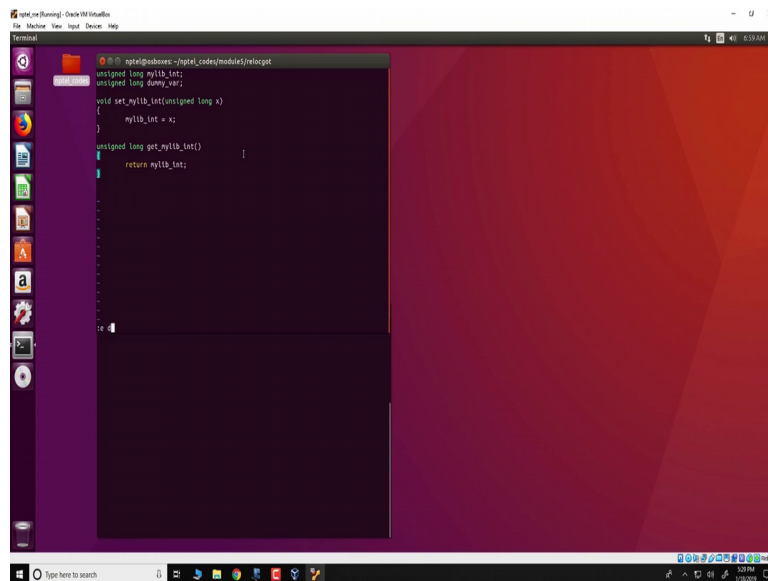


Information Security 5 Secure System Engineering
Professor Chester Rebeiro
Indian Institute of Technology Madras
Demonstration of Position Independent Code
Mod03_Lec19

Hello and welcome to this demonstration in the course for secure system engineering, in this demo will be actually looking at an example of how position independent code works, so in the previous demonstration we looked at runtime relocatable code and we seen that it requires that the loader have a lot of thing to do and it requires the loader to go and modify the code and fix the address in each of the locations and a lot of this becomes much more simplified with a PIC, a position independent code, so we will use exactly the same code as we have done the previous demonstration, this code can be downloaded in the virtual machine that comes along with this course.

(Refer Slide Time: 1:05)



```
nppt@nppt:~/code/modules/relocgot$ cat test.c
unsigned long mylib_test;
unsigned long mylib_test;

void set_mylib_test(unsigned long x)
{
    mylib_test = x;
}

unsigned long get_mylib_test()
{
    return mylib_test;
}
```

```

nptel@osboxes:~/nptel_codes/modules/relocgot
$ cat driver.c
extern void set_mylib_int(unsigned long x);
extern long get_mylib_int();
extern unsigned long mylib_int;

unsigned long glob = 5555;

int main()
{
    set_mylib_int(109);
    printf("value set in mylib is %ld\n", get_mylib_int());
    printf("value set in glob is %ld\n", glob);
}

"driver.c" 14L, 292C      12.1-8      All

```

```

nptel@osboxes:~/nptel_codes/modules/relocgot
$ make
driver.o libmylib.so makefile mylib.o README.md
driver.c libmylib.so.dSYM mylib.c README.get.txt README/reloc.txt
nptel@osboxes:~/nptel_codes/modules/relocgot$ make lib_pic
gcc -m32 -fPIC -c mylib.c -o mylib.o
gcc -m32 -fPIC -shared -o libmylib_pic.so mylib.o
objdump --disassemble-all libmylib_pic.so > libmylib_pic.so.dSYM
nptel@osboxes:~/nptel_codes/modules/relocgot$

```

And this particular example is nptel_codes module 5 relocgot, so we use exactly the same library as before, the library is called mylib.c and which is as shown over here it has mylib_int, setmylib_int and getmylib_int at to set and get functions and the driver for this as before is here which has an invocation to setmylib_int and getmylib_int, in order to generate position independent code the command or the compiler options are slightly different, so in order to do this in this example how a makefile would require makelib_pic which would generate the library as a position independent code library or a pic library.

So as you see here we have in additional option minus F pic which would generate, which would cost the compiler to generate a position independent code library and as before we also dump this assembly of this library in libmylib_pic.so.dSYM, so the first thing you notice when

the open this disassembly is that the code looks very different compared to the one we seen previously that is with the load time relocation.

