Assignment - 3
(Return-Oriented-Programming)

Name - Vasu A. Bhatt

Roll No - CS23M072

Secure Systems Engineering
(CS6570)

Table of Contents

1.Gadgets used in Q_1(Finding 73*21)

GA#	GADGETS USED	DESCRIPTION
GA1	pop ecx ; ret	To fill value 0xf in ecx(%ecx = 15).
GA2	pop edx ; xor eax,eax ; pop edi ; ret	To fill value 0x5 in edx(%edx = 5).
GA3	sub edx,ecx ; lea 0x4000(%ecx,%eax,1),%eax ; ret	ecx = ecx - edx(%ecx = 10).
GA4	pop ebx ; ret	To fill value 0x49 in eax(%ebx = 73).
GA5	pop eax ; ret	To fill value 0x15 in eax(%eax = 21).
GA6	imul ebx,eax ; add 0xa ,eax ; ret	eax = eax*ebx ; eax = eax+10. This extra 10 is dealt by GA7.
GA7	sub ecx, eax ; ret	eax = eax-ecx. To deal with the extra 10 added by GA6.

2.Gadgets Addresses(Finding 73*21)

GA#	GADGETS USED	ADDRESS
GA1	pop ecx ; ret	<0x080915f3>
GA2	pop edx ; xor eax,eax ; pop edi ; ret	<0x080b1145>
GA3	sub edx,ecx ; lea 0x4000(%ecx,%eax,1),%eax ; ret	<0x080c1952>
GA4	pop ebx ; ret	<0x0806bf1d>
GA5	pop eax ; ret	<0x080cf49a>
GA6	imul ebx,eax ; add 0xa ,eax ; ret	<0x08049765>
GA7	sub ecx, eax ; ret	<0x0804977c>

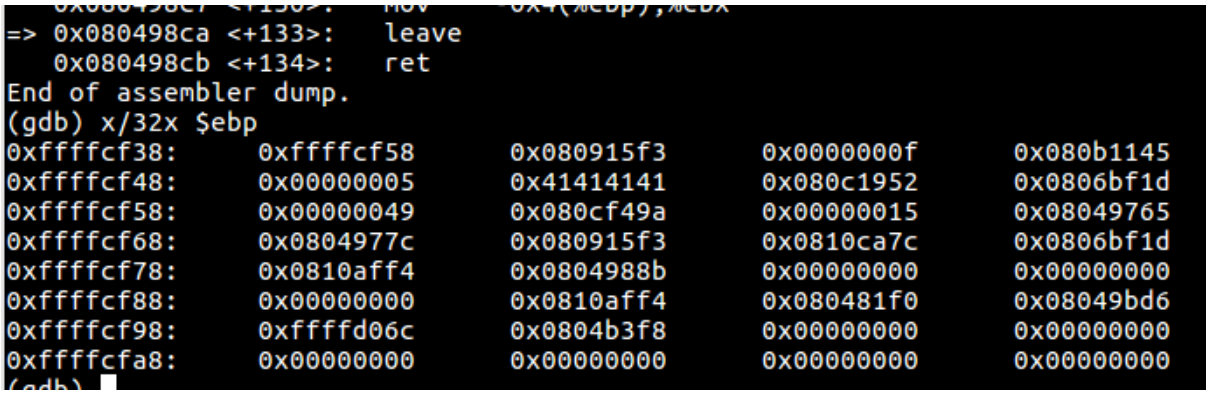
3.ROP Chain Explanation(Finding 73*21)

❖ Stack Diagram:

Address	Content
<0xffffcf7c>	<0x0804988b>(Returns backs to main())
. .	(Restoration of %ebx and %ecx to their original value)
<0xffffcf68>	GA7
<0xffffcf64>	GA6
<0xffffcf60>	0x15(Value 21)
<0xffffcf5c>	GA5
<0xffffcf58>	0x49(Value 73)
<0xffffcf54>	GA4

<0xffffcf50>	GA3
<0xffffcf4c>	‘A’*4
<0xffffcf48>	0x05(Value 5)
<0xffffcf44>	GA2
<0xffffcf40>	0x0f(Value 15)
<0xffffcf3c>	GA1
<0xffffcf38>	Old_ebp_value(0xffffcf58)

❖ GDB picture:



