Information Security

LAB 3

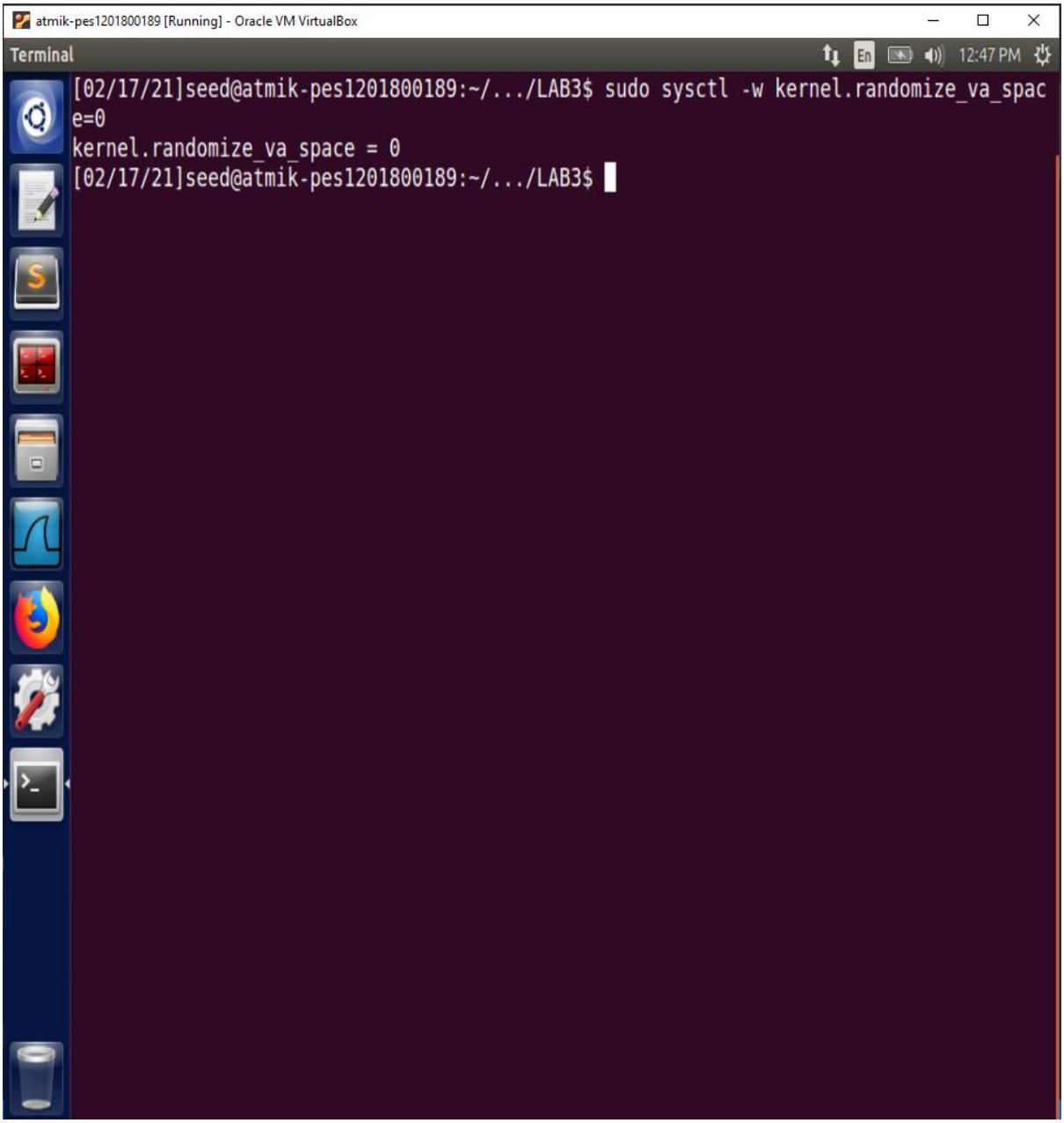
Name: Atmik Ajoy

SRN: PES1201800189

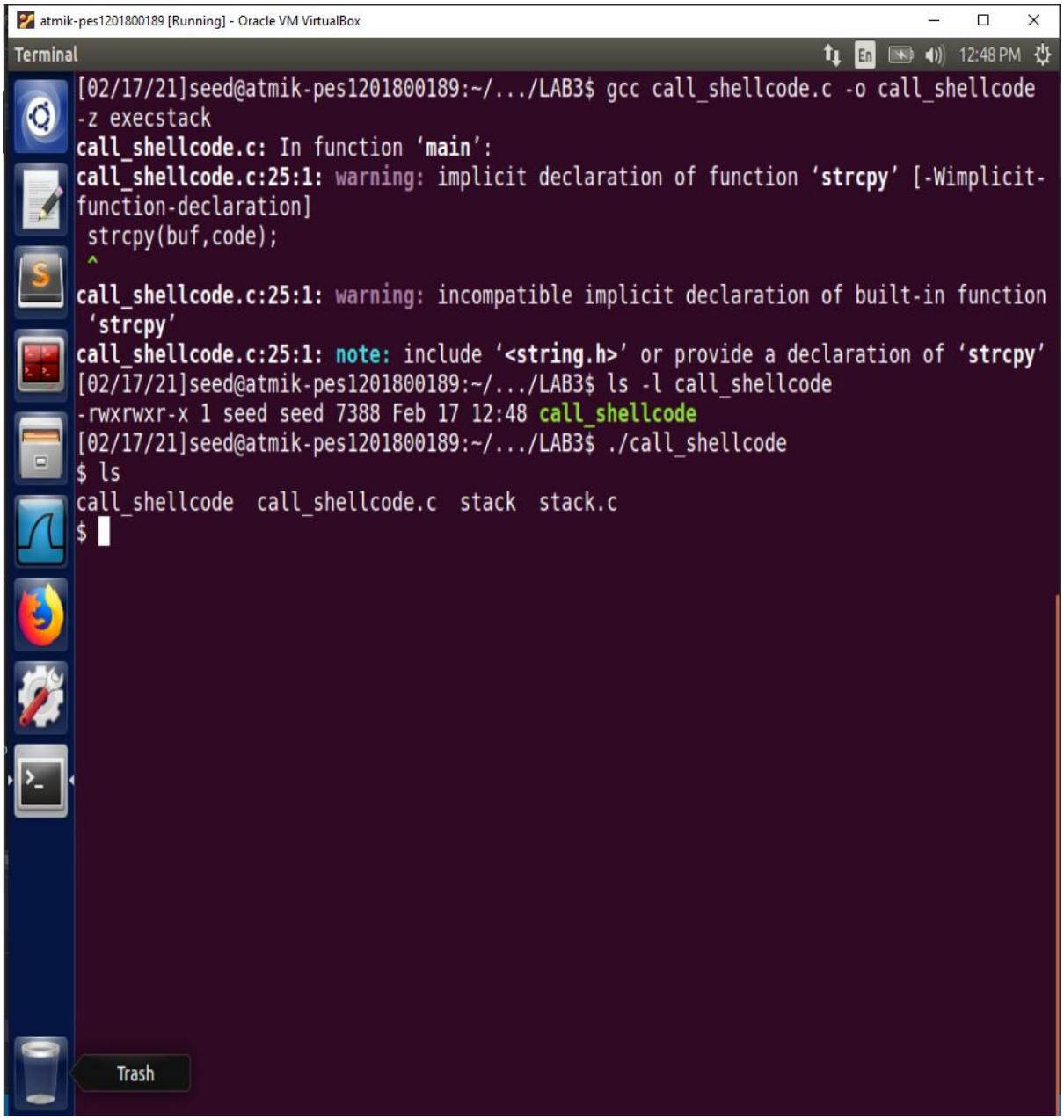
Section: ‘A’