(Refer Slide Time: 2:44)

```
Terminal
Disassembly of section .gnu.hash:
00000138 <.gnu.hash>:
138: 00 00      add  (%eax),%eax
139: 00 00      add  %al,(%eax)
13a: 50        push %eax
13b: 00 00      add  %al,(%eax)
13c: 00 00      add  %al,(%eax)
13d: 00 00      add  %al,(%eax)
13e: 00 00      add  %al,(%eax)
13f: 00 00      add  %al,(%eax)
140: 00 00      add  %al,(%eax)
141: 00 00      add  %al,(%eax)
142: 00 00      add  %al,(%eax)
143: 00 00      add  %al,(%eax)
144: 00 00      add  %al,(%eax)
145: 00 00      add  %al,(%eax)
146: 00 00      add  %al,(%eax)
147: 00 00      add  %al,(%eax)
148: 00 00      add  %al,(%eax)
149: 00 00      add  %al,(%eax)
14a: 00 00      add  %al,(%eax)
14b: 00 00      add  %al,(%eax)
14c: 00 00      add  %al,(%eax)
14d: 00 00      add  %al,(%eax)
14e: 00 00      add  %al,(%eax)
14f: 00 00      add  %al,(%eax)
150: 00 00      add  %al,(%eax)
151: 00 00      add  %al,(%eax)
152: 00 00      add  %al,(%eax)
153: 00 00      add  %al,(%eax)
154: 00 00      add  %al,(%eax)
155: 00 00      add  %al,(%eax)
156: 00 00      add  %al,(%eax)
157: 00 00      add  %al,(%eax)
158: 00 00      add  %al,(%eax)
159: 00 00      add  %al,(%eax)
15a: 00 00      add  %al,(%eax)
15b: 00 00      add  %al,(%eax)
15c: 00 00      add  %al,(%eax)
15d: 00 00      add  %al,(%eax)
15e: 00 00      add  %al,(%eax)
15f: 00 00      add  %al,(%eax)
160: 00 00      add  %al,(%eax)
161: 00 00      add  %al,(%eax)
162: 00 00      add  %al,(%eax)
163: 00 00      add  %al,(%eax)
164: 00 00      add  %al,(%eax)
165: 00 00      add  %al,(%eax)
166: 00 00      add  %al,(%eax)
167: 00 00      add  %al,(%eax)
168: 00 00      add  %al,(%eax)
169: 00 00      add  %al,(%eax)
16a: 00 00      add  %al,(%eax)
16b: 00 00      add  %al,(%eax)
16c: 00 00      add  %al,(%eax)
16d: 00 00      add  %al,(%eax)
16e: 00 00      add  %al,(%eax)
16f: 00 00      add  %al,(%eax)
170: 00 00      add  %al,(%eax)
171: 00 00      add  %al,(%eax)
172: 00 00      add  %al,(%eax)
173: 00 00      add  %al,(%eax)
174: 00 00      add  %al,(%eax)
175: 00 00      add  %al,(%eax)
176: 00 00      add  %al,(%eax)
177: 00 00      add  %al,(%eax)
178: 00 00      add  %al,(%eax)
179: 00 00      add  %al,(%eax)
17a: 00 00      add  %al,(%eax)
17b: 00 00      add  %al,(%eax)
17c: 00 00      add  %al,(%eax)
17d: 00 00      add  %al,(%eax)
17e: 00 00      add  %al,(%eax)
17f: 00 00      add  %al,(%eax)
180: 00 00      add  %al,(%eax)
181: 00 00      add  %al,(%eax)
182: 00 00      add  %al,(%eax)
183: 00 00      add  %al,(%eax)
184: 00 00      add  %al,(%eax)
185: 00 00      add  %al,(%eax)
186: 00 00      add  %al,(%eax)
187: 00 00      add  %al,(%eax)
188: 00 00      add  %al,(%eax)
189: 00 00      add  %al,(%eax)
18a: 00 00      add  %al,(%eax)
18b: 00 00      add  %al,(%eax)
18c: 00 00      add  %al,(%eax)
18d: 00 00      add  %al,(%eax)
18e: 00 00      add  %al,(%eax)
18f: 00 00      add  %al,(%eax)
190: 00 00      add  %al,(%eax)
191: 00 00      add  %al,(%eax)
192: 00 00      add  %al,(%eax)
193: 00 00      add  %al,(%eax)
194: 00 00      add  %al,(%eax)
195: 00 00      add  %al,(%eax)
196: 00 00      add  %al,(%eax)
197: 00 00      add  %al,(%eax)
198: 00 00      add  %al,(%eax)
199: 00 00      add  %al,(%eax)
19a: 00 00      add  %al,(%eax)
