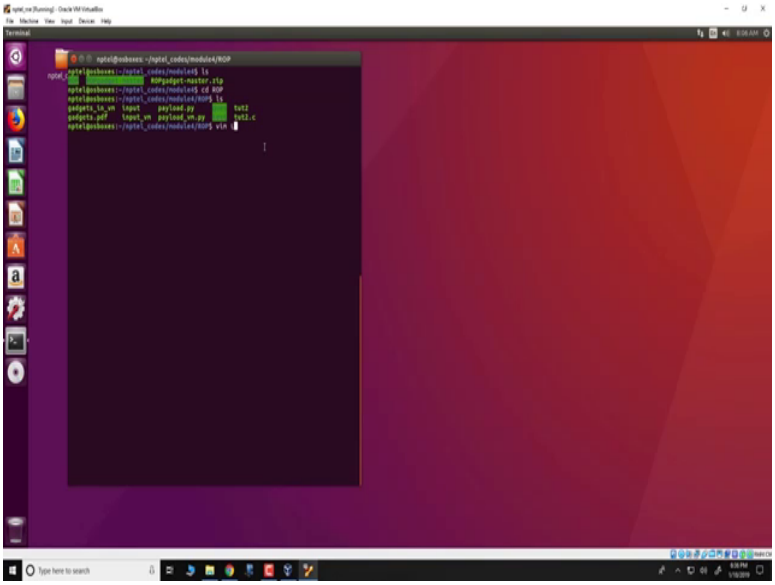
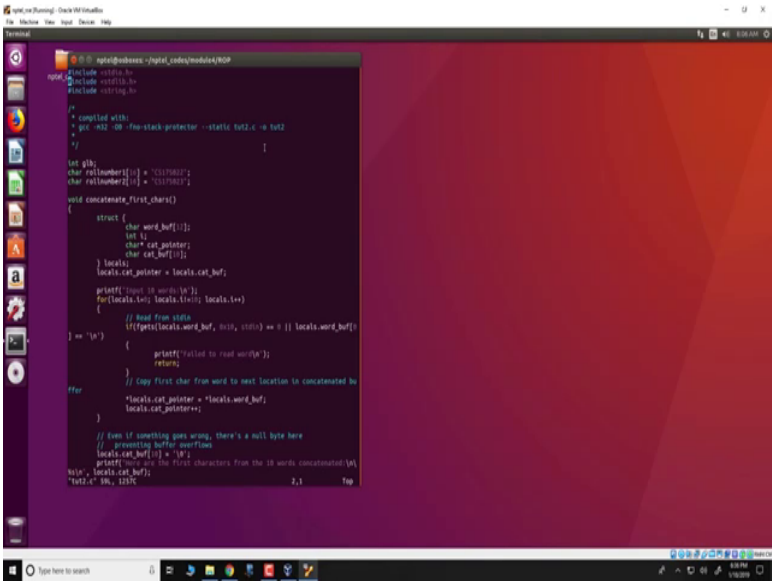
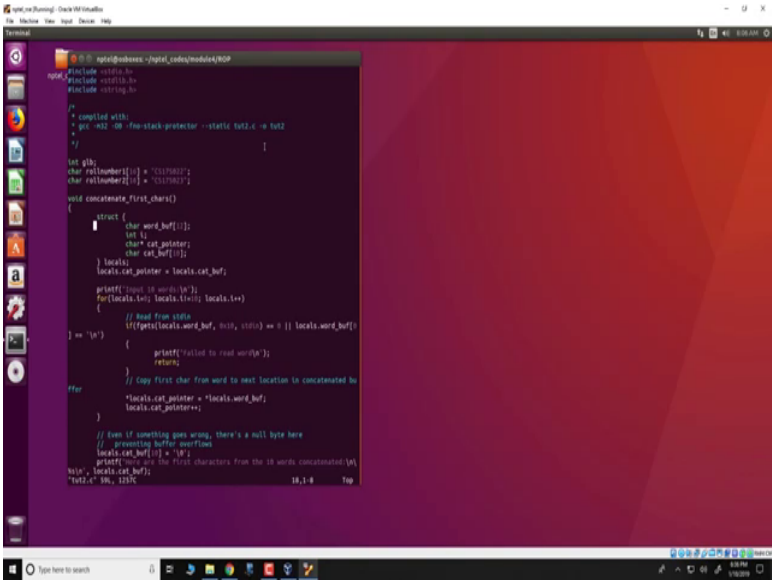
Information Security - 5 - Secure Systems Engineering
Prof. Chester Rebeiro
Indian Institute of Technology, Madras
Demonstration of a Return Oriented Programming (ROP) Attack

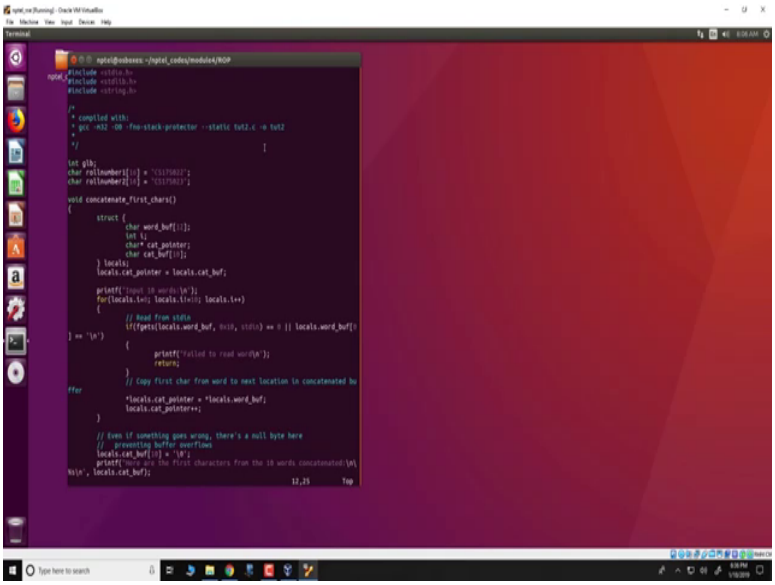
Hello and welcome to this demonstration in the course for secure system engineering, in this demonstration we will looking at ROP attack, now ROP attacks are extremely difficult to do, so we will be just glimpsing about one particular assignment which was submitted by some of the students in the course when I was actually teaching it.

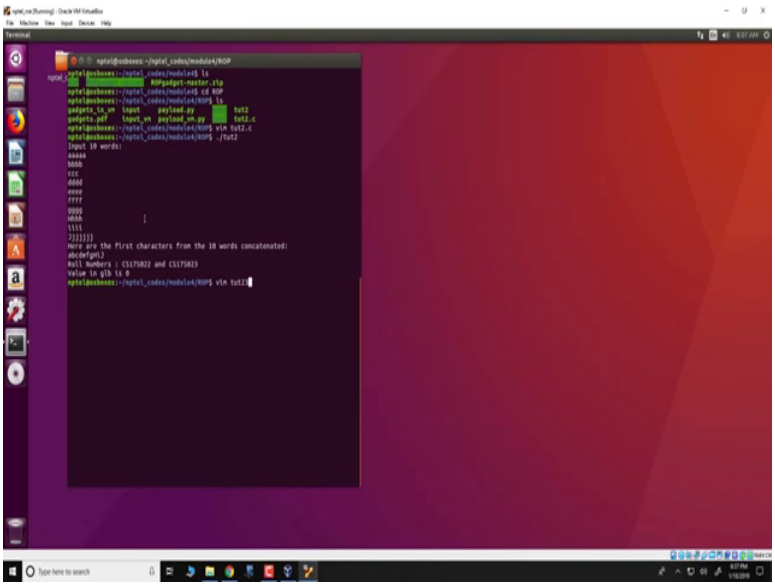
(Refer Slide Time: 0:37)