Task 1:

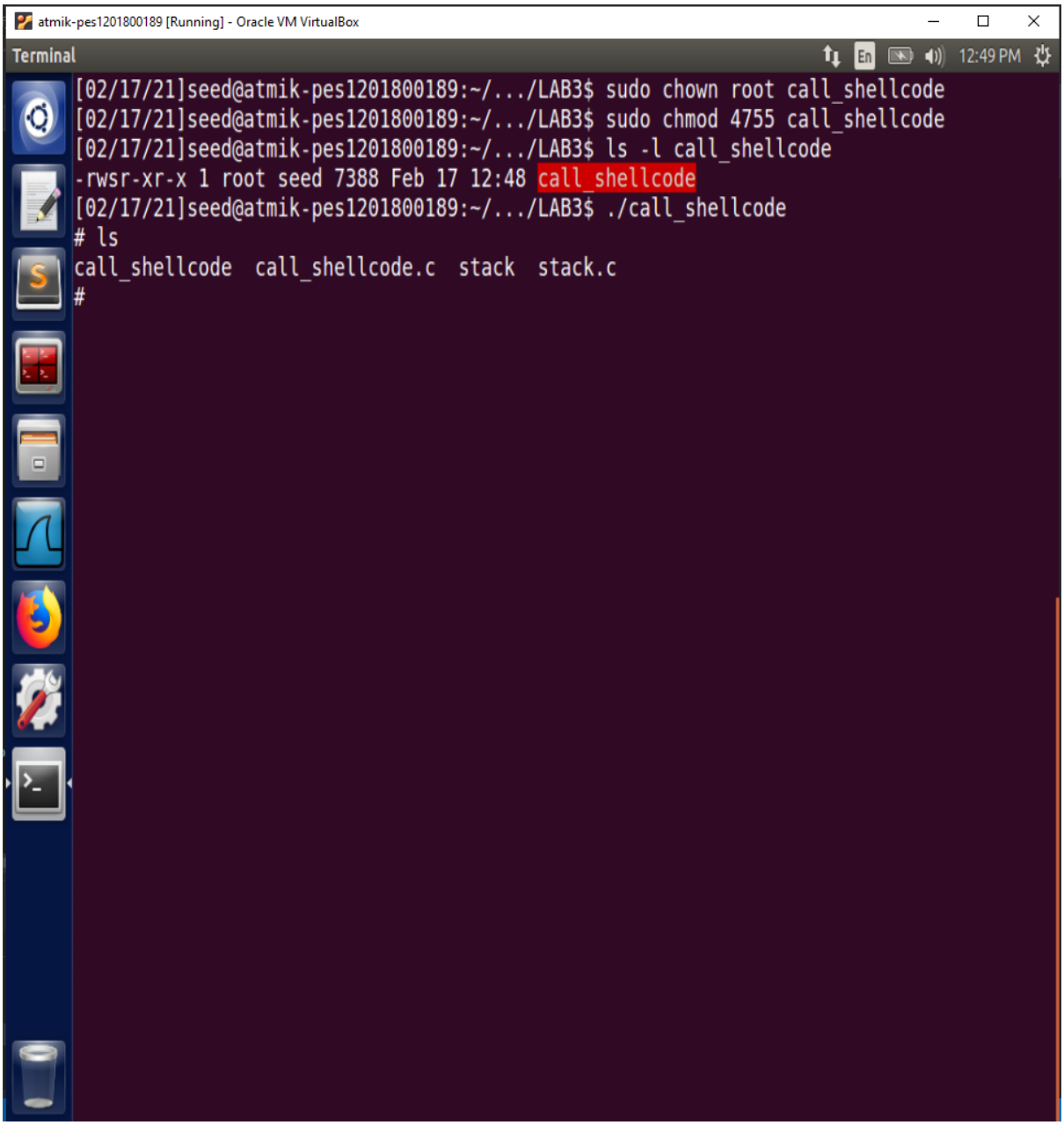
- Considering that ubuntu uses address space randomisation to randomise starting address of heap and stack, guessing them is very hard making it harder to actually carry out the buffer overflow attack. Since this lab is about carrying it out, we are going to disable the address randomisation with the command **kernel.randomize_va_space=0**



- Shellcode we have written is used to launch the shell and contains assembly level version of the code which we use stored in a buffer to launch the root shell to facilitate the attack
- ./call_shellcode invoked the aforementioned shell, hence successfully launching user shell in seed environment
- The shell program in bash uses principle of least privilege and drops its existing root privileges when invoked, thereby offering protection against buffer attacks.