19b: 00 00      add  %al,(%eax)
19c: 00 00      add  %al,(%eax)
19d: 00 00      add  %al,(%eax)
19e: 00 00      add  %al,(%eax)
19f: 00 00      add  %al,(%eax)
1a0: 00 00      add  %al,(%eax)
1a1: 00 00      add  %al,(%eax)
1a2: 00 00      add  %al,(%eax)
1a3: 00 00      add  %al,(%eax)
1a4: 00 00      add  %al,(%eax)
1a5: 00 00      add  %al,(%eax)
1a6: 00 00      add  %al,(%eax)
1a7: 00 00      add  %al,(%eax)
1a8: 00 00      add  %al,(%eax)
1a9: 00 00      add  %al,(%eax)
1aa: 00 00      add  %al,(%eax)
1ab: 00 00      add  %al,(%eax)
1ac: 00 00      add  %al,(%eax)
1ad: 00 00      add  %al,(%eax)
1ae: 00 00      add  %al,(%eax)
1af: 00 00      add  %al,(%eax)
1b0: 00 00      add  %al,(%eax)
1b1: 00 00      add  %al,(%eax)
1b2: 00 00      add  %al,(%eax)
1b3: 00 00      add  %al,(%eax)
1b4: 00 00      add  %al,(%eax)
1b5: 00 00      add  %al,(%eax)
1b6: 00 00      add  %al,(%eax)
1b7: 00 00      add  %al,(%eax)
1b8: 00 00      add  %al,(%eax)
1b9: 00 00      add  %al,(%eax)
1ba: 00 00      add  %al,(%eax)
1bb: 00 00      add  %al,(%eax)
1bc: 00 00      add  %al,(%eax)
1bd: 00 00      add  %al,(%eax)
1be: 00 00      add  %al,(%eax)
1bf: 00 00      add  %al,(%eax)
1c0: 00 00      add  %al,(%eax)
1c1: 00 00      add  %al,(%eax)
1c2: 00 00      add  %al,(%eax)
1c3: 00 00      add  %al,(%eax)
1c4: 00 00      add  %al,(%eax)
1c5: 00 00      add  %al,(%eax)
1c6: 00 00      add  %al,(%eax)
1c7: 00 00      add  %al,(%eax)
1c8: 00 00      add  %al,(%eax)
1c9: 00 00      add  %al,(%eax)
1ca: 00 00      add  %al,(%eax)
1cb: 00 00      add  %al,(%eax)
1cc: 00 00      add  %al,(%eax)
1cd: 00 00      add  %al,(%eax)
1ce: 00 00      add  %al,(%eax)
1cf: 00 00      add  %al,(%eax)
1d0: 00 00      add  %al,(%eax)
1d1: 00 00      add  %al,(%eax)
1d2: 00 00      add  %al,(%eax)
1d3: 00 00      add  %al,(%eax)
1d4: 00 00      add  %al,(%eax)
1d5: 00 00      add  %al,(%eax)
1d6: 00 00      add  %al,(%eax)
1d7: 00 00      add  %al,(%eax)
1d8: 00 00      add  %al,(%eax)
1d9: 00 00      add  %al,(%eax)
1da: 00 00      add  %al,(%eax)
1db: 00 00      add  %al,(%eax)
1dc: 00 00      add  %al,(%eax)
1dd: 00 00      add  %al,(%eax)
1de: 00 00      add  %al,(%eax)
1df: 00 00      add  %al,(%eax)
1e0: 00 00      add  %al,(%eax)
1e1: 00 00      add  %al,(%eax)
1e2: 00 00      add  %al,(%eax)
1e3: 00 00      add  %al,(%eax)
1e4: 00 00      add  %al,(%eax)
1e5: 00 00      add  %al,(%eax)
1e6: 00 00      add  %al,(%eax)
1e7: 00 00      add  %al,(%eax)
1e8: 00 00      add  %al,(%eax)
1e9: 00 00      add  %al,(%eax)
1ea: 00 00      add  %al,(%eax)
1eb: 00 00      add  %al,(%eax)
1ec: 00 00      add  %al,(%eax)
1ed: 00 00      add  %al,(%eax)
1ee: 00 00      add  %al,(%eax)
1ef: 00 00      add  %al,(%eax)
1f0: 00 00      add  %al,(%eax)
1f1: 00 00      add  %al,(%eax)
1f2: 00 00      add  %al,(%eax)
1f3: 00 00      add  %al,(%eax)
1f4: 00 00      add  %al,(%eax)
1f5: 00 00      add  %al,(%eax)
1f6: 00 00      add  %al,(%eax)
1f7: 00 00      add  %al,(%eax)
1f8: 00 00      add  %al,(%eax)
1f9: 00 00      add  %al,(%eax)
1fa: 00 00      add  %al,(%eax)
1fb: 00 00      add  %al,(%eax)
1fc: 00 00      add  %al,(%eax)
1fd: 00 00      add  %al,(%eax)
1fe: 00 00      add  %al,(%eax)
1ff: 00 00      add  %al,(%eax)