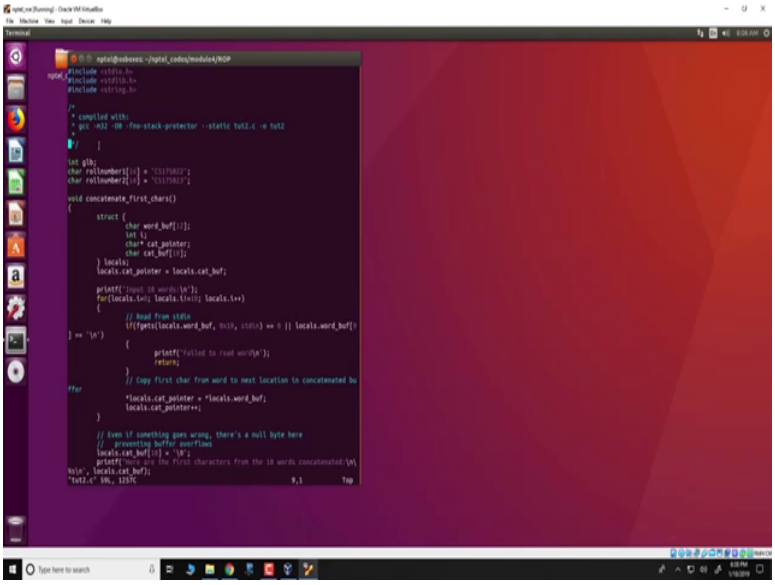


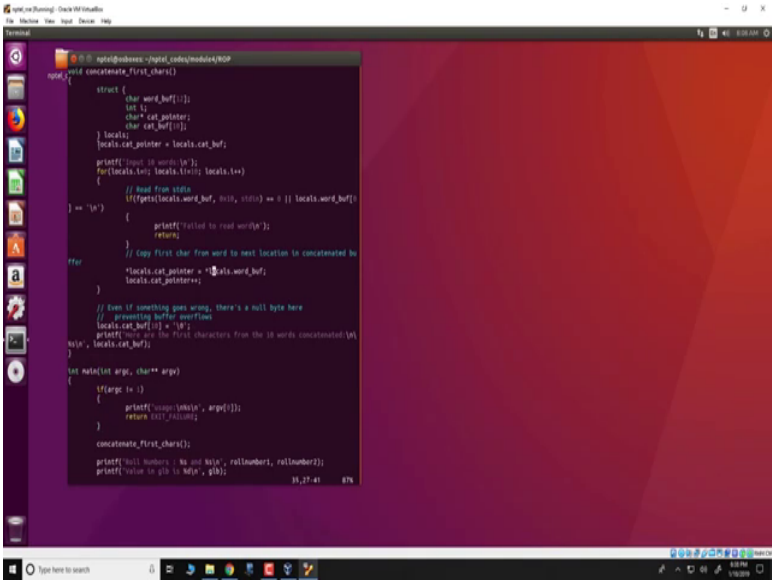


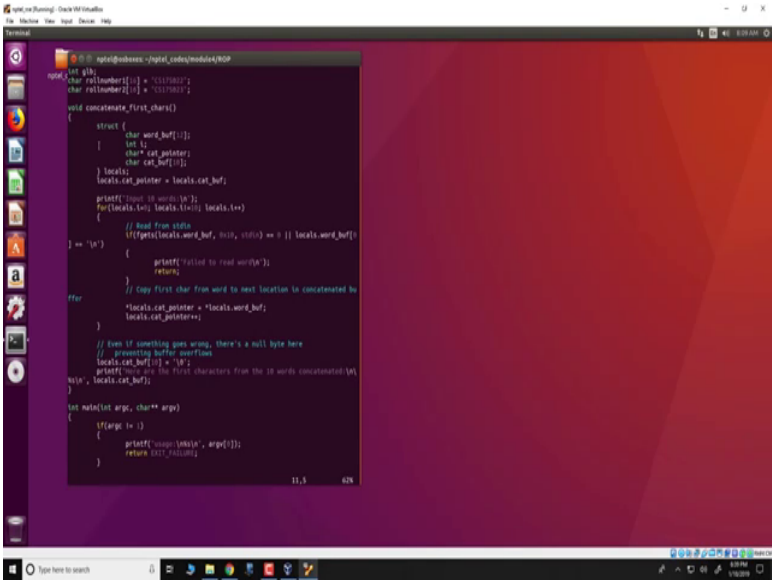

nptel@osboxes: ~/nptel_codes/module4/ROP
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/*
* compiled with:
* gcc -m32 -O0 -fno-stack-protector --static tut2.c -o tut2
*
*/
int glb;
char rollnumber1[16] = "CS175022";
char rollnumber2[16] = "CS175023";
void concatenate_first_chars()
{
struct {
char word_buf[12];
int i;
char* cat_pointer;
char cat_buf[10];
} locals;
locals.cat_pointer = locals.cat_buf;
printf("Input 10 words:\n");
for(locals.i=0; locals.i<10; locals.i++)
{
// Read from stdin
if(fgets(locals.word_buf, 0x10, stdin) == 0 || locals.word_buf[
] == '\n')
{
printf("Failed to read word\n");
return;
}
// Copy first char from word to next location in concatenated bu
ffer
*locals.cat_pointer = *locals.word_buf;
locals.cat_pointer++;
}
// Even if something goes wrong, there's a null byte here
// preventing buffer overflows
locals.cat_buf[10] = '\0';
printf("Here are the first characters from the 10 words concatenated:\n\
n\n", locals.cat_buf);
"tut2.c" 59L, 1257C
10,1-8
Top


nptel@osboxes: ~/nptel_codes/module4/ROP
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/*
* compiled with:
* gcc -m32 -O0 -fno-stack-protector --static tut2.c -o tut2
*
*/
int glb;
char rollnumber1[16] = "CS175022";
char rollnumber2[16] = "CS175023";
void concatenate_first_chars()
{
struct {
char word_buf[12];
int i;
char* cat_pointer;
char cat_buf[10];
} locals;
locals.cat_pointer = locals.cat_buf;
printf("Input 10 words:\n");
for(locals.i=0; locals.i<10; locals.i++)
{
// Read from stdin
if(fgets(locals.word_buf, 0x10, stdin) == 0 || locals.word_buf[
] == '\n')
{
printf("Failed to read word\n");
return;
}
// Copy first char from word to next location in concatenated bu
ffer
*locals.cat_pointer = *locals.word_buf;
locals.cat_pointer++;
}
// Even if something goes wrong, there's a null byte here
// preventing buffer overflows
locals.cat_buf[10] = '\0';
printf("Here are the first characters from the 10 words concatenated:\n\
n\n", locals.cat_buf);
12,25
Top


nptel@osboxes: ~/nptel_codes/module4/ROP
nptel@osboxes:~/nptel_codes/module4$ ls
ROPgadget-master.zip
nptel@osboxes:~/nptel_codes/module4$ cd ROP
nptel@osboxes:~/nptel_codes/module4/ROP$ ls
gadgets_in_vm input payload.py tut2
gadgets.pdf input_vm payload_vm.py tut2.c
nptel@osboxes:~/nptel_codes/module4/ROP$ vim tut2.c
nptel@osboxes:~/nptel_codes/module4/ROP$ ./tut2
Input 10 words:
aaaaa
bbbb
ccc
dddd
eeee
ffff
gggg
hhhh
iiii
jjjjjjj
Here are the first characters from the 10 words concatenated:
abcdefgHiJ
Roll Numbers : CS175022 and CS175023
Value in glb is 0
nptel@osboxes:~/nptel_codes/module4/ROP$ vim tut2.