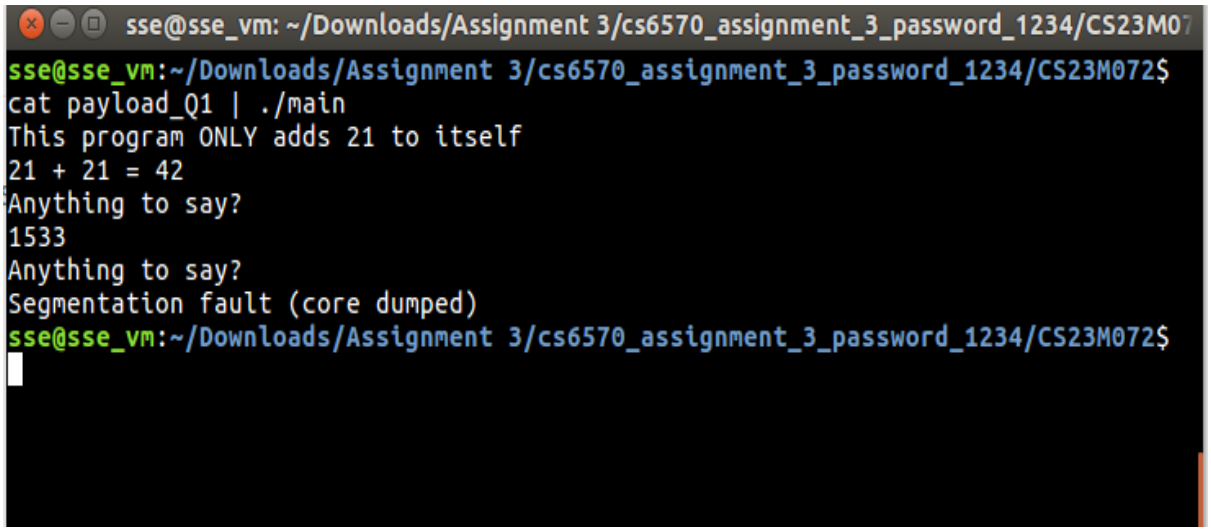
Above image shows the stack content after the payload_Q1 is entered.

❖ Explanation:

- First push ‘A’*36, to reach where old_ebp_val is stored.
- Now push its original value in the payload string to avoid any complication along the way.
- Now the Idea is to store value 73 in %ebx and value 21 in %eax then multiply both register and store back the result in %eax.
- To store value in %ebx GA4 is used and in %eax,GA5 is used.
- To multiply both the registers GA6 is used.
- The problem is GA6 adds extra 0xa(value 10) to %eax which needs to be dealt with by some other gadgets.
- We can’t directly store value 10(0xa) in any register since ‘scanf()’ takes it as the EOL and stops reading any input after it.
- Hence we store the first value 15(0xf) in %ecx using GA1 and value 5(0x5) in %edx using GA2.

- Then we subtract %edx from %ecx using GA3 to get value 10(0xa) in %ecx.
- Then after the multiplication we need to subtract %ecx from %eax, which is done by GA7.
- Lastly,we jump back to the original ‘main()’ function inorder to print the %eax.
- On the safer side we restore back the values of %ecx and %ebx registers, to avoid any complication during the execution.

4.Output Q-1


sse@sse_vm: ~/Downloads/Assignment 3/cs6570_assignment_3_password_1234/CS23M07
sse@sse_vm:~/Downloads/Assignment 3/cs6570_assignment_3_password_1234/CS23M072$
cat payload_Q1 | ./main
This program ONLY adds 21 to itself
21 + 21 = 42
Anything to say?
1533
Anything to say?
Segmentation fault (core dumped)
sse@sse_vm:~/Downloads/Assignment 3/cs6570_assignment_3_password_1234/CS23M072$

5.Gadgets used in Q_2(Finding the 7!)

GA#	GADGETS USED	DESCRIPTION
GA1	pop ecx ; ret	To fill value 0x5 in ecx(%ecx = 5)
GA2	add ecx,ecx ; ret	To get value 0xa in ecx to deal with the extra value 10 that the GA5 adds in eax.(%ecx = 10)
GA3	pop ebx ; ret	To fill value 0x2 in ebx(%ebx = 2)
GA4	pop eax ; ret	To fill value 0x1 in eax(%eax = 1)
GA5	imul ebx,eax ; add 0xa ,eax ; ret	eax = eax*ebx ; eax = eax+10. This extra 10 is dealt by GA6.
GA6	sub ecx, eax ; ret	eax = eax-ecx. To deal with the extra 10 added by GA5.
GA7	inc ebx ; ret	ebx = ebx+1.