```

```
Terminal
Disassembly of section .gnu.hash:
520: 55        push %ebp
521: 89 e5     mov  %eax,%ebp
522: 83 ec 14  sub  $0x14,%esp
523: 58        push %eax
524: ff e2     call %edx
525: c9 04 1b  add  $0xc9041b,%esp
526: c9 04 1b  leave $0xc9041b,%esp
527: e9 24 ff ff  jmp  <register_tm_clones>
0000031c <.lib_list__initializers>:
31c: 0b 7a 04  mov  $0xb7a04,%eax
31d: c3        ret
00000340 <__libc_init_once_helper@GLIBC_2.2.5>:
340: 55        push %ebp
341: 89 e5     mov  %eax,%ebp
342: c7 2a 00 00  call $0xc72a0000,%eax
343: 03 1a 00 00  add  $0x031a0000,%eax
344: 0b 7a 04 00  mov  $0xb7a04000,%eax
345: 03 1a 00 00  add  $0x031a0000,%eax
346: 89 38     mov  %eax,%eax
347: 58        pop  %ebp
348: c3        ret
00000350 <__libc_init_once@GLIBC_2.2.5>:
350: 55        push %ebp
351: 89 e5     mov  %eax,%ebp

```

```

nptel@bobboxes:~/nptel_codes/module1/relocgot
520: 55          push  %ebp
521: 89 e5       mov   %esp,%ebp
522: 83 ec 14   sub   $0x14,%esp
523: 50          push  %eax
524: ff c2     call  *%eax
525: 83 c4 10   add   $0x10,%esp
526: c9          leave
527: e9 24 ff ff jmp   400 <register_tm_clones>

0000053c <_x86_get_pc_thunk.do>:
53c: 8b 14 24   mov   (%esp),%edx
53f: c3        ret

00000540 <_set_mylib_int>:
540: 55          push  %ebp
541: 89 e5       mov   %esp,%ebp
542: e8 20 00 00 call  527 <_x86_get_pc_thunk.ax>
543: 50          push  %eax
544: 8b 8c 0c ff mov   -0x14(%eax),%eax
545: 8b 55 08   mov   0x55(%eax),%eax
546: 89 10     mov   %eax,%edx
547: 80        nop
548: 5d        pop   %ebp
549: c3        ret

00000558 <_set_mylib_int>:
558: 55          push  %ebp
55c: 89 e5       mov   %esp,%ebp
55f: c3        ret
478,0-11 478

```

```

nptel@bobboxes:~/nptel_codes/module1/relocgot
libmylib.o: file format elf32-i386
Disassembly of section .note.gnu.build-id:

00000514 .note.gnu.build-id:
104: 04 00     add   $0x0,%eax
106: 00 00     add   $0x0,%eax
108: 14 00     add   $0x14,%eax
109: 00 00     add   $0x0,%eax
110: 03 00     add   $0x3,%eax
111: 00 00     add   $0x0,%eax
112: 47      ltr   %eax
121: 4e      dec   %eax
122: 55      push %ebp
123: 00 2f   add   %eax,%eax
124: 50      pop   %eax
125: b5 1b   mov   %ebx,%eax
126: 29 bc 3c 4e 8a 30 sub   %edi,0x388403(%eax)
12a: 4e      dec   %eax
12f: 3c 44   ds   %eax,%eax
131: b6 f5 2b 0c 4d mov   $0x4d0c2b0f,%eax
/set_mylib

```

```

nptel@bobboxes:~/nptel_codes/module1/relocgot
520: 55          push  %ebp
521: 89 e5       mov   %esp,%ebp
522: 83 ec 14   sub   $0x14,%esp
523: 50          push  %eax
524: ff c2     call  *%eax
525: 83 c4 10   add   $0x10,%esp
526: c9          leave
527: e9 24 ff ff jmp   400 <register_tm_clones>

0000053c <_x86_get_pc_thunk.do>:
53c: 8b 14 24   mov   (%esp),%edx
53f: c3        ret

00000540 <_set_mylib_int>:
540: 55          push  %ebp
541: 89 e5       mov   %esp,%ebp
542: e8 20 00 00 call  527 <_x86_get_pc_thunk.do>
543: 50          push  %eax
544: 8b 8c 0c ff mov   -0x14(%eax),%eax
545: 8b 55 08   mov   0x55(%eax),%eax
546: 89 10     mov   %eax,%edx
547: 80        nop
548: 5d        pop   %ebp
549: c3        ret

00000558 <_set_mylib_int>:
558: 55          push  %ebp
55c: 89 e5       mov   %esp,%ebp
55f: c3        ret
478,53-57 478

```

```

nptel@bobboxes:~/nptel_codes/module1/relocgot
520: 83 ec 14   sub   $0x14,%esp
521: 50          push  %eax
522: ff c2     call  *%eax
523: c9          leave
524: e9 24 ff ff jmp   400 <register_tm_clones>

0000053c <_x86_get_pc_thunk.do>:
53c: 8b 14 24   mov   (%esp),%edx
53f: c3        ret

00000540 <_set_mylib_int>:
540: 55          push  %ebp
541: 89 e5       mov   %esp,%ebp
542: e8 20 00 00 call  527 <_x86_get_pc_thunk.do>
543: 50          push  %eax
544: 8b 8c 0c ff mov   -0x14(%eax),%eax
545: 8b 55 08   mov   0x55(%eax),%eax
546: 89 10     mov   %eax,%edx
547: 80        nop
548: 5d        pop   %ebp
549: c3        ret

00000558 <_set_mylib_int>:
558: 55          push  %ebp
55c: 89 e5       mov   %esp,%ebp
55f: c3        ret
479,11 488