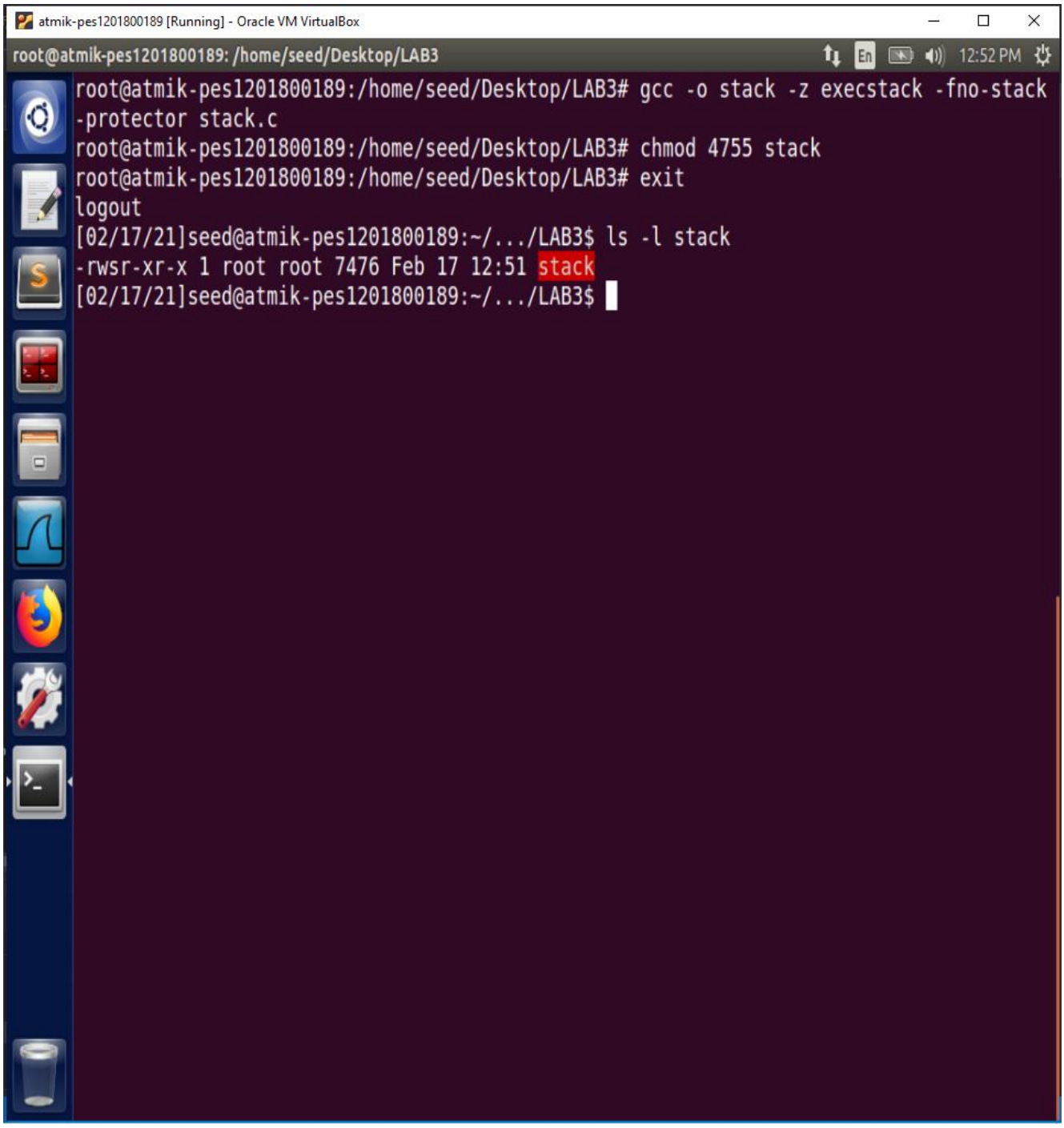


- The protection scheme is implemented in the /bin/sh file which is why we are replacing it with the /bin/zsh file since we want to try buffer overflow out.
- Now when we execute ./call_shellcode it invokes root shell in the /bin/zsh