6.Gadgets Addresses(Finding the 7!)

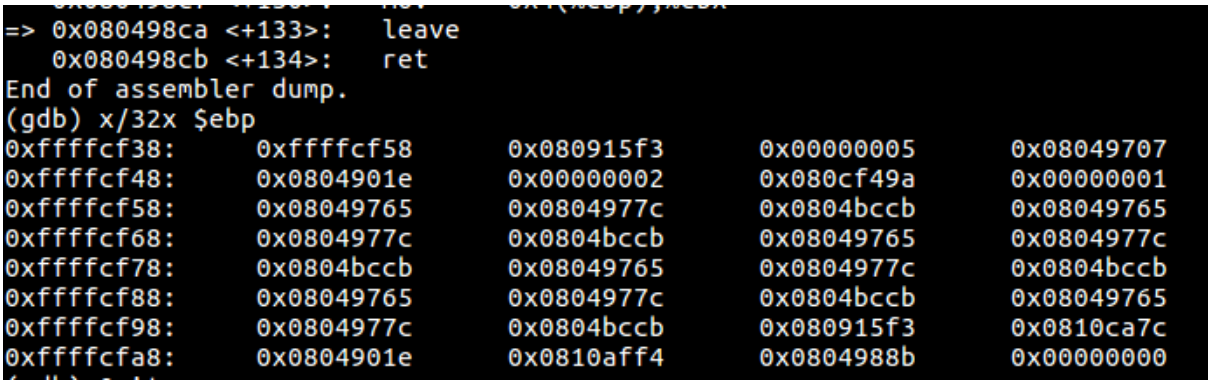
GA#	GADGETS USED	ADDRESS
GA1	pop ecx ; ret	<0x080915f3>
GA2	add ecx,ecx ; ret	<0x08049707>
GA3	pop ebx ; ret	<0x0804901e>
GA4	pop eax ; ret	<0x080cf49a>
GA5	imul ebx,eax ; add 0xa ,eax ; ret	<0x08049765>
GA6	sub ecx, eax ; ret	<0x0804977c>
GA7	inc ebx ; ret	<0x0804bccb>

7.ROP Chain Explanation(Finding the 7!)

❖ Stack Diagram:

Address	Content
<0xffffcf9c>	<0x0804988b>(Returns backs to main())
. . . .	. (Repeating GA5,GA6,GA7 for 5 more times and restoration of %ecx,%ebx) . .
<0xffffcf60>	GA7
<0xffffcf5c>	GA6
<0xffffcf58>	GA5
<0xffffcf54>	0x01(Value 1)
<0xffffcf50>	GA4
<0xffffcf4c>	0x02(Value 2)
<0xffffcf48>	GA3
<0xffffcf44>	GA2
<0xffffcf40>	0x05(Value 5)
<0xffffcf3c>	GA1
<0xffffcf38>	Old_ebp_value(0xffffcf58)

- GDB picture:

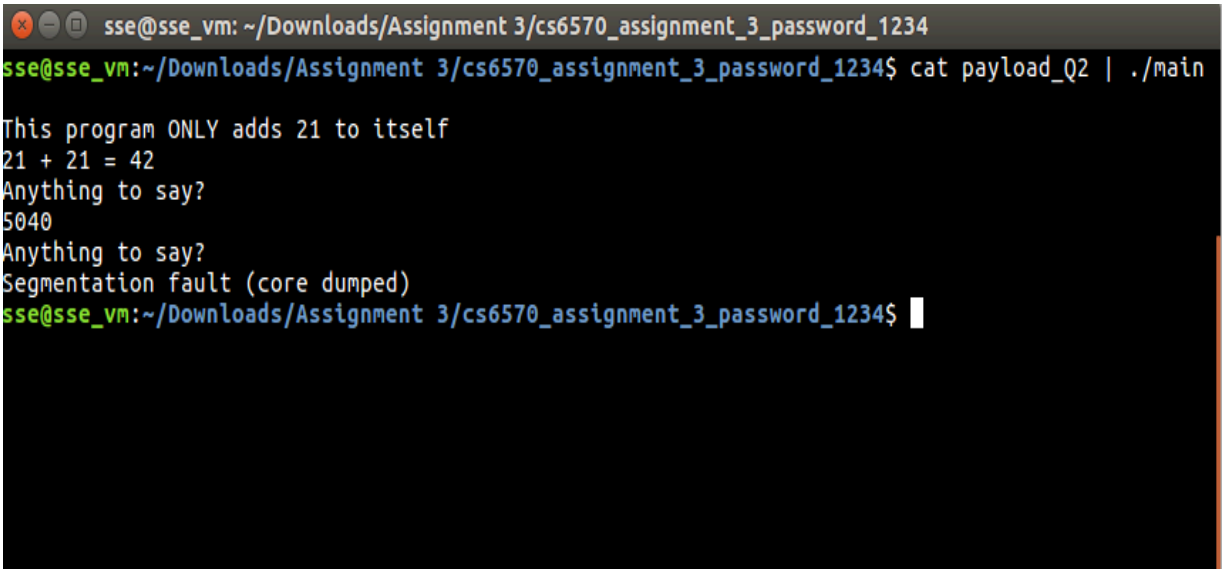


Above image shows the stack content after the payload_Q2 is entered.

❖ Explanation:

- First push ‘A’*36, to reach where old_ebp_val is stored.
- Now push its original value in the payload string to avoid any complication along the way.
- Now the Idea is to store value 2 in %ebx and value 1 in %eax then multiply both register and store back the result in %eax,then increment the value in %ebx(becomes 3 now) and repeat the multiplication till we get 7! In %eax.
- To store value in %ebx GA3 is used and in %eax,GA4 is used.
- To multiply the both register GA5 is used, and to increment %ebx GA7 is used.
- The problem is GA5 adds extra 0xa to %eax which needs to be dealt with by some other gadgets.
- We can’t directly store value 10(0xa) in any register since ‘scanf()’ takes it as the EOL and stops reading any input after it.
- Hence we store the first value 5(0x5) in %ecx using GA1 and then add %ecx with itself using GA2.
- Then after every multiplication above we need to subtract %ecx from %eax, which is done by GA6.
- Lastly,once %eax has the value 5040(7!), we jump back to the original ‘main()’ function inorder to print the %eax.
- On the safer side we restore back the values of %ecx and %ebx registers, to avoid any complication during the execution.

8.Output Q-2


sse@sse_vm: ~/Downloads/Assignment 3/cs6570_assignment_3_password_1234
sse@sse_vm:~/Downloads/Assignment 3/cs6570_assignment_3_password_1234$ cat payload_Q2 | ./main
This program ONLY adds 21 to itself
21 + 21 = 42
Anything to say?
5040
Anything to say?
Segmentation fault (core dumped)
sse@sse_vm:~/Downloads/Assignment 3/cs6570_assignment_3_password_1234$

9.Gadgets used in Q_3(Finding ‘nth’ fibonacci number)

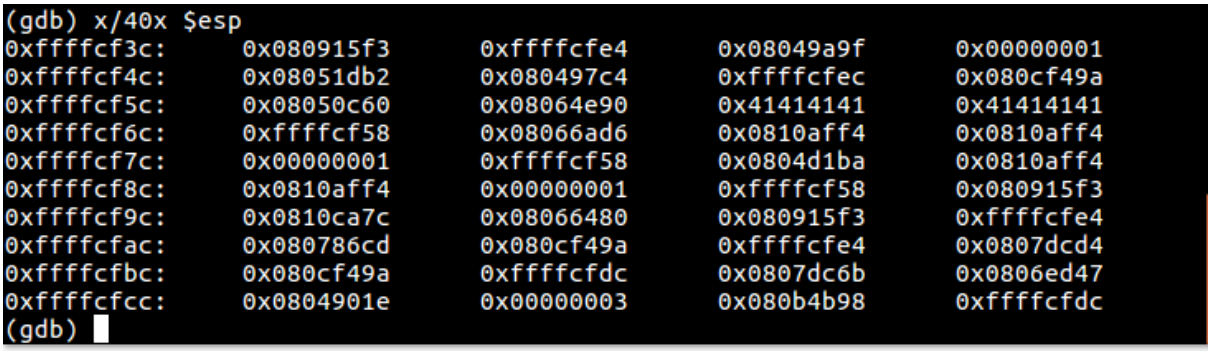
GA#	GADGETS USED	ADDRESS
GA1	pop ecx ; ret	<0x080915f3>
GA2	pop ebx ; ret	<0x0806bf1d>
GA3	pop eax ; ret	<0x080cf49a>
GA4	add dword ptr [ecx], edi ; ret	<0x08051db2>
GA5	mov edx, edi ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret	<0x08066ad6>
GA6	mov eax, ecx ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret	<0x0804d1ba>
GA7	mov eax, dword ptr [eax] ; ret	<0x08066480>
GA8	mov edi, eax ; mov esi, edx ; mov eax, dword ptr [esp + 4] ; ret	<0x080786cd>
GA9	mov dword ptr [eax], edx ; ret	<0x0807dcd4>
GA10	mov (eax),edx ; mov edx,eax ; ret	<0x0807dc6b>
GA11	inc edx ; ret	<0x0806ed47>
GA12	mov eax, ebx ; pop ebx ; ret	<0x080b4b98>
GA13	sub eax, edx ; ret	<0x0806617c>
GA14	mov edx,(eax) ; ret	<0x0807dcd4>
GA15	pop edx ; mov eax, 0x16 ; pop ebx ; pop esi ; ret	<0x0806d968>
GA16	cmovne %edx,%eax ; ret	<0x0806dc00>
GA17	pop esi ; ret	<0x080497c4>