nptel@osboxes: ~/nptel_codes/module4/ROP
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/*
* compiled with:
* gcc -m32 -O0 -fno-stack-protector --static tut2.c -o tut2
*
*/
int glb;
char rollnumber1[10] = "CS17S022";
char rollnumber2[10] = "CS17S023";
void concatenate_first_chars()
{
struct {
char word_buf[12];
int i;
char* cat_pointer;
char cat_buf[10];
} locals;
locals.cat_pointer = locals.cat_buf;
printf("Input 10 words:\n");
for(locals.i=0; locals.i<10; locals.i++)
{
// Read from stdin
if(fgets(locals.word_buf, 0x10, stdin) == 0 || locals.word_buf[0
] == '\n')
{
printf("Failed to read word\n");
return;
}
// Copy first char from word to next location in concatenated bu
ffer
*locals.cat_pointer = *locals.word_buf;
locals.cat_pointer++;
}
// Even if something goes wrong, there's a null byte here
// preventing buffer overflows
locals.cat_buf[10] = '\0';
printf("Here are the first characters from the 10 words concatenated:\n\
%s\n", locals.cat_buf);
}
"tut2.c" 59L, 1257C
9,1 Top


nptel@osboxes: ~/nptel_codes/module4/ROP
void concatenate_first_chars()
{
struct {
char word_buf[12];
int i;
char* cat_pointer;
char cat_buf[10];
} locals;
locals.cat_pointer = locals.cat_buf;
printf("Input 10 words:\n");
for(locals.i=0; locals.i<10; locals.i++)
{
// Read from stdin
if(fgets(locals.word_buf, 0x10, stdin) == 0 || locals.word_buf[0
] == '\n')
{
printf("Failed to read word\n");
return;
}
// Copy first char from word to next location in concatenated bu
ffer
*locals.cat_pointer = *locals.word_buf;
locals.cat_pointer++;
}
// Even if something goes wrong, there's a null byte here
// preventing buffer overflows
locals.cat_buf[10] = '\0';
printf("Here are the first characters from the 10 words concatenated:\n\
%s\n", locals.cat_buf);
}
int main(int argc, char** argv)
{
if(argc != 1)
{
printf("usage:\n%s\n", argv[0]);
return EXIT_FAILURE;
}
concatenate_first_chars();
printf("Roll Numbers : %s and %s\n", rollnumber1, rollnumber2);
printf("Value in glb is %d\n", glb);
35,27-41 87%

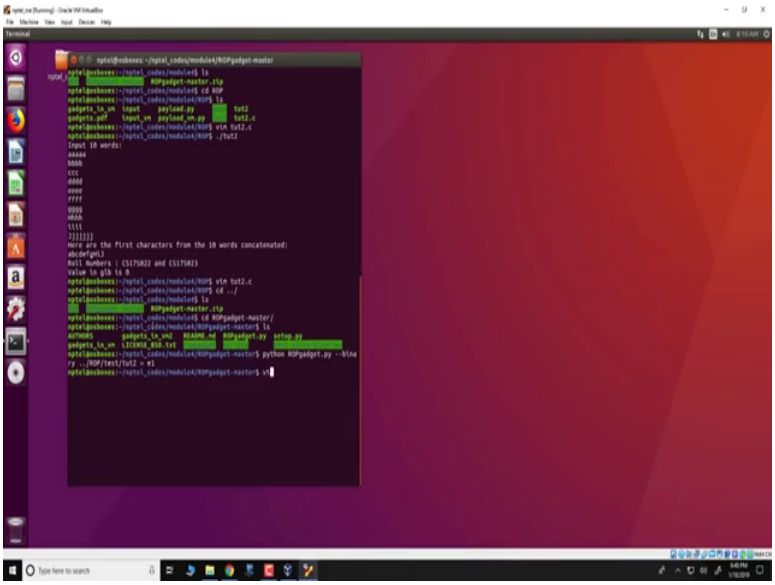

nptel@osboxes: ~/nptel_codes/module4/ROP
int glb;
char rollnumber1[10] = "CS17S022";
char rollnumber2[10] = "CS17S023";
void concatenate_first_chars()
{
struct {
char word_buf[12];
int i;
char* cat_pointer;
char cat_buf[10];
} locals;
locals.cat_pointer = locals.cat_buf;
printf("Input 10 words:\n");
for(locals.i=0; locals.i<10; locals.i++)
{
// Read from stdin
if(fgets(locals.word_buf, 0x10, stdin) == 0 || locals.word_buf[0
] == '\n')
{
printf("Failed to read word\n");
return;
}
// Copy first char from word to next location in concatenated bu
ffer
*locals.cat_pointer = *locals.word_buf;
locals.cat_pointer++;
}
// Even if something goes wrong, there's a null byte here
// preventing buffer overflows
locals.cat_buf[10] = '\0';
printf("Here are the first characters from the 10 words concatenated:\n\
%s\n", locals.cat_buf);
}
int main(int argc, char** argv)
{
if(argc != 1)
{
printf("usage:\n%s\n", argv[0]);
return EXIT_FAILURE;
}
11,5 62%

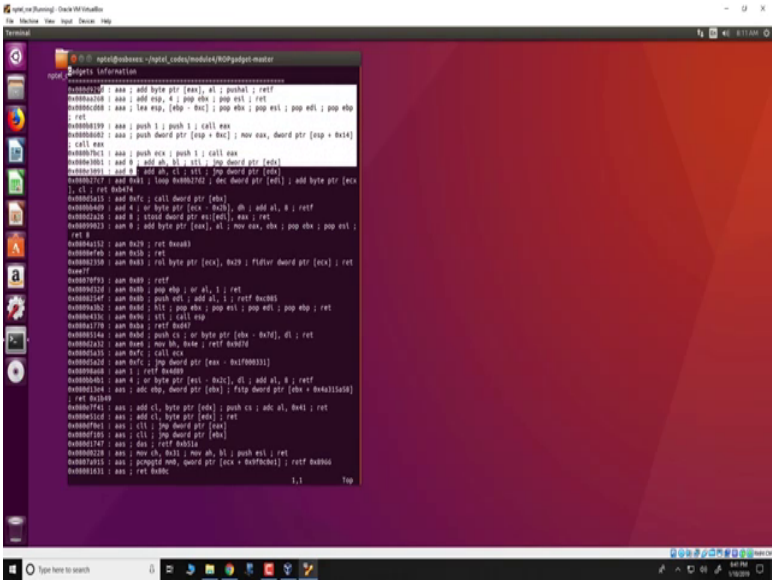
So this assignment essentially it work with a file in this directory called ROP and you can actually download this file, it is part of VM which comes along with this course and in the VM,, if the directory NPTEL codes module 4 ROP, you could actually find this required code, so the file that was attacked was TUT2.C or it actually tutorial 2.C and there is essentially did a lot of things, there was a main and there was a concatenate first chares the function and I have to give credit to the students who would this codes.