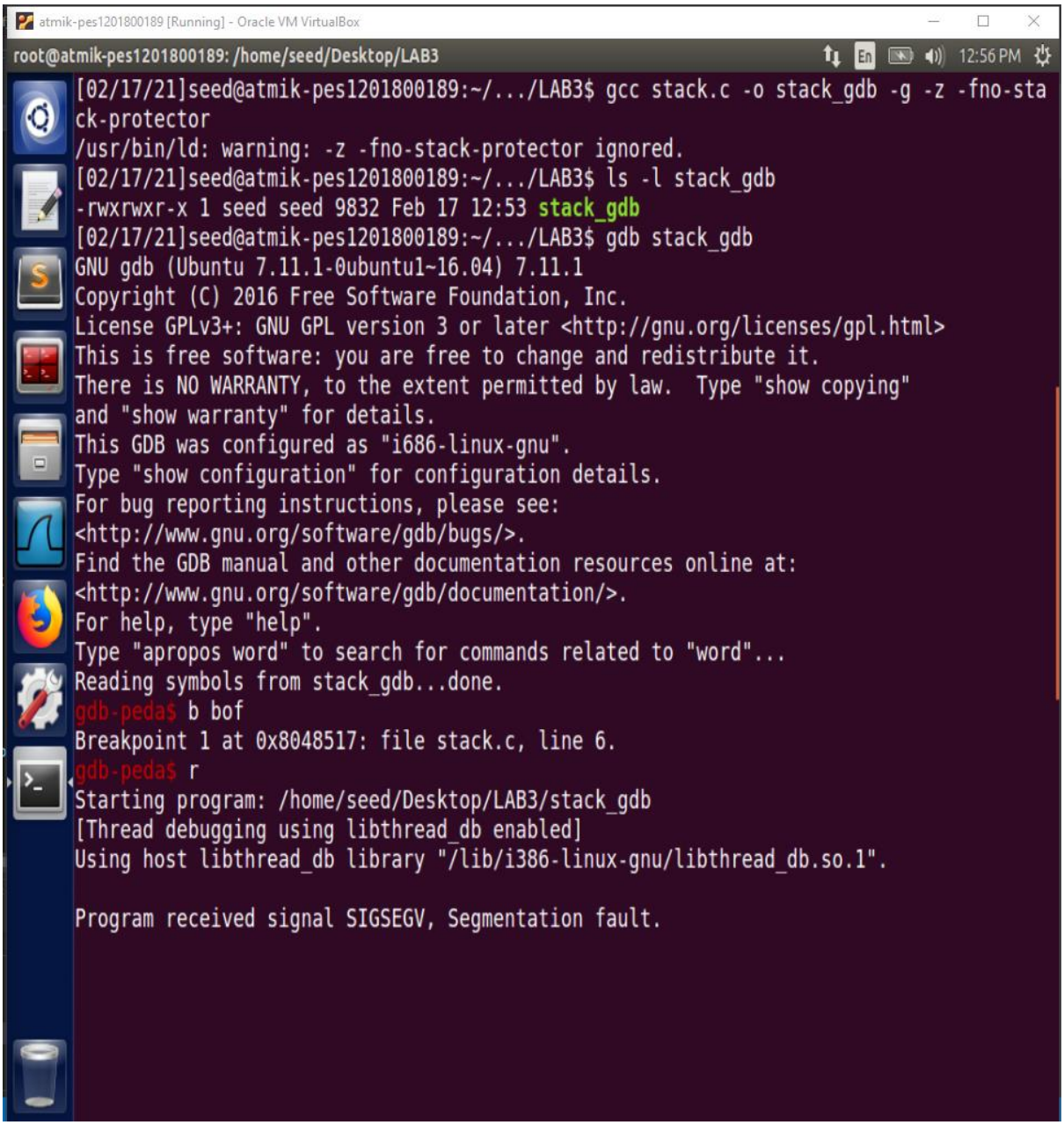
Task 2:

- Fread function used in this context reads 517 bytes from badfile that will be created in the next task
- there is code responsible for buffer overflow problem as str is of 517 bytes and buffer of 12 bytes, usually relating to a segmentation fault but when exploited for buffer overflow can launch root bash if run as set-uid prg



Task 3:

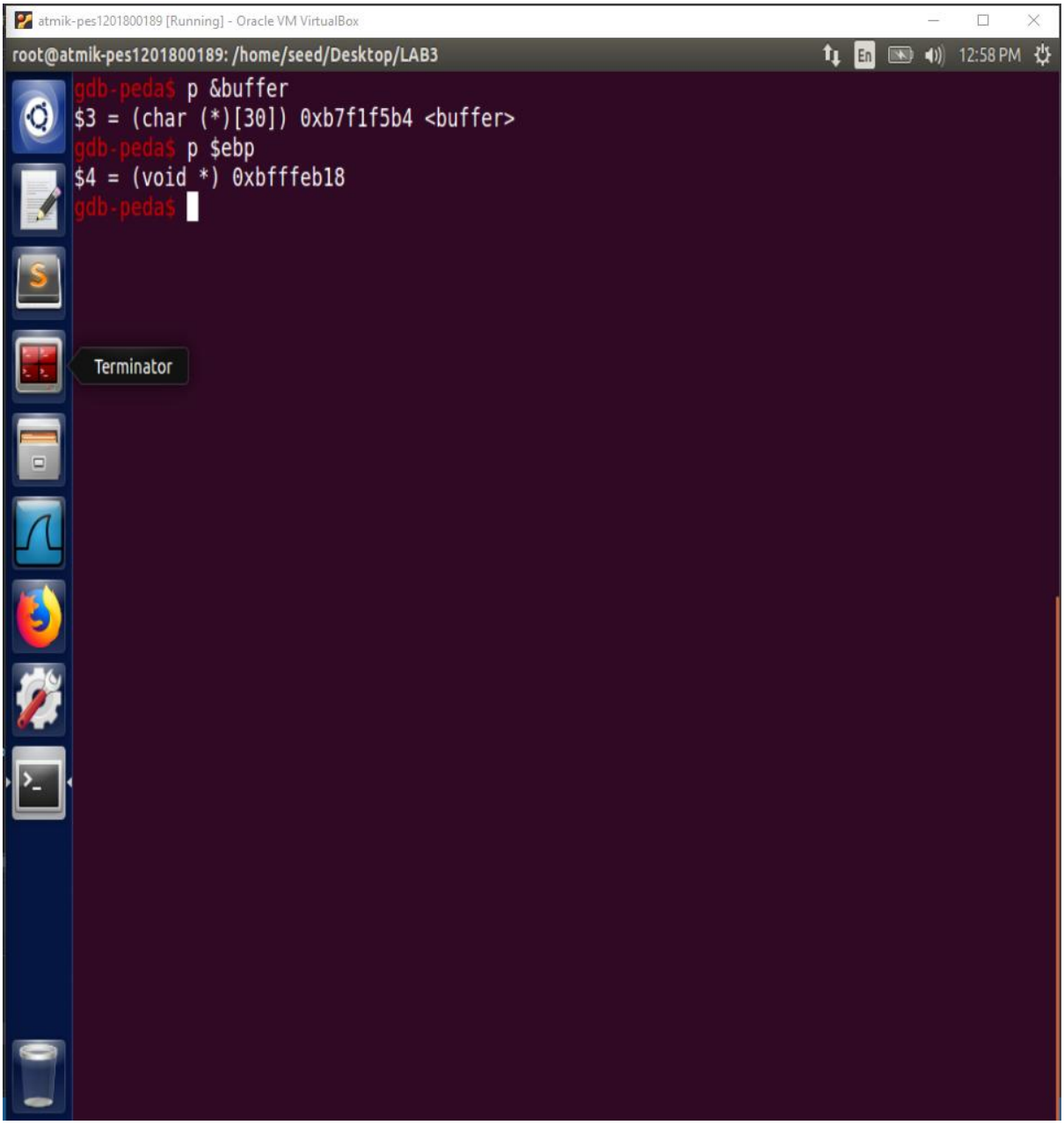
- **-fno-stack-protector** option and **-z execstack** option are used to turn off StackGuard and non executable stack protections
- **b bof** sets a breakpoint at the function bof() defined in the stack.c program. Its used later to obtain the address of the method.