```

Okay, so let us, so what you see over here and if we actually compare this with the previous library code that we obtained in, you see that this and this are completely different, so will go into what exactly happens over here, now the setmylib_int essentially would take an input parameter X and said the global variable mylib_int with that particular value, so this is done very differently a with pic code, so essentially this two instructions creates the stack frame, the next instruction is a call to this function call __X86.get_pc_thunk.ax, so essentially what this function does? This function is something which the compiler adds in.

(Refer Slide Time: 4:15)

```
00000370: 33f: c3                ret
00000371: 540: 55                push  %ebp
00000372: 541: 89 e5             mov   %ebp,%ebp
00000373: 542: e8 2a 00 00 00   call  372..._x86_get_pc_thunk.ass>
00000374: 543: 01 08 1a 00 00   add   $0x1a8,%eax
00000375: 544: 0b 08 ec ff ff   mov   %eax(%eax),%eax
00000376: 545: 0b 08 ec ff ff   mov   %eax(%eax),%eax
00000377: 546: 90                nop
00000378: 547: 5d                pop   %ebp
00000379: 548: c3                ret
0000037a: 550: 55                push  %ebp
0000037b: 551: 89 e5             mov   %ebp,%ebp
0000037c: 552: e8 0f 00 00 00   call  372..._x86_get_pc_thunk.ass>
0000037d: 553: 01 08 1a 00 00   add   $0x1a8,%eax
0000037e: 554: 0b 08 ec ff ff   mov   %eax(%eax),%eax
0000037f: 555: 0b 08 ec ff ff   mov   %eax(%eax),%eax
00000380: 556: 90                nop
00000381: 557: 5d                pop   %ebp
00000382: 558: c3                ret
00000383: 560: 55                push  %ebp
00000384: 561: 89 e5             mov   %ebp,%ebp
00000385: 562: e8 07 00 00 00   call  372..._x86_get_pc_thunk.ass>
00000386: 563: 01 08 1a 00 00   add   $0x1a8,%eax
00000387: 564: 0b 08 ec ff ff   mov   %eax(%eax),%eax
00000388: 565: 0b 08 ec ff ff   mov   %eax(%eax),%eax
00000389: 566: 90                nop
0000038a: 567: 5d                pop   %ebp
0000038b: 568: c3                ret
0000038c: 570: 55                push  %ebp
0000038d: 571: 89 e5             mov   %ebp,%ebp
0000038e: 572: e8 04 24 00 00   call  372..._x86_get_pc_thunk.ass>
0000038f: 573: 01 08 1a 00 00   add   $0x1a8,%eax
00000390: 574: c3                ret
```

```
00000370: 33f: c3                ret
00000371: 540: 55                push  %ebp
00000372: 541: 89 e5             mov   %ebp,%ebp
00000373: 542: e8 2a 00 00 00   call  372..._x86_get_pc_thunk.ass>
00000374: 543: 01 08 1a 00 00   add   $0x1a8,%eax
00000375: 544: 0b 08 ec ff ff   mov   %eax(%eax),%eax
00000376: 545: 0b 08 ec ff ff   mov   %eax(%eax),%eax
00000377: 546: 90                nop
00000378: 547: 5d                pop   %ebp
00000379: 548: c3                ret
0000037a: 550: 55                push  %ebp
0000037b: 551: 89 e5             mov   %ebp,%ebp
0000037c: 552: e8 0f 00 00 00   call  372..._x86_get_pc_thunk.ass>
0000037d: 553: 01 08 1a 00 00   add   $0x1a8,%eax
0000037e: 554: 0b 08 ec ff ff   mov   %eax(%eax),%eax
0000037f: 555: 0b 08 ec ff ff   mov   %eax(%eax),%eax
00000380: 556: 90                nop
00000381: 557: 5d                pop   %ebp
00000382: 558: c3                ret
00000383: 560: 55                push  %ebp
00000384: 561: 89 e5             mov   %ebp,%ebp
00000385: 562: e8 07 00 00 00   call  372..._x86_get_pc_thunk.ass>
00000386: 563: 01 08 1a 00 00   add   $0x1a8,%eax
00000387: 564: 0b 08 ec ff ff   mov   %eax(%eax),%eax
00000388: 565: 0b 08 ec ff ff   mov   %eax(%eax),%eax
00000389: 566: 90                nop
0000038a: 567: 5d                pop   %ebp
0000038b: 568: c3                ret
0000038c: 570: 55                push  %ebp
0000038d: 571: 89 e5             mov   %ebp,%ebp
0000038e: 572: e8 04 24 00 00   call  372..._x86_get_pc_thunk.ass>
0000038f: 573: 01 08 1a 00 00   add   $0x1a8,%eax
00000390: 574: c3                ret
```

This function just as two statements, it moves the stack pointer or the contents of the stack pointer into the EAX register and then returns, so note that when a call is done the return address for this call that is corresponding to, the address corresponding to this instruction is pushed onto the stack and at that time the stack pointer is pointing to that return address, therefore what is moved into the EAX register is the address of the following instruction that is this instruction.

