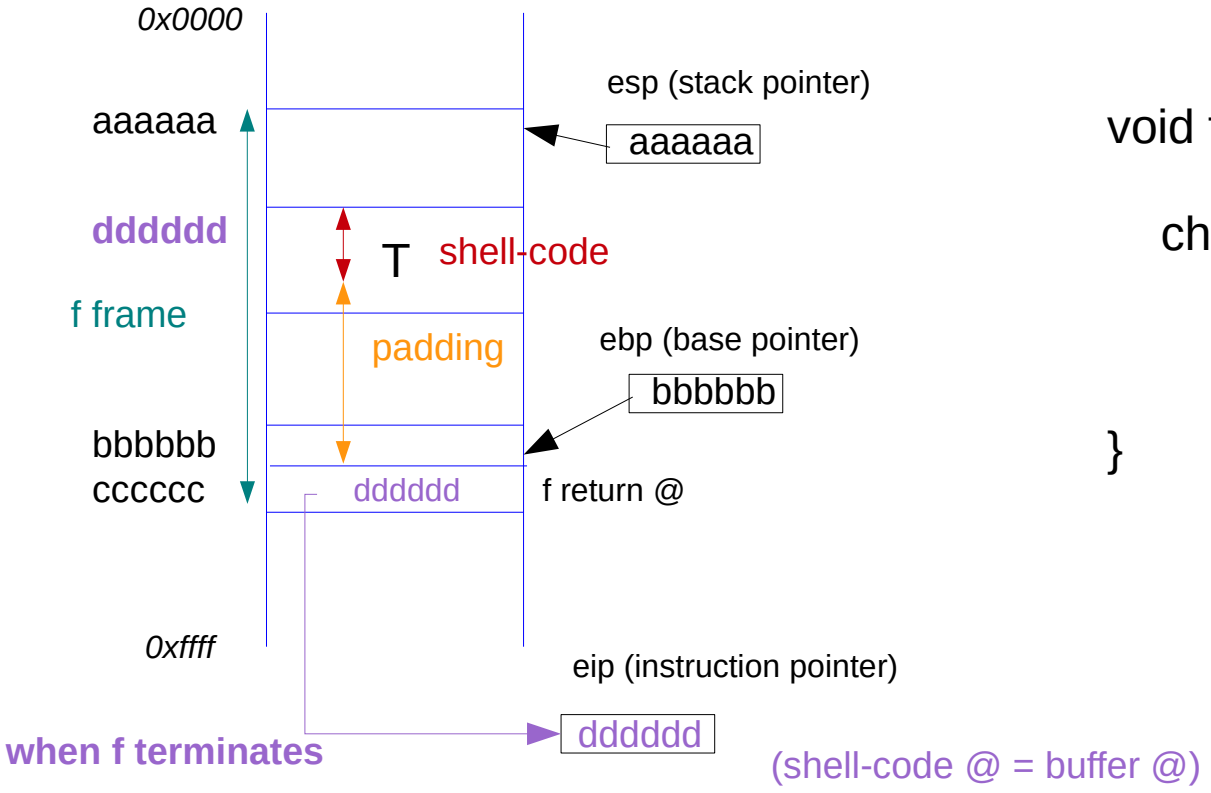
Solution 1 : put the shellcode below the return @ (in the caller's frame)




```
void f (…..) {

   char T[N]
   ….


}
```

T should be filled with the following input : <padding> <dddddd> <shellcode>

Solution 2 : put the shellcode inside the target buffer (if large enough)



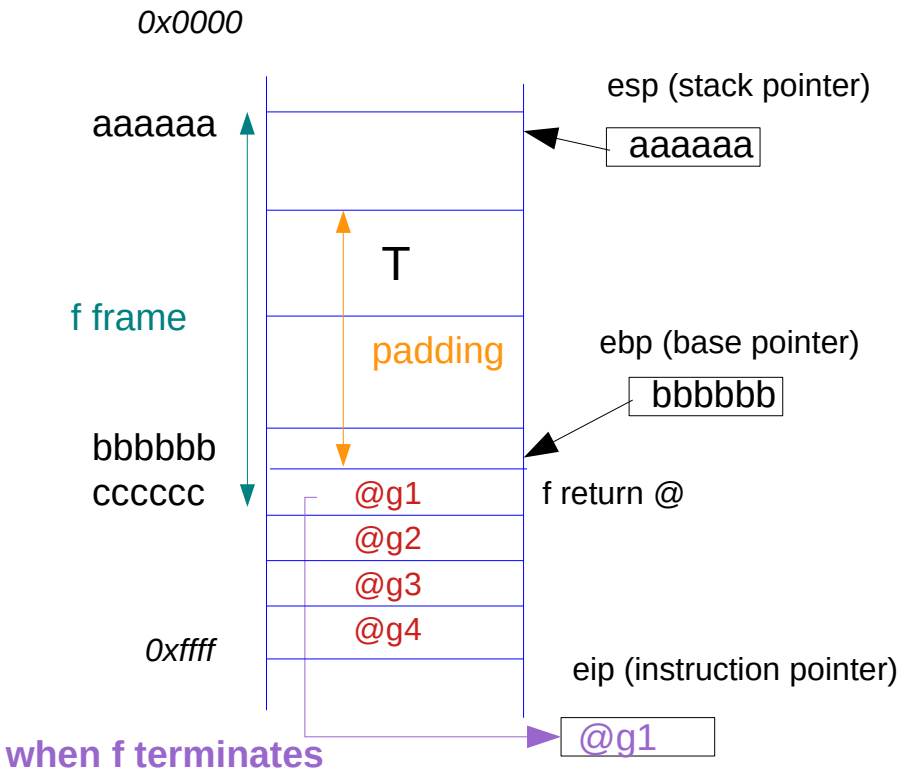

```
void f (.....) {

   char T[N]
   ....


}
```

T should be filled with the following input : <shellcode> <padding> <dddddd>

Part 2

ROP attack

Encode the shell-code as a sequence of ROP gadgets ...




```
void f (.....) {

   char T[N]
   ....


}
```

T should be filled with the following input : <padding> <@g1><@g2><@g3><@g4> ...

Gadget execution :

A gadget is :
- either a ret-terminated instruction sequence of the target program
- or some data to be processed from the stack by the following gadget

Example :

To put 42 into rax, 0 into rbx and call syscall function

Gadget sequence :

g1 : pop rax ; ret
g2 : 42
g3 : xor rbx, rbx ; ret
g4 : syscall

Stack content :