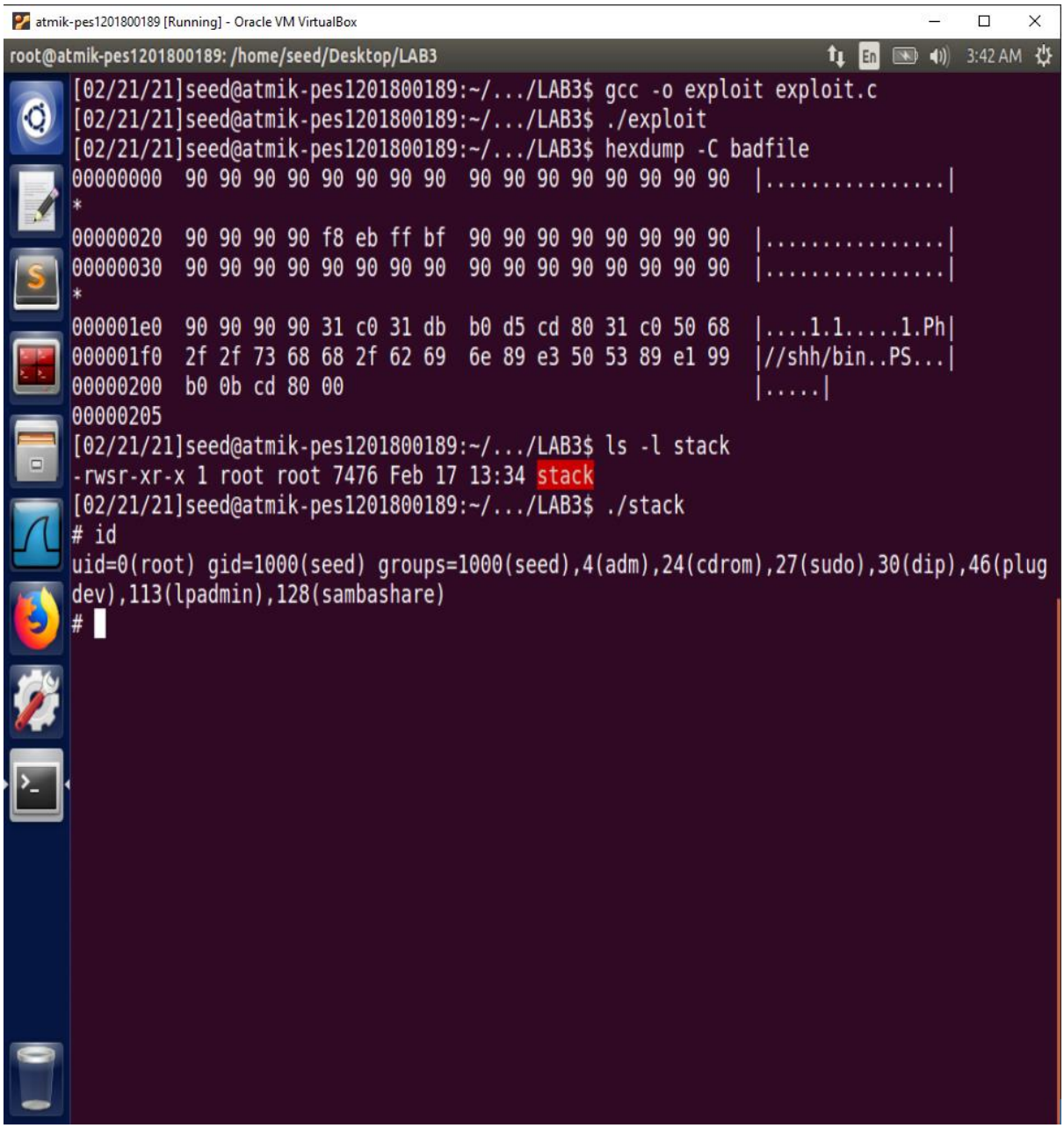
- the program runs until the breakpoint when **r** is entered , our break point is bof function which was set in the previous step and it also displays all the associated registers assigned to program run.

```
atmik-pes1201800189 [Running] - Oracle VM VirtualBox
root@atmik-pes1201800189: /home/seed/Desktop/LAB3
[----------------------------------registers-----------------------------------]
EAX: 0xbfffeb47 --> 0x34208
EBX: 0xb7f1c000 --> 0x1b1db0
ECX: 0xb7f1cbcc --> 0x25000
EDX: 0x0
ESI: 0x0
EDI: 0x205
EBP: 0xbfffeb18 --> 0xbfffed58 --> 0x0
ESP: 0xbfffeaf0 --> 0xb7fe96eb (<_dl_fixup+11>: add    esi,0x15915)
EIP: 0xb7dc88a6 (<__GI__IO_fread+38>:   mov    eax,DWORD PTR [esi])
EFLAGS: 0x10206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0xb7dc889a <__GI__IO_fread+26>:      imul   edi,DWORD PTR [ebp+0x10]
   0xb7dc889e <__GI__IO_fread+30>:      test   edi,edi
   0xb7dc88a0 <__GI__IO_fread+32>:      je     0xb7dc8948 <__GI__IO_fread+200>
=> 0xb7dc88a6 <__GI__IO_fread+38>:      mov    eax,DWORD PTR [esi]
   0xb7dc88a8 <__GI__IO_fread+40>:      and    eax,0x8000
   0xb7dc88ad <__GI__IO_fread+45>:      jne    0xb7dc88ea <__GI__IO_fread+106>
   0xb7dc88af <__GI__IO_fread+47>:      mov    edx,DWORD PTR [esi+0x48]
   0xb7dc88b2 <__GI__IO_fread+50>:      mov    ecx,DWORD PTR gs:0x8
[------------------------------------stack-------------------------------------]
0000| 0xbfffeaf0 --> 0xb7fe96eb (<_dl_fixup+11>:        add    esi,0x15915)
0004| 0xbfffeaf4 --> 0x0
0008| 0xbfffeaf8 --> 0xb7f1c000 --> 0x1b1db0
0012| 0xbfffeafc --> 0xb7f1c000 --> 0x1b1db0
0016| 0xbfffeb00 --> 0xbfffed58 --> 0x0
0020| 0xbfffeb04 --> 0xb7feff10 (<_dl_runtime_resolve+16>:      pop    edx)
0024| 0xbfffeb08 --> 0xb7dc888b (<__GI__IO_fread+11>:   add    ebx,0x153775)
0028| 0xbfffeb0c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
__GI__IO_fread (buf=0xbfffeb47, size=0x1, count=0x205, fp=0x0) at iofread.c:37
37      iofread.c: No such file or directory.
gdb-peda$ p &buffer
```