The reason why we do this is that you want to get the address of this particular instruction and this instruction can be relocatable, so therefore we cannot hard core the address but rather find an indirect way to actually get the address of this instruction, next we add a value of 1A B8 to EAX register, so this value gives the address of a table known as a GOT table, so this is

the GOT table, therefore at the end of this instruction the address of the GOT table is move into the EAX register and then at an offset of -14 in the GOT table is where the actual address of mylib_int is stored.

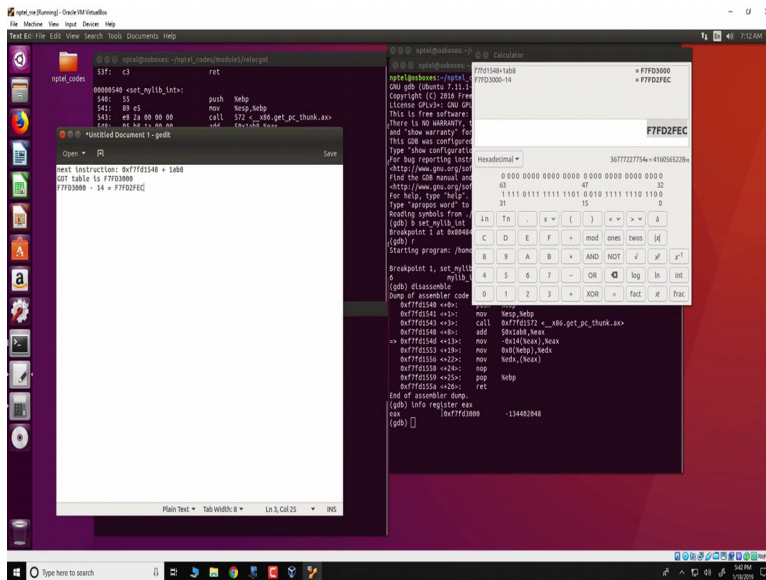
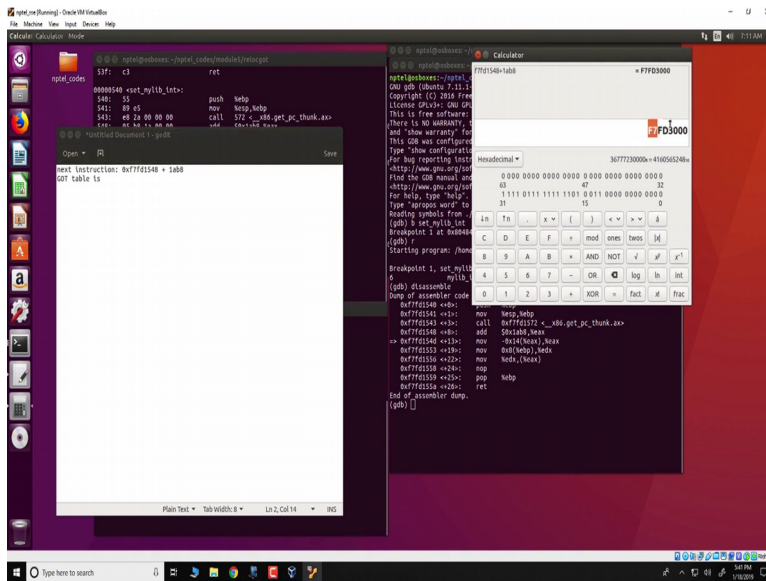
So this address is then moved into the EAX register and the value of X present in the stack at a location 8 bytes from the frame pointer is then moved into the EDX register, then X is stored into mylib_int, so let us see GDB working with this, so we will open this and run GDB, executable for the driver is d pic, so we run d pic as it is, ops as before we need to export the LD library path okay, so as expected we would see in mylib is 100 and the value of log is 5555.