GA18	mov dword ptr [esi], eax ; add esp, 4 ; pop ebx ; pop esi ; ret	<0x08064e90>
GA19	pop esp ; add byte ptr [eax], al ; add byte ptr [eax], al ; pop ebx ; pop esi ; ret	<0x080657a6>

Notes :

- Instead of explaining all the gadgets separately above,below is the working of the script which explains how the above ROP chain of gadgets is working.
- Please make note that I'm considering the starting of the fibonacci sequence from 0th index, meaning 0th fib = 1, 1st fib = 1 , 2nd fib = 2 so on and so forth.
- No loop inside the script is used hence it can find answers for very large ‘n’ values without stack getting overflowed,but make sure not to give input ‘n = 10 ‘ since the scanf() inside ./main considers it as EOL and stops reading anything after it.
- For generating python script, python version 3 is used and the way to use it is also mentioned in section no.10 to generate different payloads to generate different ‘n’.

❖ GDB picture:



Above image shows the stack content after the Question_3 payload is entered.

❖ Explanation:

- %ecx stores the memory address where the final result is going to be stored which initially contains value 1 inside stack and %edi stores the value 1, which is where our initial two sequences of fibonacci starts using gad_1 and gad_2.
- Add the value of %edi and the value stored at the address stored in %ecx and store it back there using gad_3.
- Now copy the value of %edi in %edx and store the value of result in %eax using gad_5 and gad_6.
- Now store the value of %eax in %edi and %edx in the memory location where %ecx is pointing to using gad_7 and gad_8, essentially we did a swap of values in %edi and the address pointed by the %ecx.
- Now take the value of the counter in %edx, and increment it using gad_10 and gad_11.
- Store that modified value of the counter in the memory reserved to hold it using gad_12.
- Now take the value ‘n’ in %eax, and subtract %eax-%edx using gad_13.
- This result indicates how many iterations are left inorder to find the ‘nth’ fibonacci number.
- This also sets the flag for the conditional mov used in the ROP chain.
- Store the return address(printf() add) in %eax and gad_8’s address in %edx using gad_4 and gad_15.
- Now the conditional move happens,only if the counter value is not equal to ‘n’,indicating there are still iteration left and can’t return back to printf() now so hold the gad_8 address in %eax, otherwise the conditional move won’t happen and %eax will hold

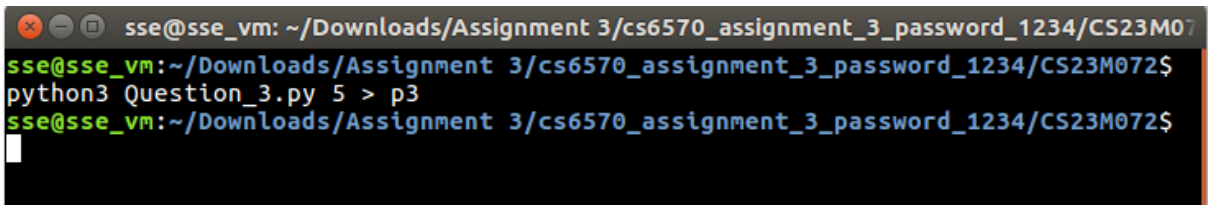
the address of printf(), if we have reached at the final iteration using gad_16.