- **P &buffer** prints the address of the buffer that is used by the program
- **P $epb** prints the value of the register ebp which is stack pointer used

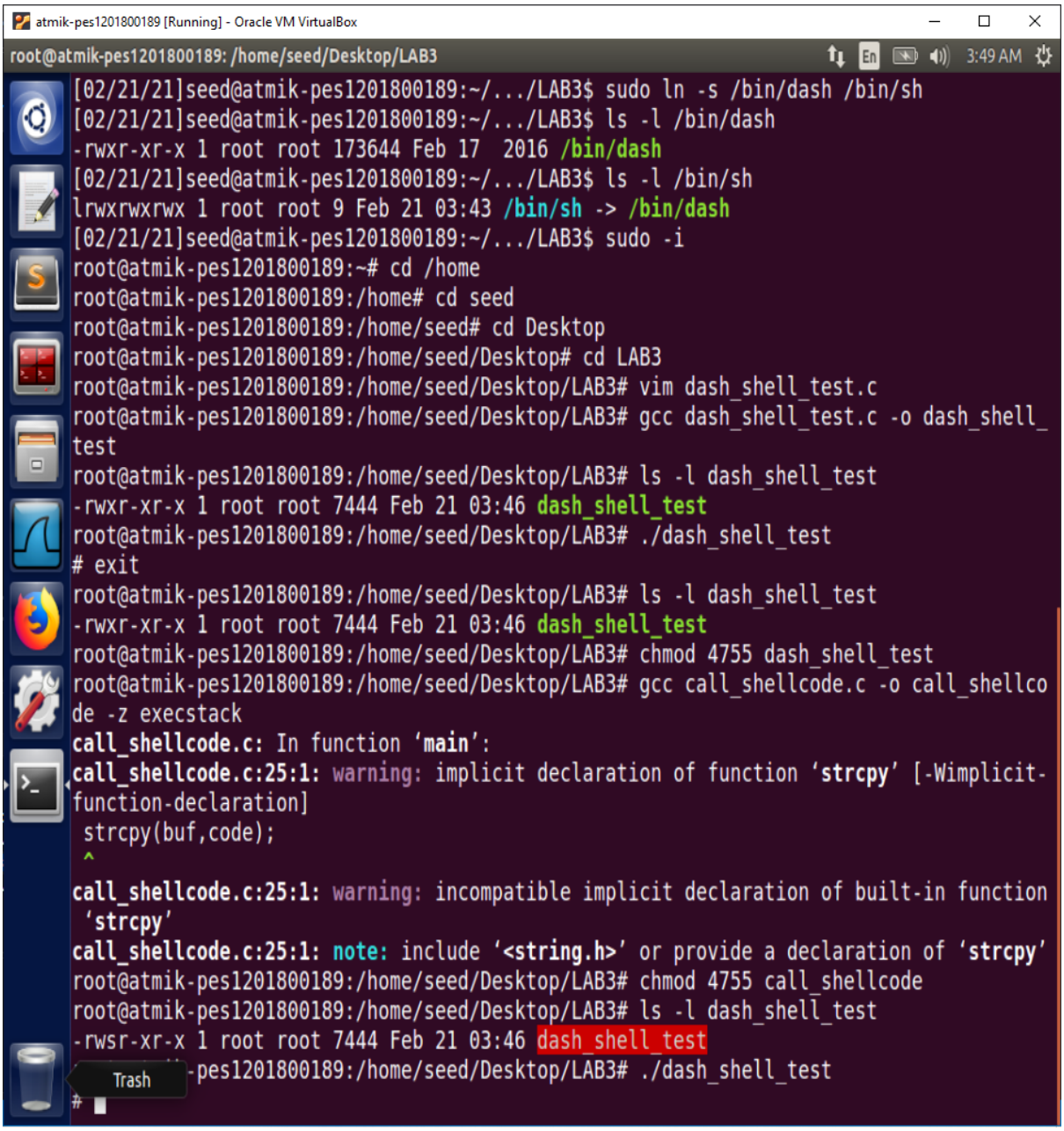

atmik-pes1201800189 [Running] - Oracle VM VirtualBox
root@atmik-pes1201800189: /home/seed/Desktop/LAB3
12:58 PM
gdb-peda$ p &buffer
$3 = (char (*)[30]) 0xb7f1f5b4 <buffer>
gdb-peda$ p $ebp
$4 = (void *) 0xbfffeb18
gdb-peda$
Terminator

- **Exploit** contains the code that creats the badfile which has large amounts of data which will be read by stack.c
- **Hexdump** displays the contents of the badfile which is created by exploit.c
- **./stack** executes the stack program and it has the buffer overflow flaw that the exploit code has assemble code to access root through

