**Esercizi BufferOverflow**

**Write a program that prints the addresses of the parameters of a function called by main. Check that, if you have address space randomization on they are all different. Check the pass by register optimization.**

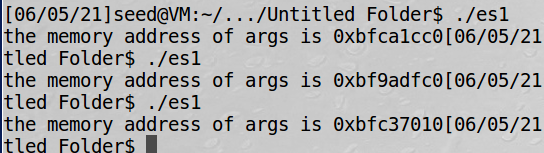
**#include<stdio.h>**

**int main (int args, char \*\* argv){**

**printf("the memory address of args is %p", (void\*) &args);**

**return 0;**

**}**

****

**we can observe the address is different each time we run the dumb program.**

**we use %p , for print address of some variable. We need & in front the variable for print.**

**Some dettail : \*variable = pointer of the variable (puntatore di una variabile );**

**\*\* variable = pointer of the Pointer of the variable.**

**ebp = frame pointer**

**esp = stack pointer**

**Enter with debug :**

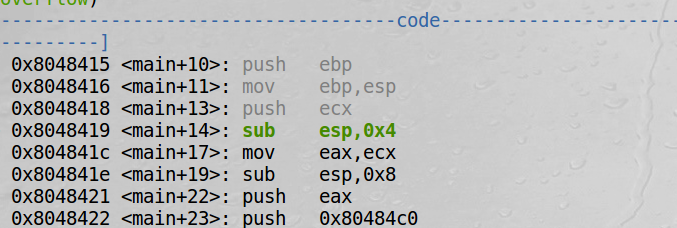
**gdb es1**

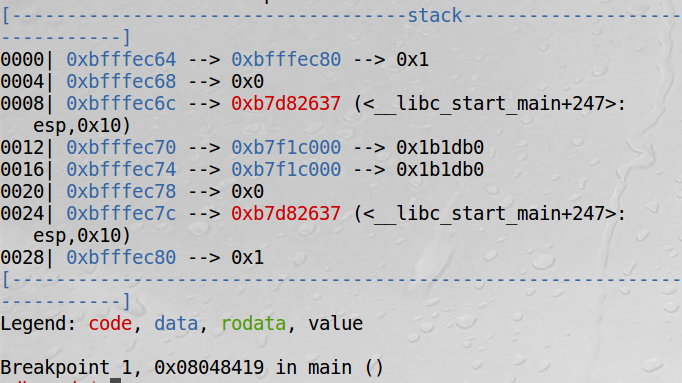
**(we are inside debug):**

**b main (=set a break point pointer on the function ,, so we can check it )**

**run (run the code)**

**info all -registers → show all register**

****

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**es 2 Write a program capable of printing the stack frame of a function. Try different techniques to distinguish stack frames. Nest 2 or more calls, see the chaining of SP and PC.**