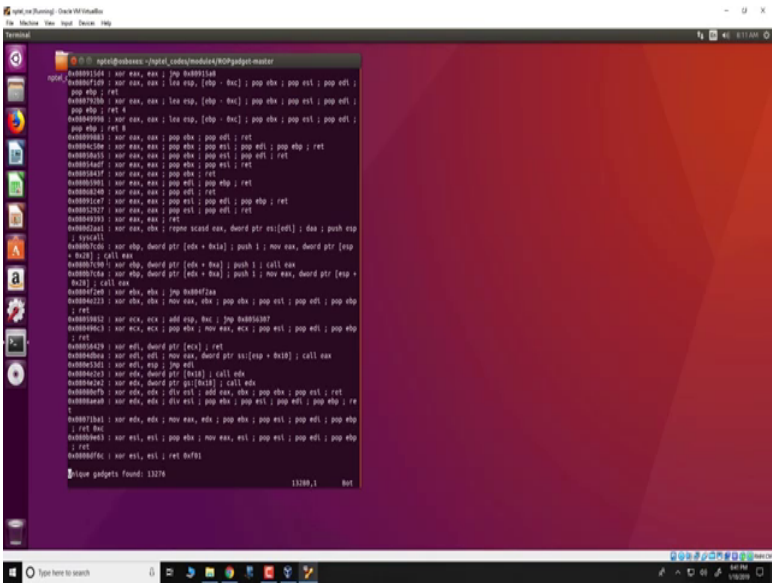
So their roll numbers are present here, the right working of this code is as follows, so you, this is the executable tutorial 2, it inputs ten words as like this and the output is the first characters of every input word, so you see over here that the first characters ABCD to J are actually printed out on the screen, so this is what this program actually does and there is a wonderbility in this particular program, the wonderbility is present in this function, concatenate first chares and we will all go into what the wonderbility is but believe it as an assignment and as a challenge to any of the viewers to actually try to find out these wonderbility in this function.

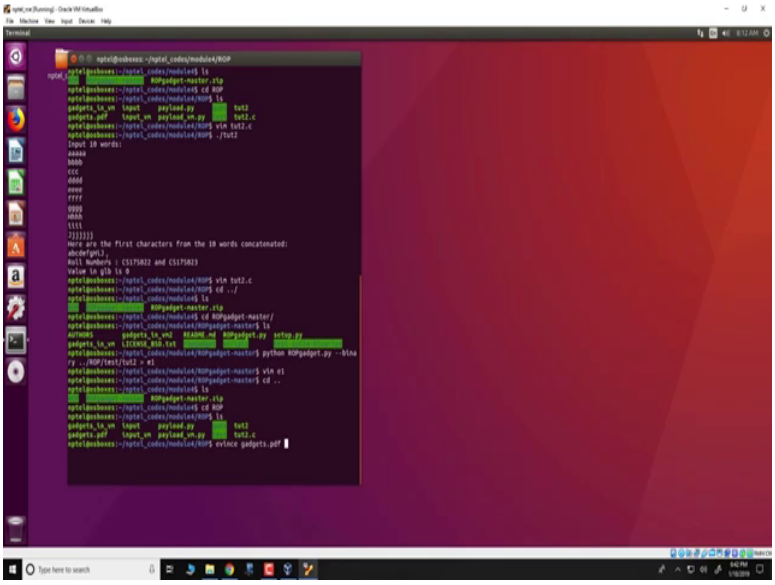
But what we will see over here is, we will try to build an ROP attack which uses that wonderbility, so the aim of the ROP attack is to somehow compute 10 factorial, as you know 10 factorial is the product of numbers from 1 to 10, so the aim of these attack is to determine or computer 10 factor using ROP gadgets and fill in the value of the result in this global variable GLB, so this GLB value would be printed somewhere, over here print GLB and you can actually look at this file and note that nowhere in the file we are actually using GLB anywhere else except in this print F and also note that we are not computing 10 factorial or any other factorial anywhere in this particular program.

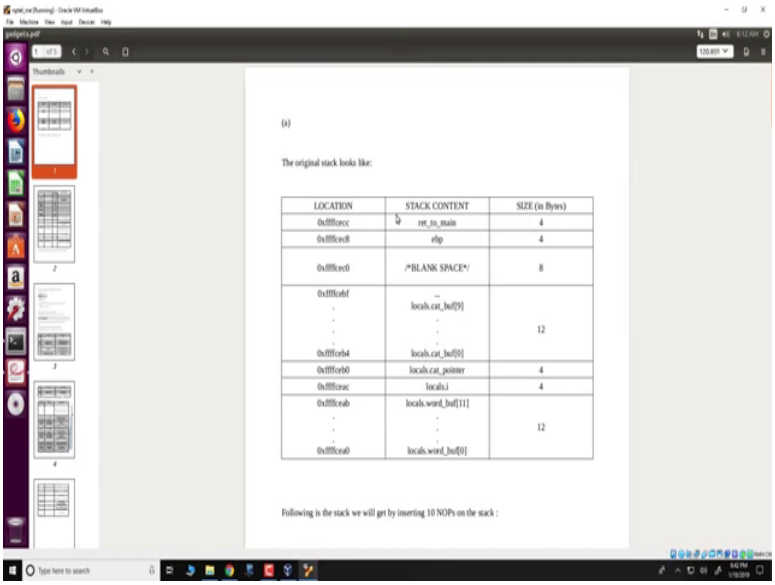
(Refer Slide Time: 3:35)

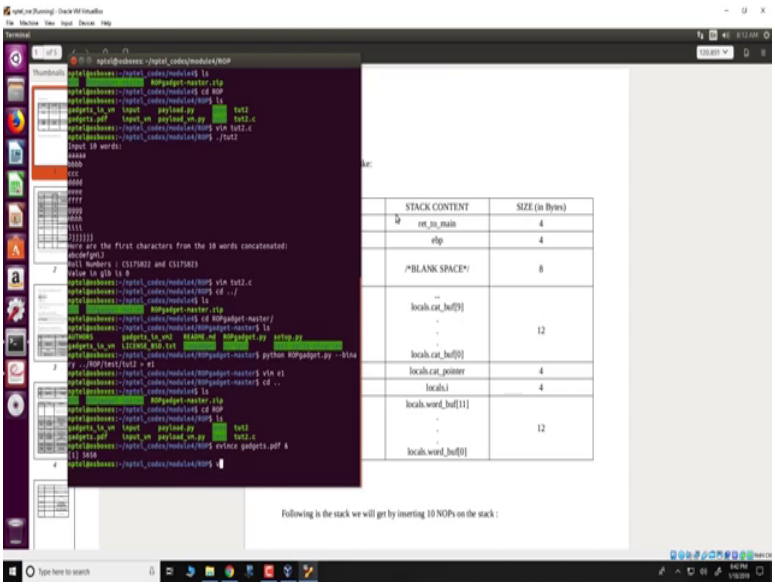


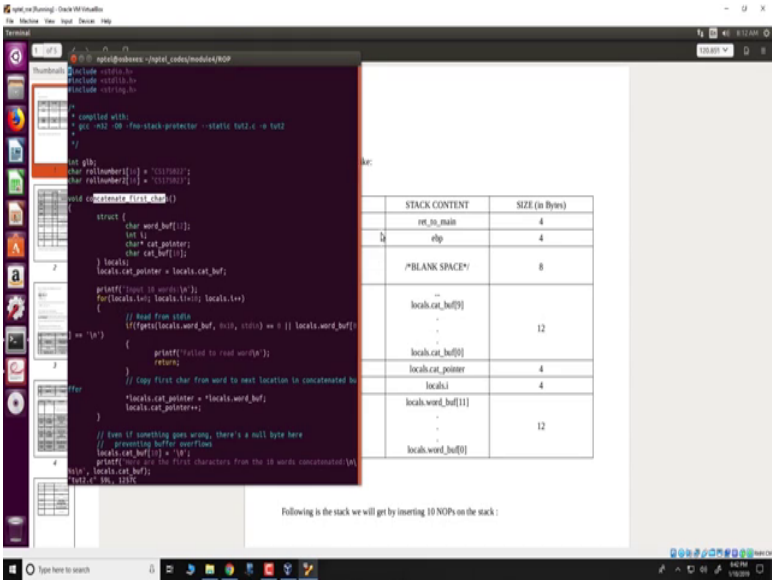





nptel@osboxes: ~/nptel_codes/module4/ROP
Here are the first characters from the 10 words concatenated:
abcdefgh1_
Roll Numbers : CS175022 and CS175023
Value in glb is 0