(Refer Slide Time: 7:01)

```
53f: c3                ret
0000540 <set_mylib_int>:
540: 55                push %ebp
541: 89 e5             mov %esp,%ebp
542: e8 20 00 00 00    call 572 <_x86_get_pc_thunk.ax>
543: 54                add $0x14,%eax
544: 8b 84 ec ff ff ff mov -0x14(%eax),%eax
545: 8b 5c 08          mov 0x8(%eax),%ecx
546: 89 18             mov %ecx,%eax
547: 5d                pop %ebp
548: c3                ret
0000550 <get_mylib_int>:
550: 55                push %ebp
551: 89 e5             mov %esp,%ebp
552: e8 20 00 00 00    call 572 <_x86_get_pc_thunk.ax>
553: 54                add $0x14,%eax
554: 8b 84 ec ff ff ff mov -0x14(%eax),%eax
555: 8b 5c 08          mov 0x8(%eax),%ecx
556: 89 18             mov %ecx,%eax
557: 5d                pop %ebp
558: c3                ret
0000572 <_x86_get_pc_thunk.ax>:
572: 8b 54 24          mov 0x24(%eax),%eax
573: c3                ret
```

```
next instruction: 0x77f0148 + 100
```

```
53f: c3                ret
0000540 <set_mylib_int>:
540: 55                push %ebp
541: 89 e5             mov %esp,%ebp
542: e8 20 00 00 00    call 572 <_x86_get_pc_thunk.ax>
543: 54                add $0x14,%eax
544: 8b 84 ec ff ff ff mov -0x14(%eax),%eax
545: 8b 5c 08          mov 0x8(%eax),%ecx
546: 89 18             mov %ecx,%eax
547: 5d                pop %ebp
548: c3                ret
0000550 <get_mylib_int>:
550: 55                push %ebp
551: 89 e5             mov %esp,%ebp
552: e8 20 00 00 00    call 572 <_x86_get_pc_thunk.ax>
553: 54                add $0x14,%eax
554: 8b 84 ec ff ff ff mov -0x14(%eax),%eax
555: 8b 5c 08          mov 0x8(%eax),%ecx
556: 89 18             mov %ecx,%eax
557: 5d                pop %ebp
558: c3                ret
0000572 <_x86_get_pc_thunk.ax>:
572: 8b 54 24          mov 0x24(%eax),%eax
573: c3                ret
```

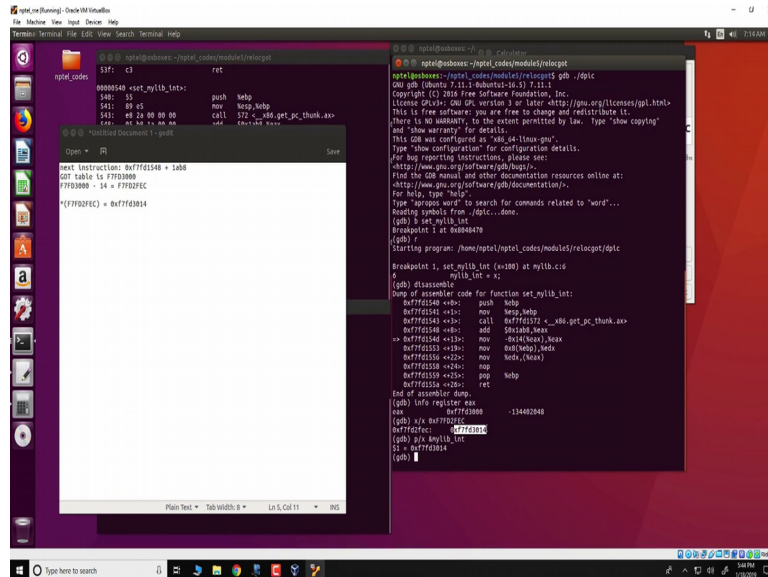


So let us see how this is working internally with the GDB, `gdb./dpic`, we put a breakpoint at `setmylib_int` and run the program and then we disassemble the function, so you see this okay, so there is a call to this `getpcthunk.ax`, so as we know what happens over here is the address of the next instruction that is this instruction gets moved into the EAX register right and let us open G edit and just make note of it that the EAX register comprises of the next instruction, to this EAX register we are adding the offset of `1AB8`.

Let see what we get over here, take the calculator put it in hexadecimal mode and plus `1AB8`, so what we see is that we get an address of `X7FD3000`, address of our GOT table is `F7FD3000` so we can just store it over here and we can also verify this by looking at GDB and we can see this by looking at contents of the EAX register as follows, register EAX and we see that indeed it has `F7FD3000`.

Now within the GOT table at an offset of 0X-14 is where we have the address of the mylib_int present, so let us subtract from this, the value of 0X14 and we see this entry, so F7FD3000-14 is this entry which is the entry in GOT table which contains the address of the variable mylib_int.

