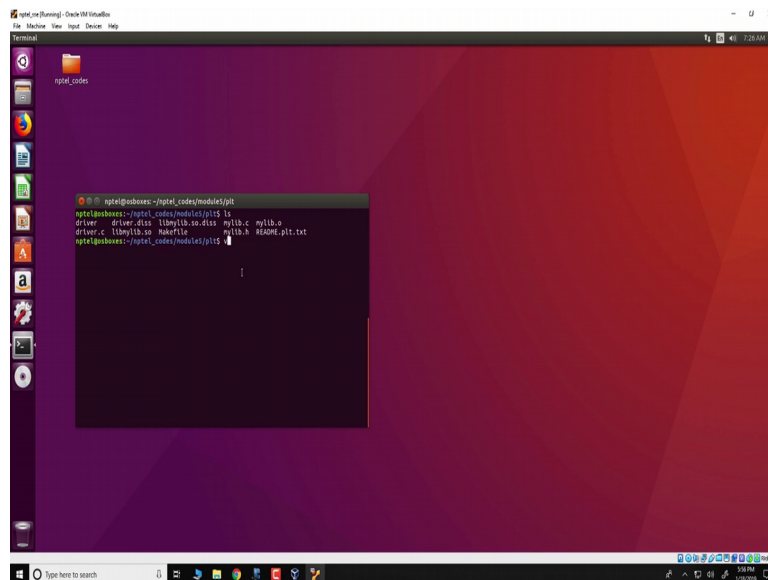


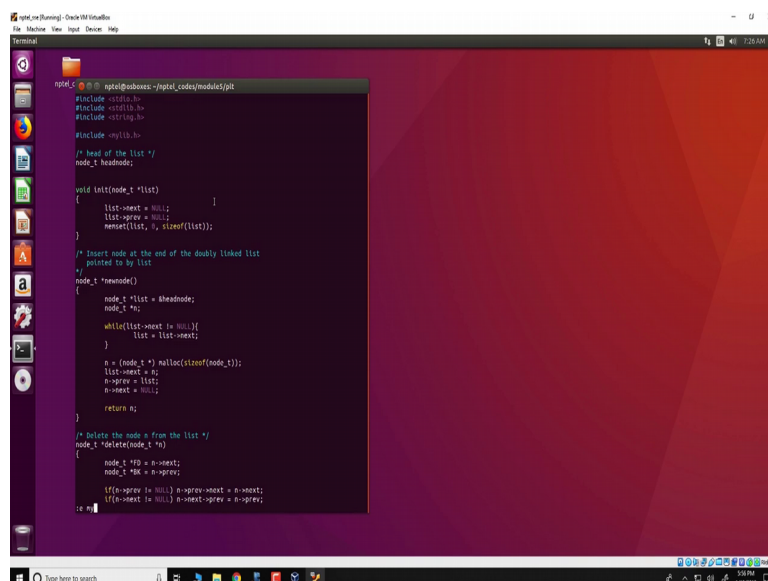
**Information Security 5 Secure System Engineering**  
**Prof. Chester Rebeiro**  
**Indian Institute of Technology Madras**  
**PLT Demonstration**  
**Mod03\_Lec20**

Hello and welcome to this demonstration in the course for secure system engineering, so we will be looking at a demonstration of how PLT works, so PLT is essentially used to achieve a position independent code for functions, it is used quite a bit in order to achieve address space layout randomization and as we have seen the previous lectures ASLR can actually prevent a lot of at this, so this particular demonstration would actually show how would we need a PLT code or rather how PLT code actually works internally.

(Refer Slide Time: 0:57)



```
ngpt@ngpt:~/ngpt_codes/modules/plt
ngpt@ngpt:~/ngpt_codes/modules/plt$ ls
driver driver.diss libmylib.so.diss mylib.c mylib.o
driver.c libmylib.so makefile lib.h README.plt.txt
ngpt@ngpt:~/ngpt_codes/modules/plt$
```



```
ngpt@ngpt:~/ngpt_codes/modules/plt
ngpt@ngpt:~/ngpt_codes/modules/plt$ cat driver.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mylib.h>

/* head of the list */
node_t headnode;

void list(node_t *list)
{
    list->next = NULL;
    list->prev = NULL;
    memset(list, 0, sizeof(list));
}

/* Insert node at the end of the doubly linked list
   pointed to by list
*/
node_t *newnode()
{
    node_t *list = &headnode;
    node_t *n;

    while(list->next != NULL){
        list = list->next;
    }

    n = (node_t *) malloc(sizeof(node_t));
    list->next = n;
    n->prev = list;
    n->next = NULL;

    return n;
}

/* Deletes the node n from the list */
node_t *delete(node_t *n)
{
    node_t *p = n->next;
    node_t *q = n->prev;

    if(n->prev != NULL) n->prev->next = n->next;
    if(n->next != NULL) n->next->prev = n->prev;
    return n;
}
```

```
npTEL_C @ npTEL@osboxes:~/npTEL_codes/module5/plt
#include <stdio.h>
#define MAX_SIZE 100

typedef struct node_t {
    char data[100];
    struct node_t *next;
    struct node_t *prev;
} node_t;

node_t *newnode();
node_t *delete(node_t *n);
void print();

extern node_t headnode;

#endif

mylib.h: 17, 232C      1,1      all
```