**We can use gdb (outside) for print the stack frame or call inside system(gdb ) in a program c**

**int sum(int a,int b){**

**int p1=a;**

**int p2=b;**

**return p1+p2;**

**}**

**int sub(int a,int b){**

**int p1=a;**

**int p2=b;**

**return p1-p2;**

**}**

**int main (int args, char \*\* argv){**

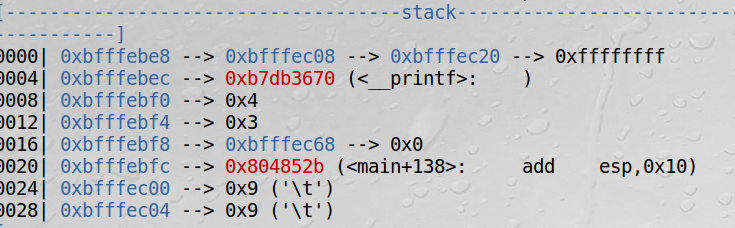
**int ris=sum(4,3);**

**int ris2=sub(9,9);**

**printf("the sum is %d",ris);**

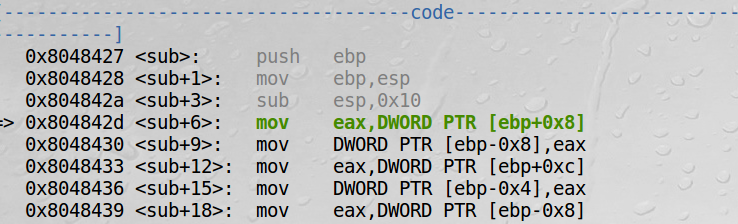
**printf("the sum is %d",ris2);**

**return 0;}**

****

**THE PARAMETERS passed to the function are also stored in stack . (4,3,9,9 )**

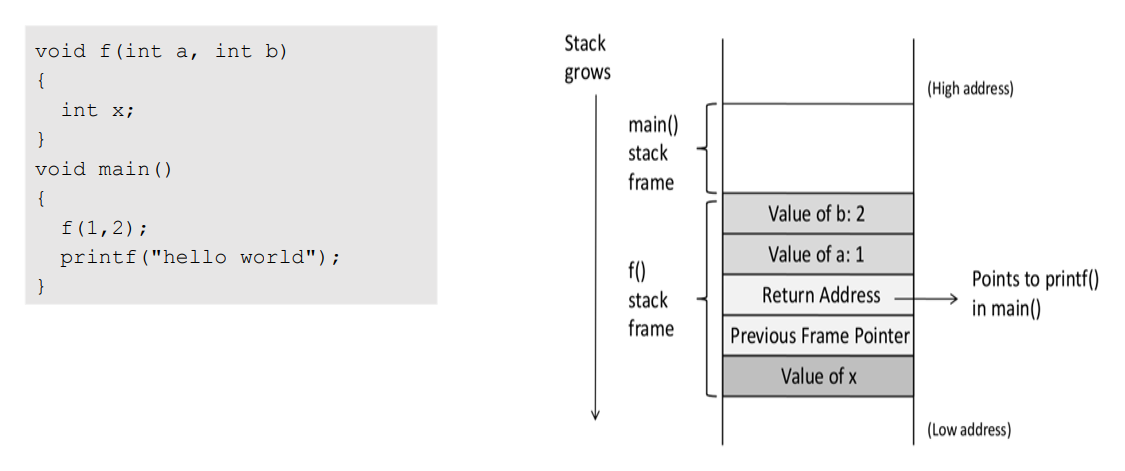
**We can observe that the stack grown in the beside way (up→ down ).**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

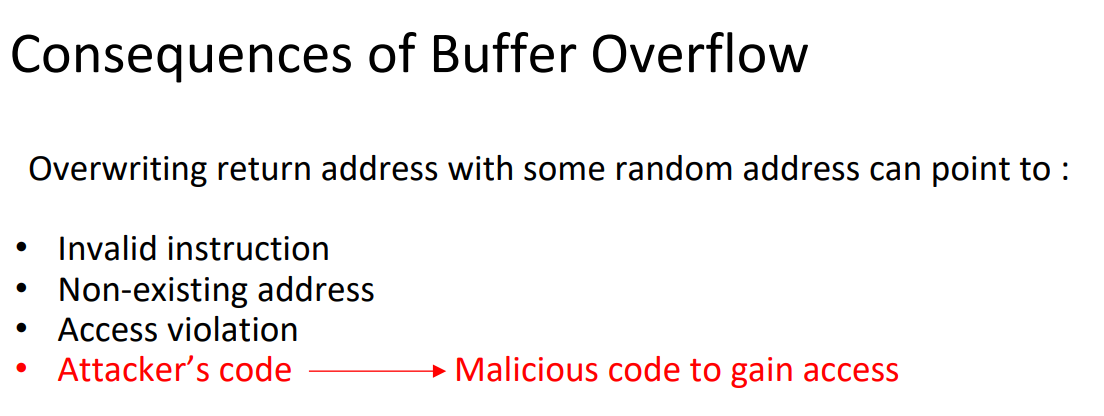
**3) Write a program that generates a buffer overflow zeroing the whole stack: observe the behavior. Check the same behavior if strcpy is used.**

**IMPORTANT OBSERVATION**

****

In each frame we have the value of variables , the pointer of the previous frame and **the RETURN ADDRESS !!! (it’s the the address to return when it completely execute the all code of the function). (eg: in the main, we call the function f ---> after the performance fo f , we need return the main, so in this case the return address is the next instruction after f : it’s the print function )**

**what the consequence of bufferOverlow？**

****

**3) Write a program that generates a buffer overflow zeroing the whole stack: observe the behavior. Check the same behavior if strcpy is used.**

**#include <string.h>**

**void foo(char \*str){**

**char buffer[12];**

**/\* The following statement will result in buffer overflow \*/**

**strcpy(buffer, str);**

**}**

**int main()**

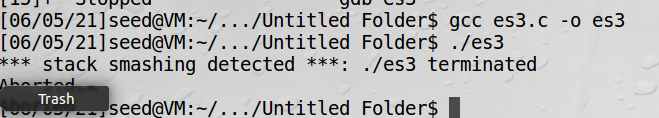
**{**

**char \*str = "This is definitely longer than 12";**

**foo(str);**

**return 1;**

**}**

****

**it give the same result if we use strcpy**

**for avoid buffer overflow > strncpy() instead of strcpy() and so on.**

**strncpy combats buffer overflow by requiring you to put a length in it. strcpy depends on a trailing \0, which may not always occur.**

**(copia fino al marcatore \0 “)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**4)Write a program that make the function return to itself in an infinite loop.**

**#include <string.h>**

**int foo()**

**{**

**return foo();**

**}**

**int main()**

**{**

**foo();**

**}**

****

**9/6/2021**

**%X → stampa indirizzo;**

**gcc -fno-stack-protecotor loop.c -o loop → compilare DISATTIVANDO STACK PROTECTOR ( cioè: if overwrite the return address or similar portions of the stack, Stack protection will abort your program)**