gadgets.pdf
Thumbnails
(a)
The original stack looks like:
LOCATION
STACK CONTENT
SIZE (in Bytes)
0xffffcecc
ret_to_main
4
0xffffcec8
ebp
4
0xffffcec0
/*BLANK SPACE*/
8
0xffffcebf
0xffffceb4
locals.cat_buf[9]
locals.cat_buf[0]
12
0xffffceb0
locals.cat_pointer
4
0xffffceac
locals.i
4
0xffffcea8
0xffffcea0
locals.word_buf[11]
locals.word_buf[0]
12
Following is the stack we will get by inserting 10 NOPs on the stack :


nptel@osboxes: ~/nptel_codes/module4/ROP
Here are the first characters from the 10 words concatenated:
abcdefgh1_
Roll Numbers : CS175022 and CS175023
Value in glb is 0
STACK CONTENT
SIZE (in Bytes)
ret_to_main
4
ebp
4
/*BLANK SPACE*/
8
locals.cat_buf[9]
locals.cat_buf[0]
12
locals.cat_pointer
4
locals.i
4
locals.word_buf[11]
locals.word_buf[0]
12
Following is the stack we will get by inserting 10 NOPs on the stack :



Okay, so starting with the ROP attack, the first thing to do with the ROP attack is to identify the gadgets which are present in the program, so the gadgets can be obtained using this particular, using a particular tool which is present in this directory ROP gadget master, it can go to this ROP gadget master and use the tool python ROP gadget.PY specify a binary as the input and provide the path to the executable, which you want to evaluate, so what this gadget tool would do?

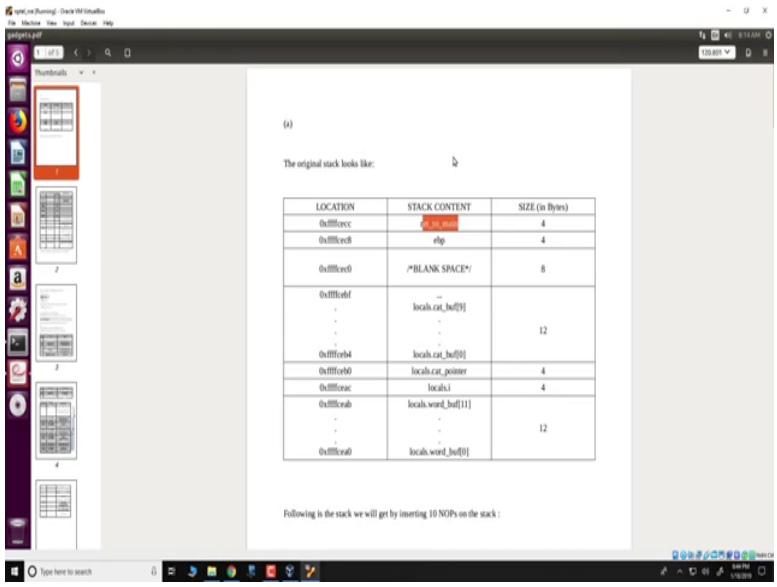
The ROP gadget tool would do was that it would pass through the entire executable, tutorial 2 and identify all possible gadgets that it can find, so we can just redirectory output into this file E1 okay and will open E1 and see that these are the gadgets that it will found okay, so note that the each of this gadgets has a few instructions and then has a return or a return F or call instruction at the end, this particular address over here is the address of the particular gadget.

So as you see the tool has found a large number of gadgets present in the executable tutorial 2 and in fact there are like 13,276 gadgets that are present, now a coming back to the payload that we want to write, we want to compute 10 factorial using these gadgets, so essentially what we need to do is pick out a gadgets from these 13,276, arrange them in a manner on the stack, so that in the end 10 factorial is computed.

So will not go into details about how we are actually picking up these gadgets but we will just see a report on this, so report is present over here okay, so we just go look at this report for the entire attack and what we see over here is the contains of the stack and this is the contains of the stack when the function concatenate first chares is been executed.

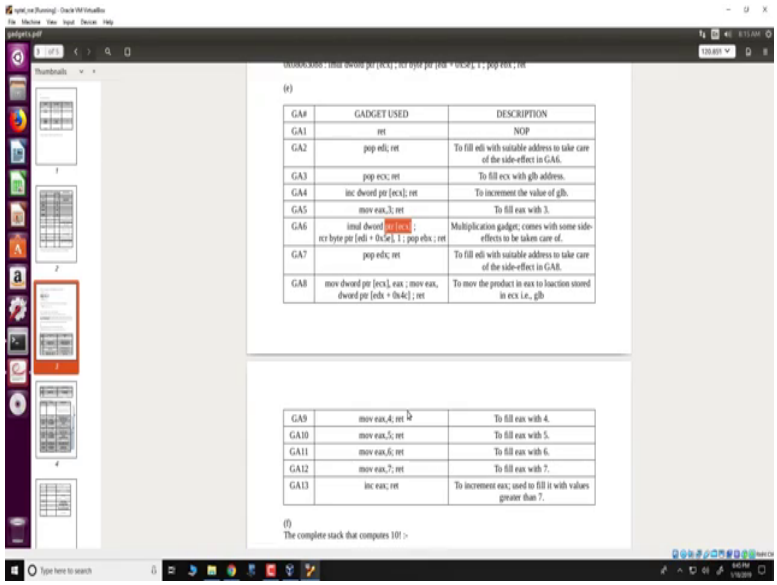
So as we know that when this function is executed there is a stack frame which is active and this frame look something like this will, so there is return address which is stored, which is of 4 bytes, then there is a previous frame pointer essentially the frame pointer corresponding to main, then there is some other data and locals which are present, so for our specific compilation the locations of the stack and the stack frame are present as follows over here but when you run it or when you actually try to execute this program, you may get a different value for these addresses, however the structure of the stack and the relativeness of wed, the various data are present on the stack would be identical.

(Refer Slide Time: 7:54)



So what an ROP attack actually does, is that it replaces this return to main and starts filling in gadgets over here, so starting from the first gadgets which is placed at this point, you would have gadgets going upwards like this, such that each of these gadgets have some operation in computing the 10 factorial.