So the code we use over here is a present in the virtual machine that is comes along with this course and this code in particular is present in this directory npTEL\_codes module 5 PLT, so we used two C codes over here, one is mylib.c and the driver.c, so mylib.c compiles to a library libmylib.so and driver.c uses this library and creates the executable call driver, so mylib.c look something like this, it is essentially a library for doubly link list, it defines a structure for the list which is defined in mylib.h, this is a doubly linked list, so it has the data and it has the previous and the next node pointers.

(Refer Slide Time: 2:00)

```
mpdl_c @ mpdl@soboxes:~/mpdl_codes/modules/pit
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mylib.h>

/* head of the list */
node_t headnode;

void init(node_t *list)
{
    list->next = NULL;
    list->prev = NULL;
    memset(list, 0, sizeof(list));
}

/* Insert node at the end of the doubly linked list
   pointed to by list */
node_t *newnode()
{
    node_t *list = &headnode;
    node_t *n;

    while(list->next != NULL){
        list = list->next;
    }

    n = (node_t *) malloc(sizeof(node_t));
    list->next = n;
    n->prev = list;
    n->next = NULL;

    return n;
}

/* Delete the node n from the list */
node_t *delete(node_t *n)
{
    node_t *p = n->prev;
    node_t *q = n->next;

    if(n->prev != NULL) n->prev->next = n->next;
    if(n->next != NULL) n->next->prev = n->prev;
    free(n);
}

void malicious_function()
{
    printf("in malicious function!\n");
    exit(0);
}

void main(int argc, char **argv)
{
    node_t *n1, *n2, *n3, *n4;

    init(&headnode);
    n1 = newnode();
    strcpy(n1->data, "A");

    n2 = newnode();
    strcpy(n2->data, "B");

    n3 = newnode();
    strcpy(n3->data, "C");

    print();

    delete(n3);
    delete(n2);
    delete(n1);
    print();
}

/* Print all out */
void print()
{
    node_t *list = &headnode;

    while(list->next != NULL){
        list = list->next;
        printf("%s\n", list->data);
    }
}
```

And there are a lot of functionalities which are present in this code, so for example you could add a new node to this list, you can delete node and you can actually print all the elements in the list, so this particular code is called from driver.c as follows so this is an example over here, you see that we have included mylib.h and we have a main function over here, we initialise that doubly linked list, create nodes like this, new node as your web created three nodes N1, N2 and N3 each comprising of the data A, B and C and then printed the doubly linked list.

(Refer Slide Time: 2:53)

```

nptel@osboxes:~/nptel_codes/modules/p15
nptel@osboxes:~/nptel_codes/modules/p15$ cat driver.c
driver.c libmylib.so makefile mylib.h README.pit.txt
nptel@osboxes:~/nptel_codes/modules/p15$ make clean
rm -f *.o *.so *.d.diss
rm -f driver assignment
nptel@osboxes:~/nptel_codes/modules/p15$ make
gcc -m32 -i . -fPIC -g -c mylib.c -o mylib.o
gcc -m32 -shared -fPIC -o libmylib.so mylib.o
objdump --disassemble-all libmylib.so > libmylib.so.diss
nptel@osboxes:~/nptel_codes/modules/p15$ make driver
gcc -m32 -g driver.c -i . -lmylib -o driver
driver.c: In function 'main':
driver.c:137:22: warning: Implicit declaration of function 'list' [-Wimplicit-func
tion-declaration]
    list(&headNode);
    ^
nptel@osboxes:~/nptel_codes/modules/p15$
objdump --disassemble-all driver > driver.diss
nptel@osboxes:~/nptel_codes/modules/p15$ ./driver
./driver: error while loading shared libraries: libmylib.so: cannot open shared
object file: No such file or directory
nptel@osboxes:~/nptel_codes/modules/p15$ export LD_LIBRARY_PATH~/
nptel@osboxes:~/nptel_codes/modules/p15$ ./driver
A
B
C
nptel@osboxes:~/nptel_codes/modules/p15$

```

So in order to compile the mylib we run the makefile, so we run make clean and then make and what we see is that we are able to create library libmylib.so, similarly we could also make the driver as follows okay, so that is a warning which is present here which you can ignore but the important thing for us to consider is that we are linking the driver.c to the library or using this minus L mylib, the output is driver, so when you run driver as follows where it to run we need to first at the LD library path let set as follows leave it point to the entry and run it and what it means are the nodes in the W link list comprising of A, B and C okay.