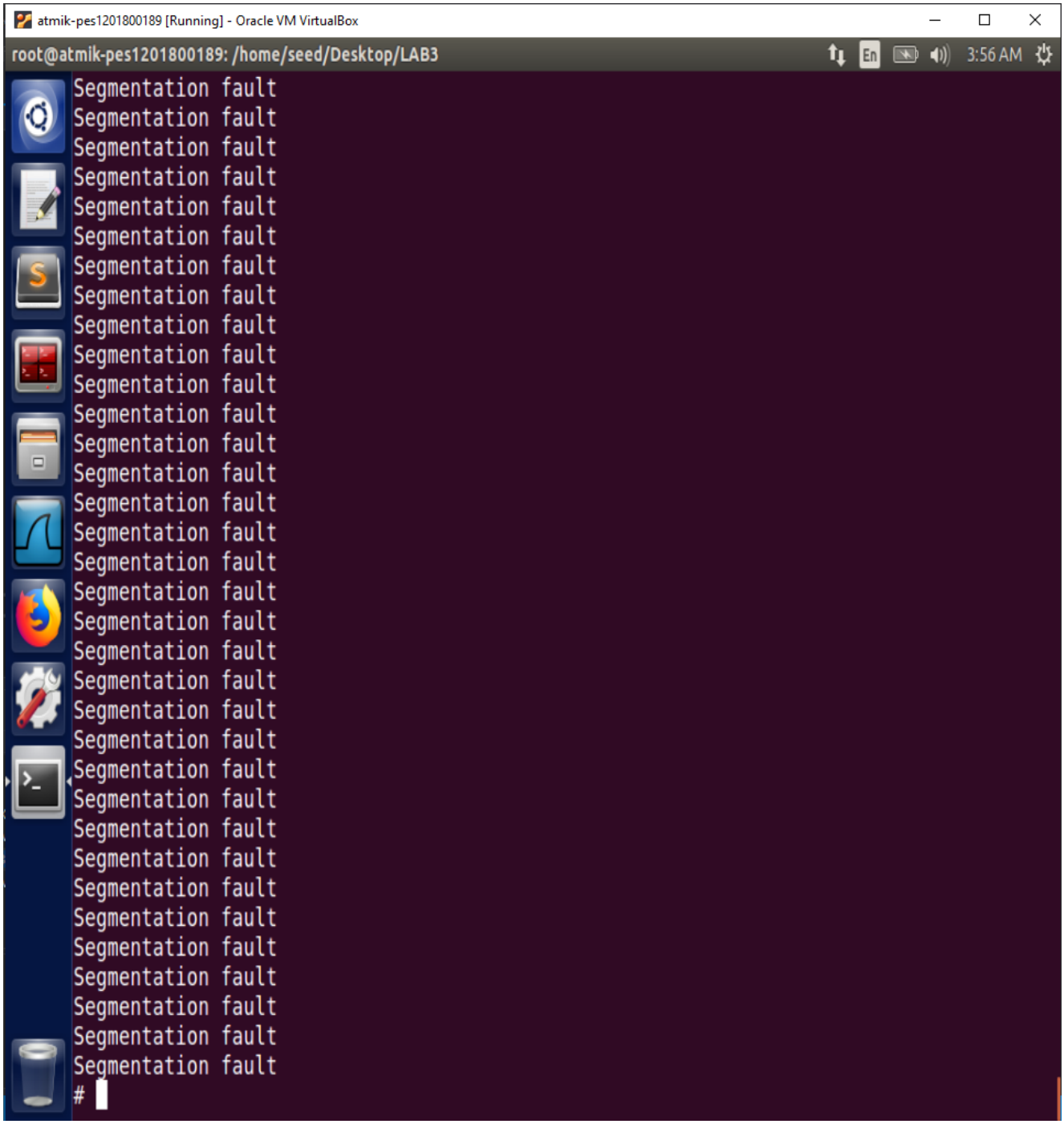


Task 4:

- **sudo ln -s /bin/dash /bin/sh** command changes the link /bin/sh in order to ensure dash to be running