(Refer Slide Time: 8:14)



(e)

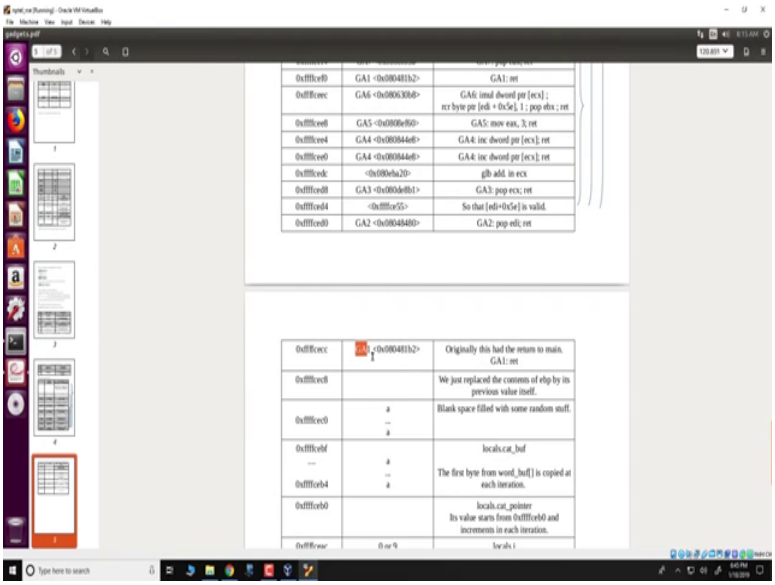
GA#	GADGET USED	DESCRIPTION
GA1	ret	NOP
GA2	pop edi; ret	To fill edi with suitable address to take care of the side-effect in GA6.
GA3	pop ecx; ret	To fill ecx with glb address.
GA4	inc dword ptr [ecx]; ret	To increment the value of glb.
GA5	mov eax,3; ret	To fill eax with 3.
GA6	imul dword ptr [ecx] ; rcr byte ptr [edi + 0x5e], 1 ; pop ebx ; ret	Multiplication gadget; comes with some side-effects to be taken care of.
GA7	pop ebx; ret	To fill edi with suitable address to take care of the side-effect in GA8.
GA8	mov dword ptr [ecx], eax ; mov eax, dword ptr [edx + 0x4c] ; ret	To mov the product in eax to loaction stored in ecx i.e., glb

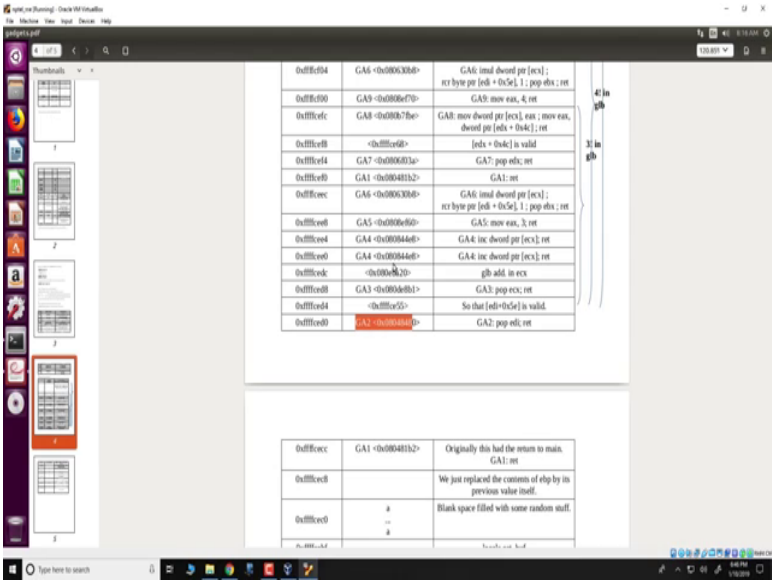
GA9	mov eax,4; ret	To fill eax with 4.
GA10	mov eax,5; ret	To fill eax with 5.
GA11	mov eax,6; ret	To fill eax with 6.
GA12	mov eax,7; ret	To fill eax with 7.
GA13	inc eax; ret	To increment eax; used to fill it with values greater than 7.

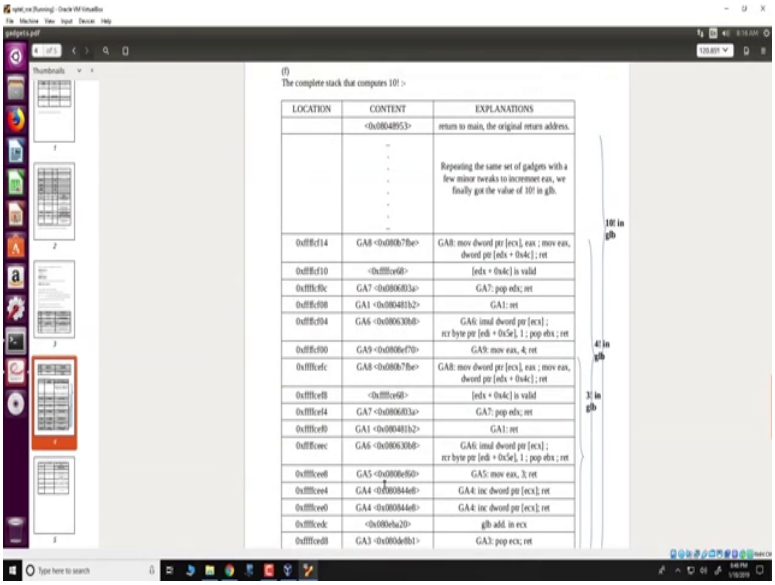
(f)
The complete stack that computes 10! :-

The gadgets that we use are actually there are 14 of them, pick from this space of that 13,000 odd gadgets which are found in tutorial 2 using the gadget tool and these are the 13 gadgets which were used, so these gadgets varied in functionality from very simple things like return which is equal an to just doing no-op operation, to something which is much more complicated like this gadget number 6, which is essentially integer multiplication, it multiplies some contents pointed to by these ECX register with the EX register, the description of each of this gadgets is present over here, so what we do in order to compute 10 factorial is we select gadgets and start to build a sequence of operations which he place on the stack such that after all of these gadgets complete its execution, we would be able to complete or obtain 10 factorial.

(Refer Slide Time: 9:24)

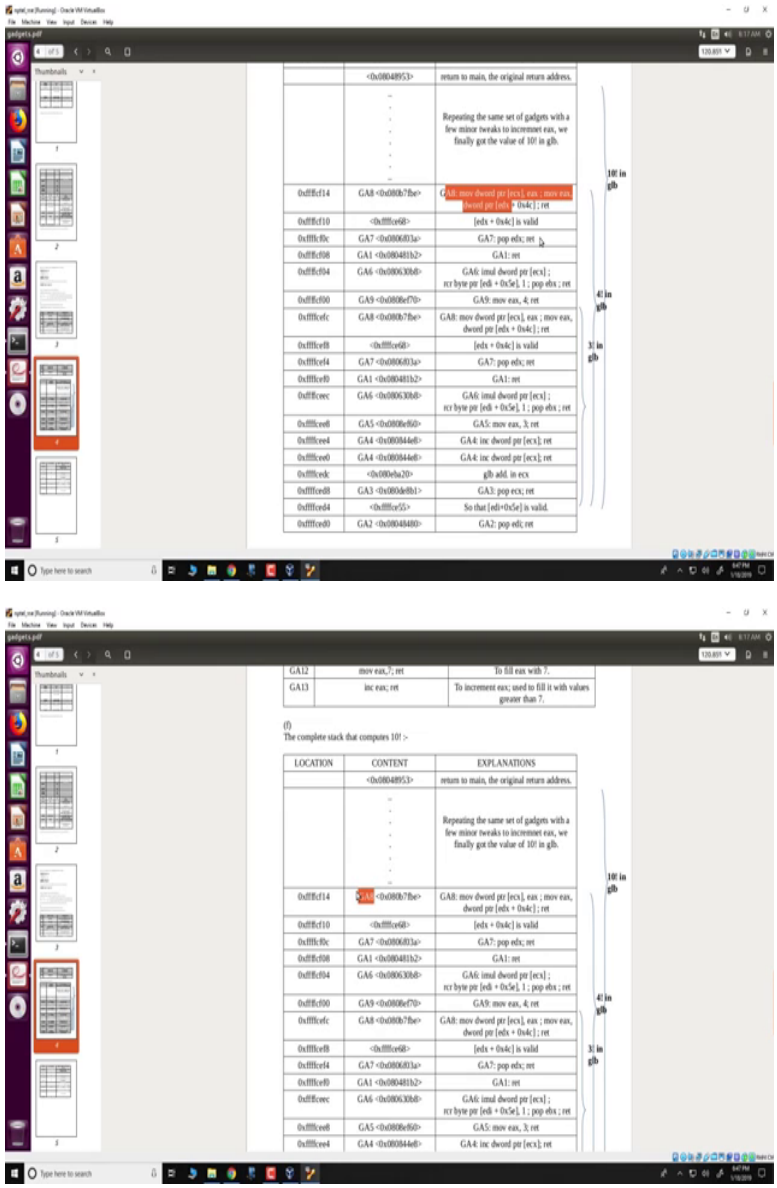






The modified stack with all gadgets placed look something like this, so it starts from gadget number 1 placed in this location, this location originally had the return to main which we had seen and now we overflow a buffer and replace this return to main this sequence of gadgets, so what happens during executions, is first gadget 1 executes and the return in gadget 1 would pick this address from the stack which corresponds to gadget 2 and execute, than gadget 3, gadget 4 and so on executes in such a way, the sequence of placing the gadgets in such a way so that at the end of the sequence 10 factorial is computed.

