Information Security - 5 - Secure Systems Engineering
Professor Chester Rebeiro
Indian Institute of Technology, Madras
Heap_Demo2

(Refer Slide Time: 0:27)

Hello and welcome to this demonstration in the course for secure systems engineering. So we will be looking at another demonstration (of) for how the heap works, we will take a small program you could actually download this program from your VirtualBox and run this program preferably from your VirtualBox the program is called t1 dot C and it is a very small program, so what it does is that we define two pointers x and y both are character pointers, we first malloc 15 bytes for x and then we free x, then later we also malloc 13 bytes for y and then we free y, this is a very simple program and let us see how it actually executes.

So what we are actually printing in these two printf statements is the address of x and y, as before we will make clean and make it and run the program and what you see over here surprisingly is that both x and y get the same address. So what this means is that the data corresponding to x can be accessed by y and vice versa the reason this happens that x and y get the same address is that internally pt malloc manages all the free chunks of data in list.

So therefore, when we malloc x of 15 bytes we have a chunk of data in the heap corresponding to x, when x gets freed this malloc chunk gets a part of a linked list. Since this is the first malloc that is done in the program therefore in this particular point in time the list has just one chunk that is available and that chunk is x. Now when malloc is invoked again pt

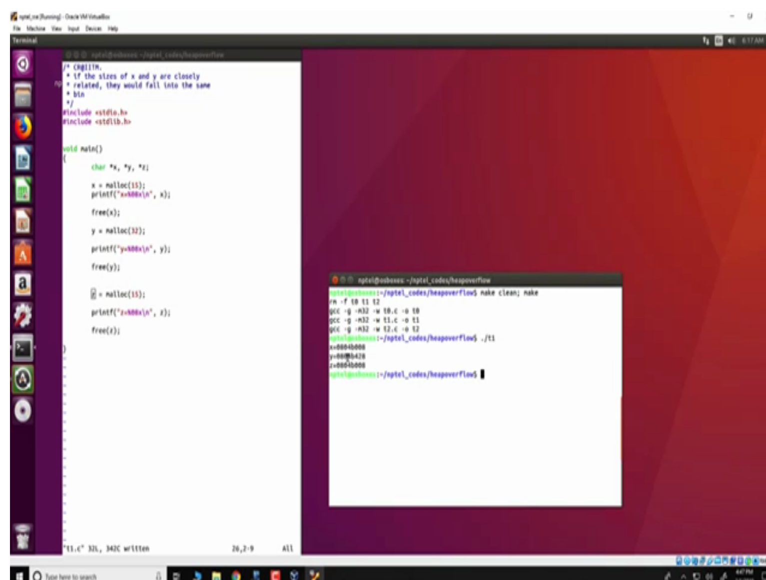
malloc would first look into these free list and identify if there is a chunk which can satisfy this particular request.

Now in this particular case we have a list and in that list we have one chunk that is present corresponding to the freed x chunk and therefore this chunk gets allocated to y, therefore what would happen over here is that y and x would obtain the same chunk of memory and therefore would have the same address and this is the reason why x and y would have the same address.

Now the result of course would vary depending on the size of the mallocs that was requested. Now in this particular case 15 and 13 are relatively close, so they fall within the same free list. If for example we would have allocated something which is much larger, let us say 32 bytes like this then malloc will not be able to use the will not be able to reuse the freed chunk of memory which was just freed from x and would have to allocate something bigger and therefore if we have 15 followed by 32 which was requested by the mallocs x and y would get different chunks.

So we can verify this again by something like this and what we see over here is that x and y are of different addresses meaning that they have been allocated to different chunks of memory. Similarly if we would have had a third pointer defined and allocated it and requested for just let us say 15 bytes again, we would see that (x) x and z would get the same chunk of memory and would have the same address while y would have a different address.

(Refer Slide Time: 4:39)



So we could just run it like this, (small typo here) so this should have been z yeah in fact what we see is x and z have got the same address because they have been z has been allocated the chunk which has been freed by x because it was the best fit for x, while y still gets a chunk which was different from that of x, thank you.