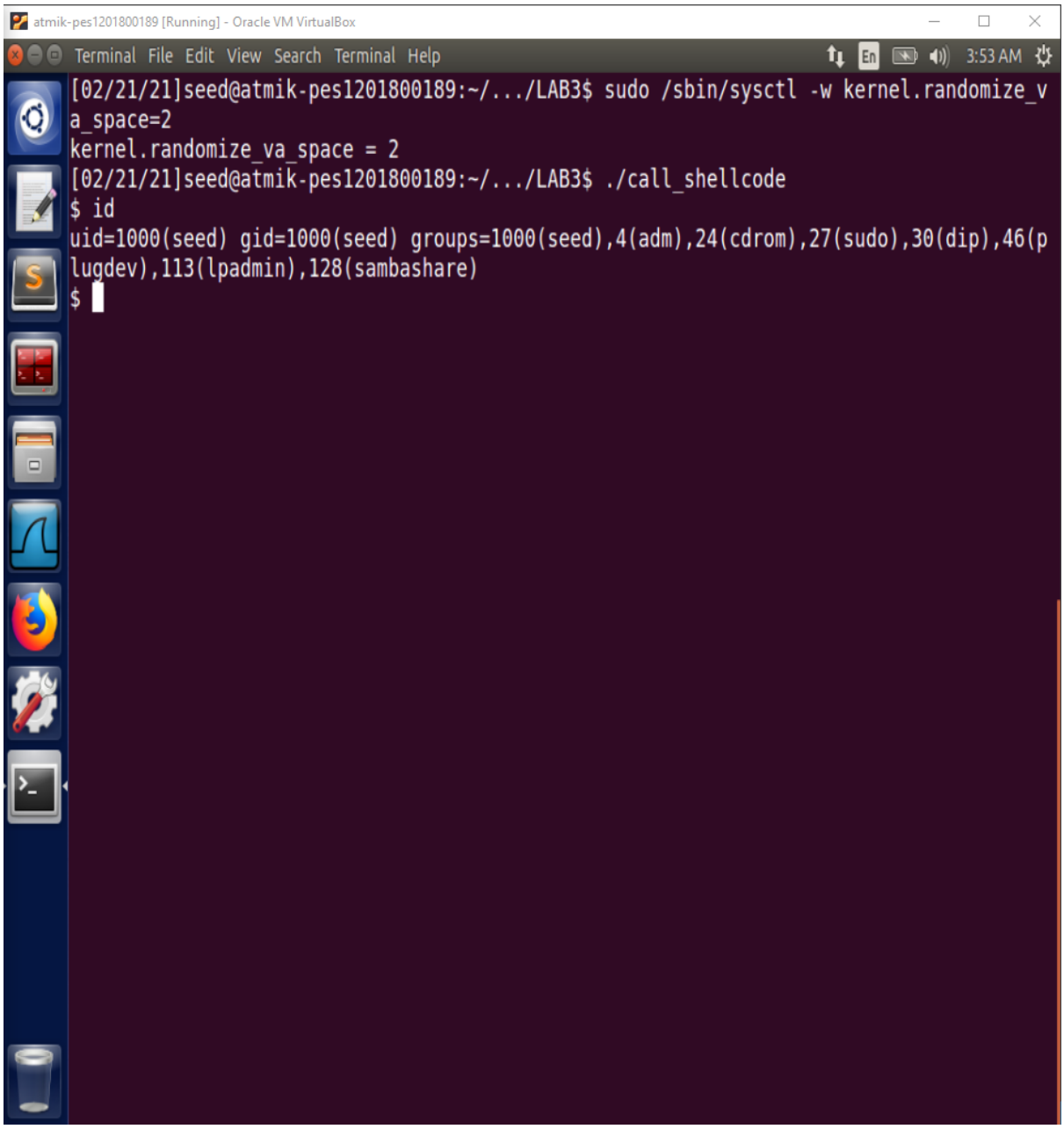


- **dash_shell_test.c** is compiled and executed as set-uid prg


atmik-pes1201800189 [Running] - Oracle VM VirtualBox
root@atmik-pes1201800189: /home/seed/Desktop/LAB3
3:56 AM
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
#

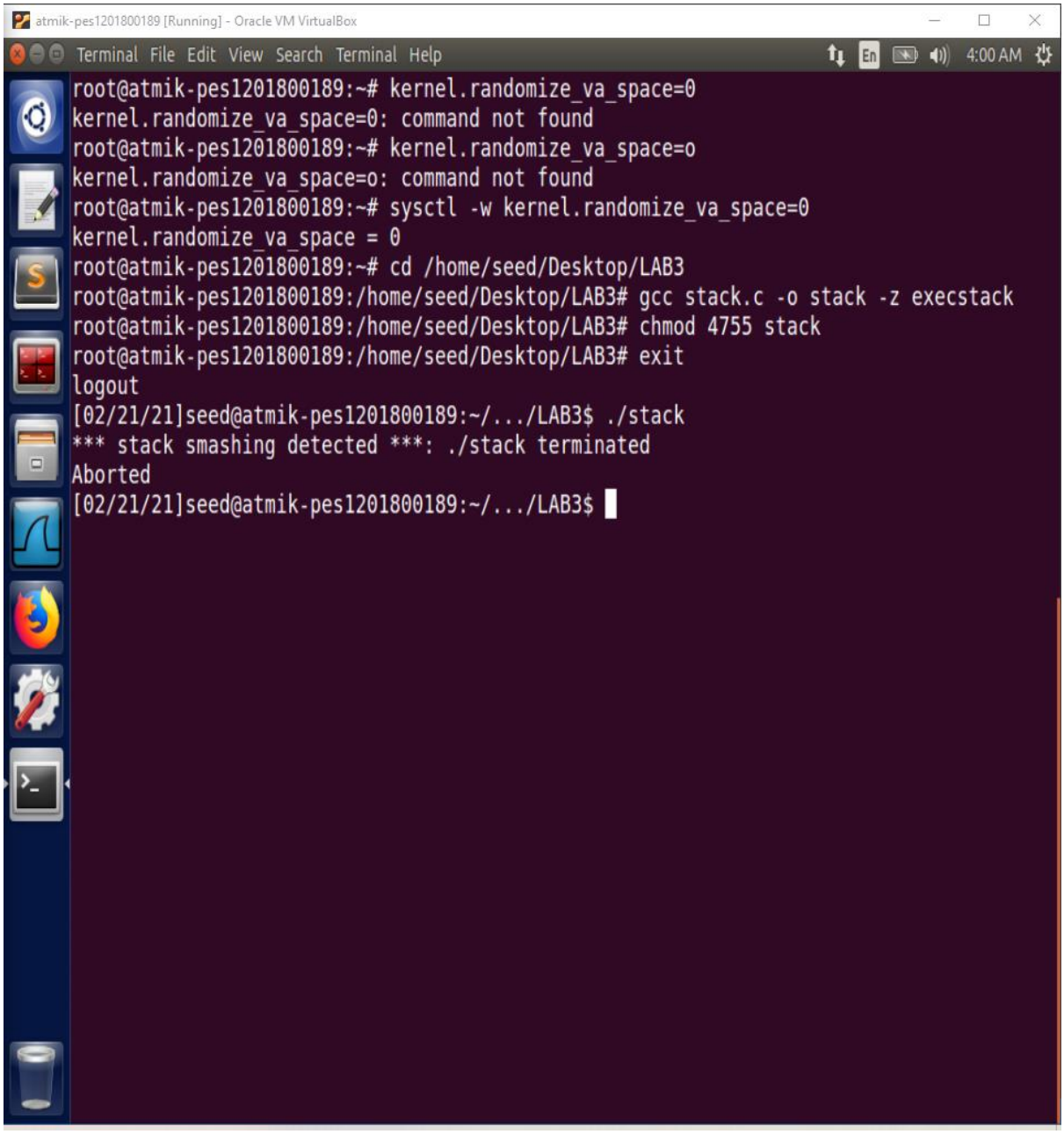
Task 5:

- **sudo /sbin/sysctl -w kernel.randomize_va_space=2** command actually ensures full randomization of all the addresses
- **sh -c "while [1]; do ./stack; done;"** command executes ./stack repeatedly which exploits buffer overflow vulnerability present in stack program, used to invoke shell with root privileges.



Task 6:

- when the stack.c is compiled without the -fno-stack-protector parameter, Stackguard is disabled
- **./stack** displays error and no shell is invoked



Task 7:

- When stack.c is compiled with noexecstack parameter, non executable stack protection is evoked
- ./stack doesn't invoke any shell and instead displays a segmentation fault