(Refer Slide Time: 10:30)

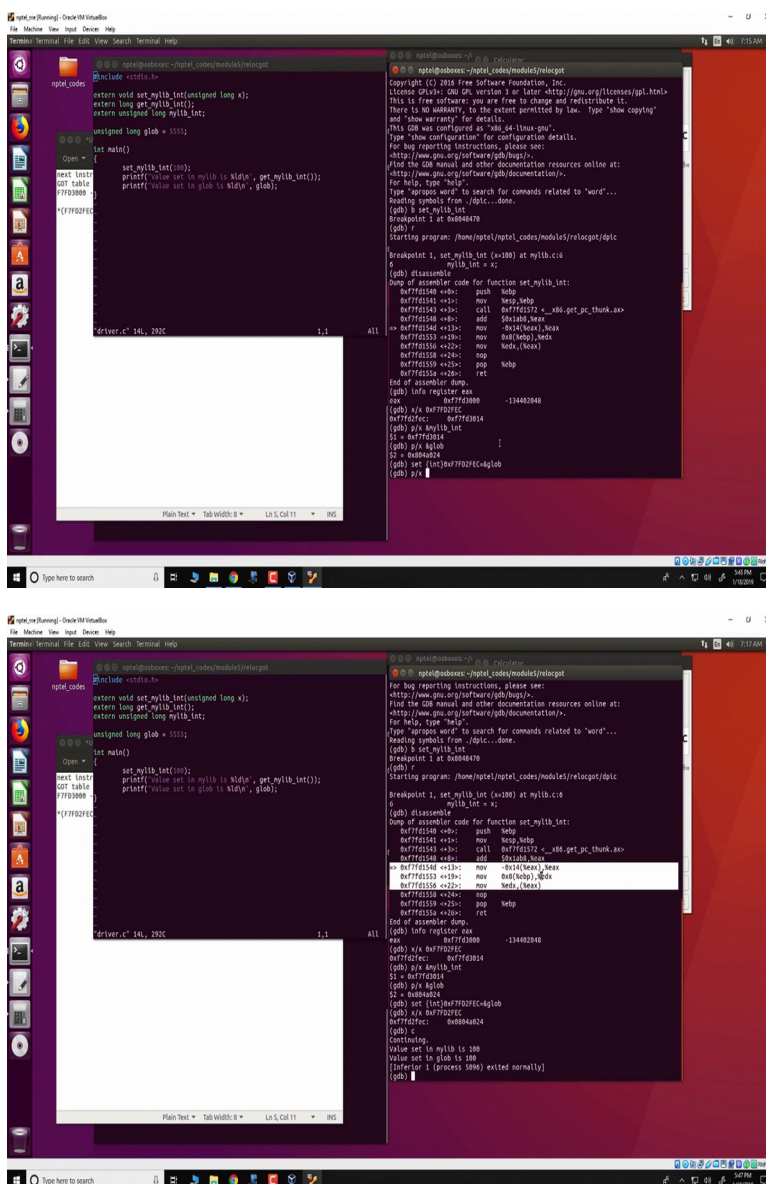


```
gdb (gdb) disassemble
Dump of assembler code for function set_mylib_int:
00005400 <+0>: push    %ebp
00005401 <+1>: mov     %esp,%ebp
00005402 <+2>: call   @PLT@, @GOT@(%rip), @GOT@(%rip), @GOT@(%rip), @GOT@(%rip)
00005403 <+3>: add    $0x14,%esp
00005404 <+4>: mov    %eax,%eax
00005405 <+5>: mov    %eax,%eax
00005406 <+6>: mov    %eax,%eax
00005407 <+7>: mov    %eax,%eax
00005408 <+8>: pop    %ebp
00005409 <+9>: ret
End of assembler dump.
(gdb) info register eax
eax             0xf7fd3000
(gdb) x/x 0xf7fd3000
0xf7fd3000: 0xf7fd3000
(gdb) p/x $mylib_int
$5 = 0xf7fd3000
(gdb)
```

So let us actually look at the contents of this location, so you can dump that memory location using the GDB command x/x and paste and put a 0X here and we see this F7, so this address is the location of mylib_int, so we can check this by also printing the address of mylib_int, where you see that it is identical, so essentially what is happening here is the GOT table stores the offset for each and every variable present in the program, this GOT table is fixed in the program space but the variables move into any region in the program space.

The loader will ensure that the contents of the GOT table will be appropriately modified, so as to reflect the correct address of these variables, so we can actually play a small trick over here, if we modify this particular location which contains the address of the global variable mylib_int, we can actually cost the program to behave spuriously.

(Refer Slide Time: 12:10)



So see the driver.c and let us look at the address of this global variable glob, so that is done by p/x glob and we see that this has an address 804A024, now what we can do is change the contents of the GOT table to point to glob, so this is done as follows, we can set int to be equal to ampersand glob, so what we have done over here is that the GOT entry for mylib_int now points to this global variable glob.

You can check this, so like p/x okay, so this is the address of glob and now if you continue executing what you see is that we have got a invalid a malicious output, so the value of glob according to this program should have been 5555 but since we have modify the GOT table therefore glob actually has a value of 100, so in this way even though ASLR actually makes attacks much more difficult but still there are potential flaws which an attacker could use to

manipulate the working of the program, it could actually change the behaviour of the program and so on. Thank you.