- This address which %eax is holding will be copied onto to the stack which comes in a path of our ROP chain execution, using the gad_17 and gad_18.
- Now to iterate in a loop, we pop %esp using the gadget_19 which ultimately sets the stack pointer to the starting of our stack ROP chain just before the gad_4, which is working as a loop maker.
- In this fashion our ROP chain will iterate till the counter becomes equal to ‘n’ and when that happens we come out of the loop naturally and print the value of ‘n’ the fibonacci number which is stored in %eax.

10.Output Q-3

- To generate payload for ‘n’ the fibonacci number give following command in terminal

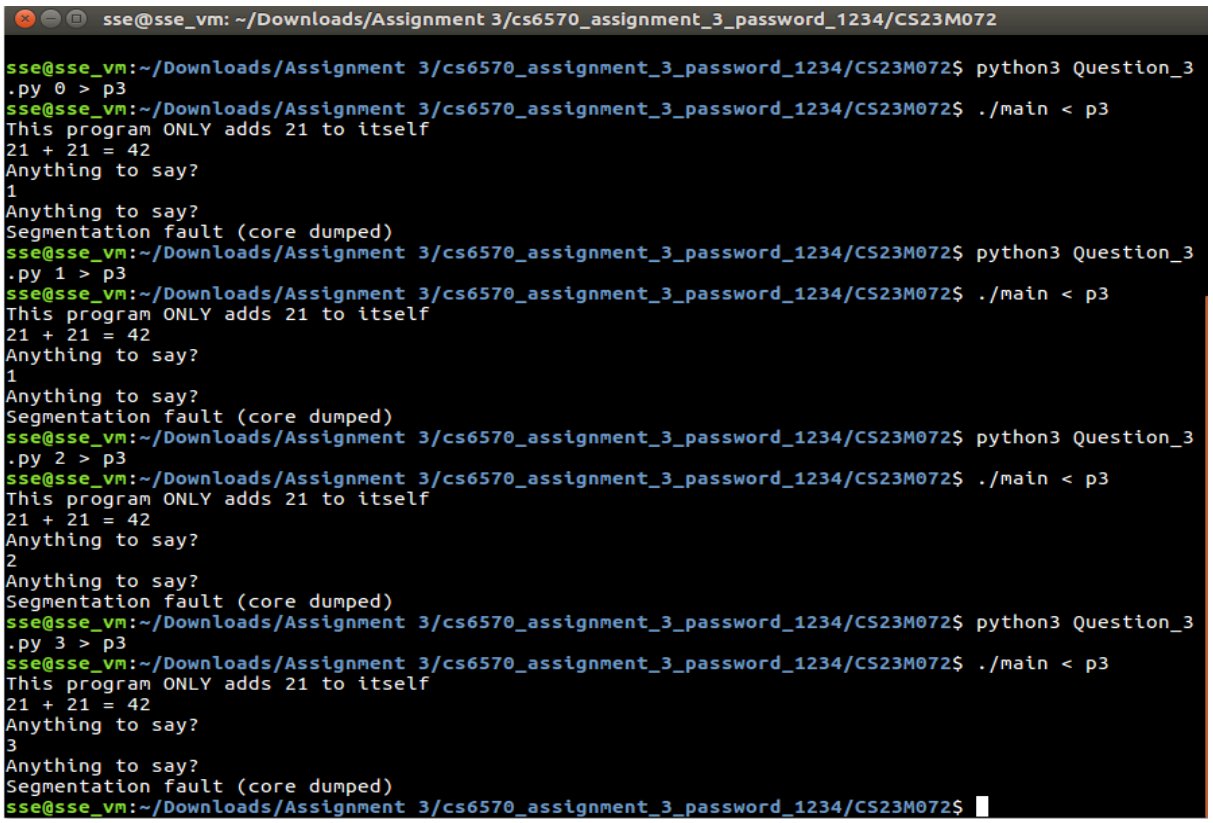
  “python3 Question_3.py n > name_of_payload”



In above image n is taken as 5 and the name of the payload generated is p3.

- To give that dynamically generated payload as an input to ./main file use following command

  “./main < name_of_payload”



Above image shows proper working of script with payload to generate any ‘nth’ fibonacci number.