(Refer Slide Time: 10:14)

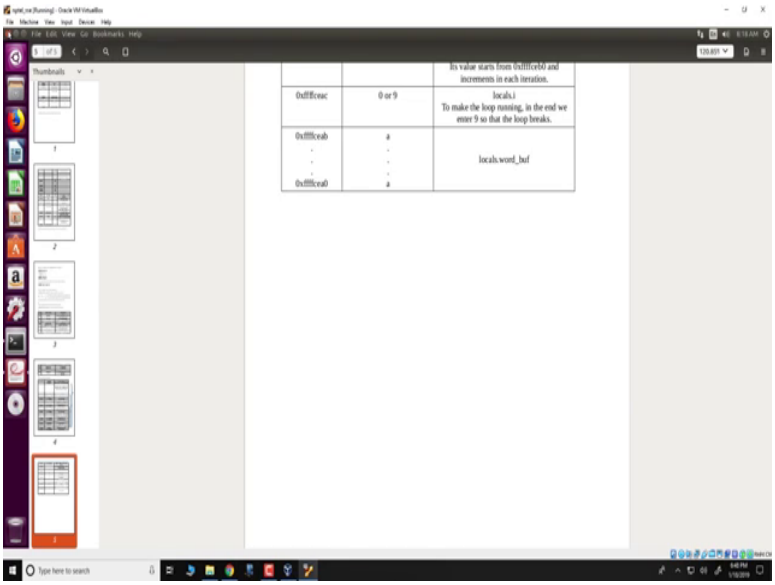


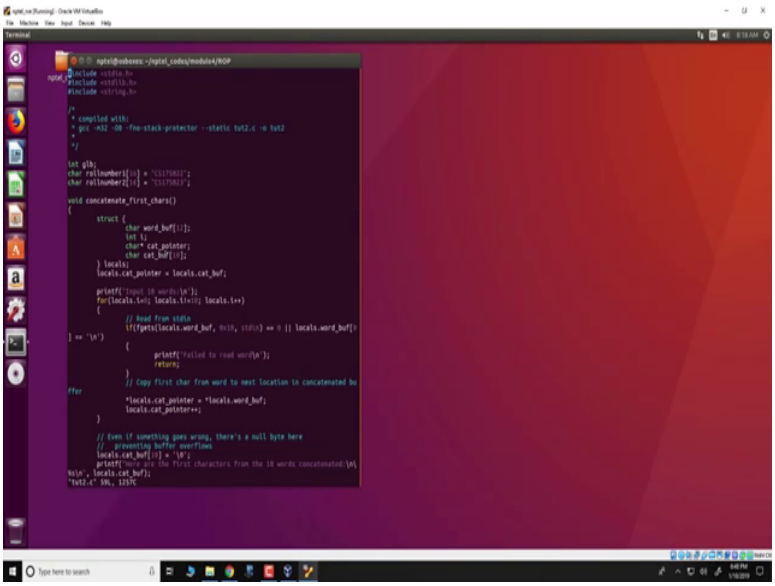
To be more specific for example each of these sequences for example at this position we would have computed 3factorial and the result for 3 factorial would be placed in the global variable GLB, here to here from the gadgets GAN9 present in this location to the gadget GA8

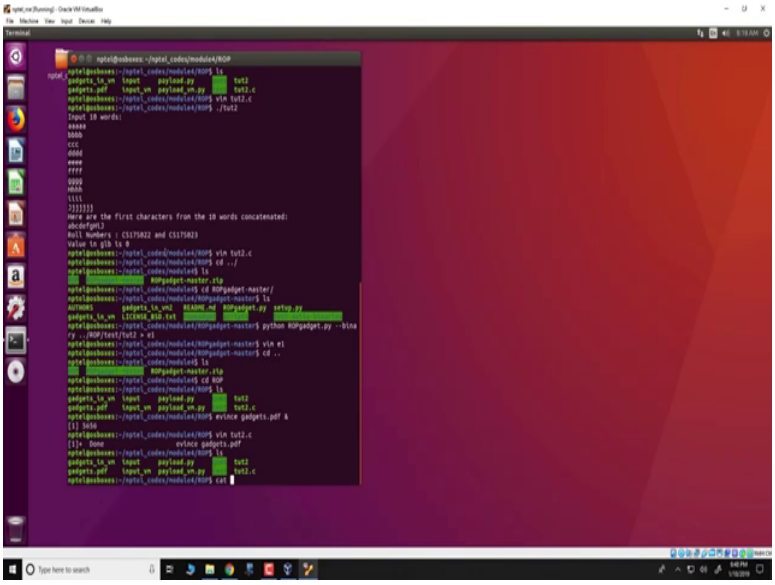
over here we will compute 4factorial, so this is evident from this move EX to 4 and then to the contents of GLB, we multiply 4 and therefore after a few more gadgets we would have stored a 4factorial in GLB, now this is repeated several times in a sequence are like 5, 6 and so on until the entire 10 factorial is computed, the one hurdle which one would probably face while trying to go this is that the stack segment may actually overflow for instant because this is from the second function which is invoked soon after main.

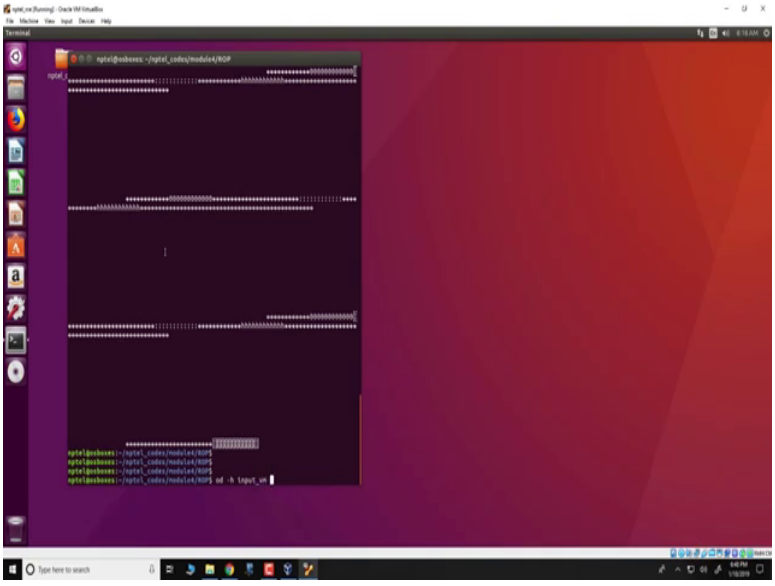
Therefore, the content of the stack is relatively small and if we keep adding gadgets onto this stack, the stack may quite easily overflow and thing not work, so it is quite tricky way to choose gadgets and place gadgets in such a way so that the require payload gets executed.

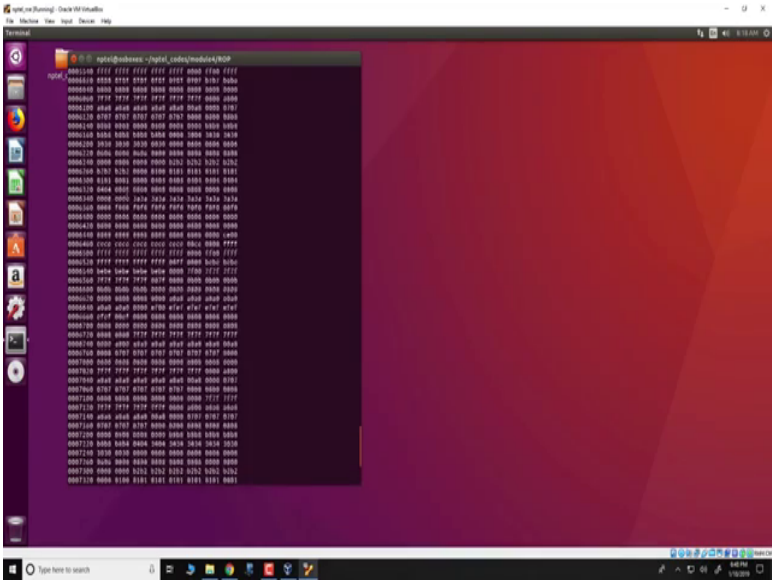
(Refer Slide Time: 11:52)

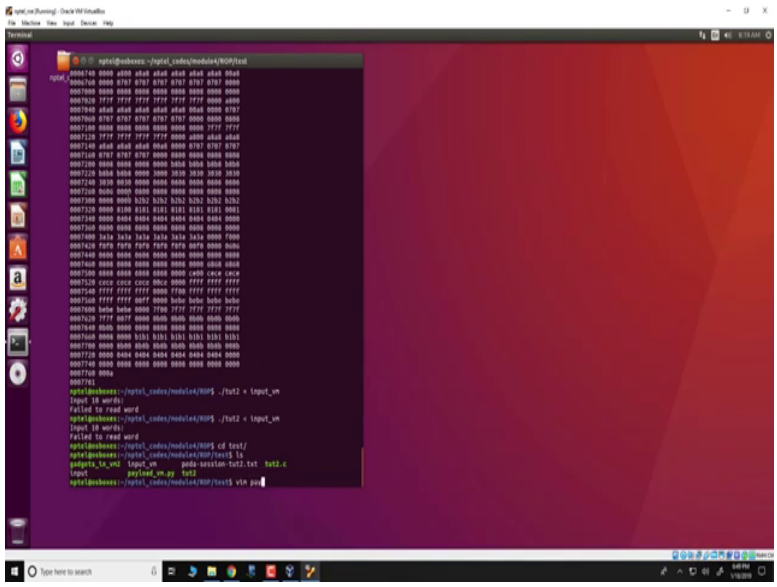


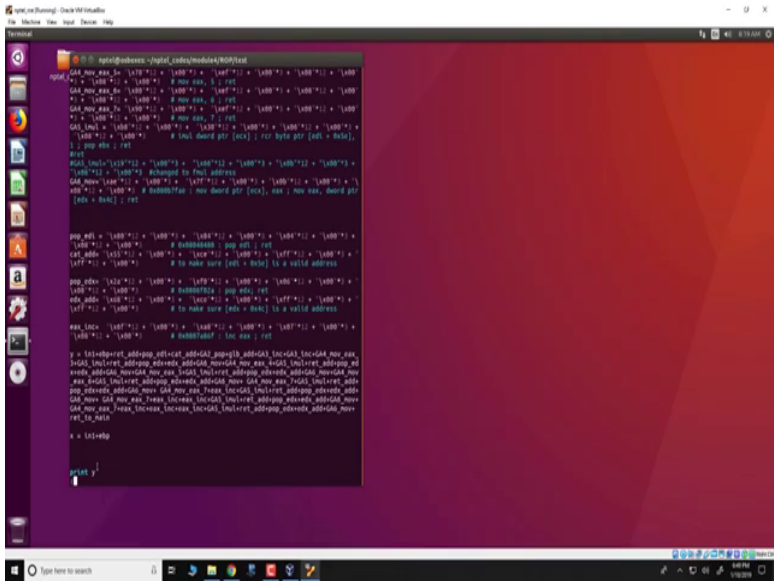



nptel@osboxes: ~/nptel_codes/module4/ROP
Input 10 words:
Here are the first characters from the 10 words concatenated:
abcdefghIJ
Roll Numbers : CS175022 and CS175023
Value to glb is 0
Type here to search


nptel@osboxes: ~/nptel_codes/module4/ROP
Type here to search

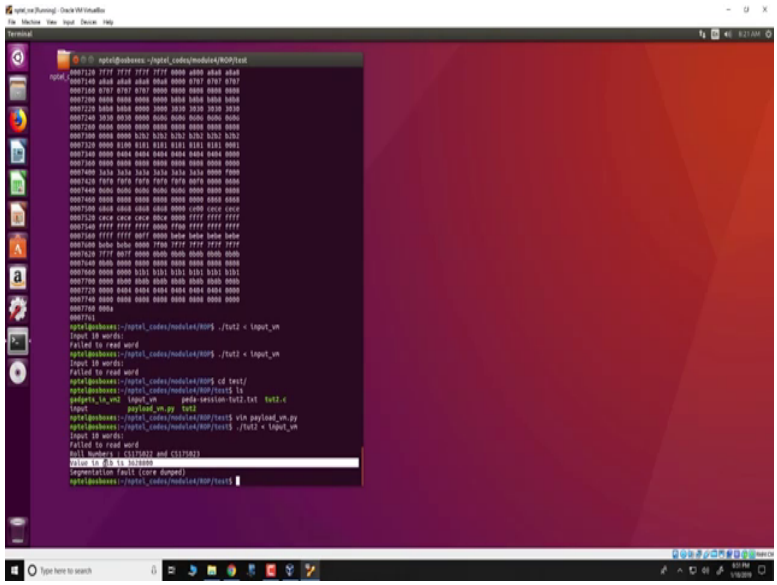

nptel@osboxes: ~/nptel_codes/module4/ROP
Type here to search





So let us see how this thing works, will not go into details about the internals of this because it is extremely complex and essentially out of scope for this particular course but people just run this and the input is, the input where the gadgets are placed is in this file, import VM, so it is a binary file, so we could open it with octal dumb profile and this is how, this is what the input is actually used to the tutorial program, so let us see how this ROP gadgets actually executes, so there is a payload_VM.PY, will just have a look at that and see that, this particular code arranges the gadgets and forms the required input which is placed in Y, in the string Y over here and finally Y gets printed.

(Refer Slide Time: 13:02)



So this particular python strip or you could actually go through it to see how these gadgets are arranged and eventually the file that gets created is, this file called input._VM, so we run this tutorial 2, give it this malicious input, which is input_VM and we see that the value in GLB is 3628800, this essentially is the value for 10 factorial are which you can actually verify, so one problem in this code is that, there is a segmentation fault and after the 10 factorial gets printed and this is because the program was not terminated in a safe way or with a nice exit, so I would leave an assignment or a very hard challenge, one of the most difficult challenges in this course is to modify this particular code in such a way that the ROP gadget terminates in a grace hole way. Thank you.