(Refer Slide Time: 4:07)

```

driver: file format elf32-i386

Disassembly of section .interp:

00040154 <.interp>:
00040154: 2f          dis    (64),hex:(hex)
00040155: 0c          lshl  (hex),hex:(hex)
00040156: 09 02 2f 0c 04 2d 0c    lmul  $0x0c040c,0x2f(hex),hex
00040159: 09 02 78 2e 73 0f    lmul  $0x0f732e78,0x73(hex),hex
00040165: 70 20 08          wr    hex(hex),hex

Disassembly of section .note.stab:

00040168 <.note.stab>:
00040168: 04 00          add  $0x04,hex
00040169: 00 00          add  hex,hex
0004016a: 00 00          add  hex,hex
0004016b: 00 00          add  hex,hex
0004016c: 00 00          add  hex,hex
0004016d: 00 00          add  hex,hex
0004016e: 00 00          add  hex,hex
0004016f: 00 00          add  hex,hex
00040170: 00 00          add  hex,hex
00040171: 00 00          add  hex,hex
00040172: 00 00          add  hex,hex
00040173: 4e          lsc   hexl
00040174: 35          push hex
00040175: 00 00          add  hex,hex
00040176: 00 00          add  hex,hex
00040177: 00 00          add  hex,hex
00040178: 00 00          add  hex,hex
00040179: 00 00          add  hex,hex
0004017a: 00 00          add  hex,hex
0004017b: 00 00          add  hex,hex
0004017c: 00 00          add  hex,hex
0004017d: 00 00          add  hex,hex
0004017e: 00 00          add  hex,hex
0004017f: 47          lsc   hexl

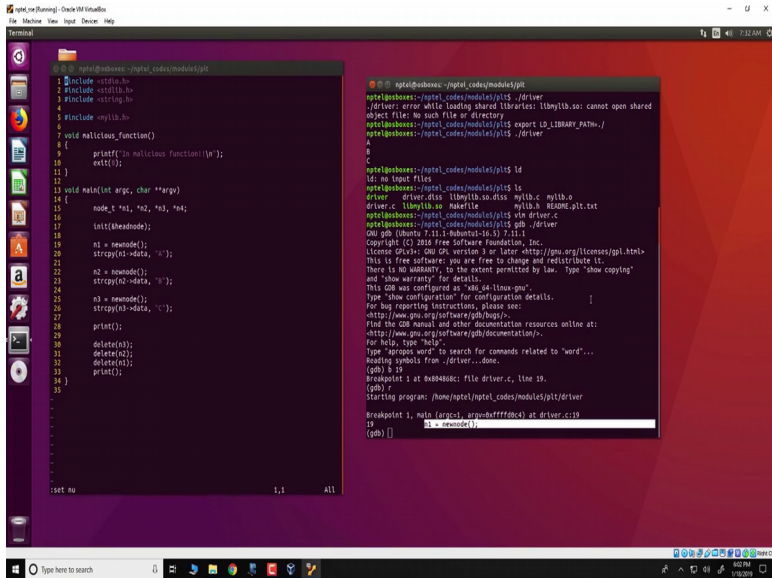
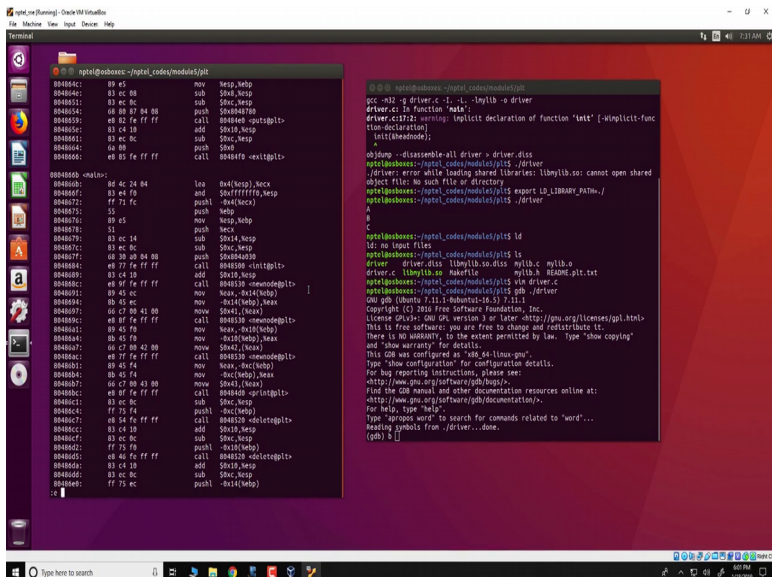
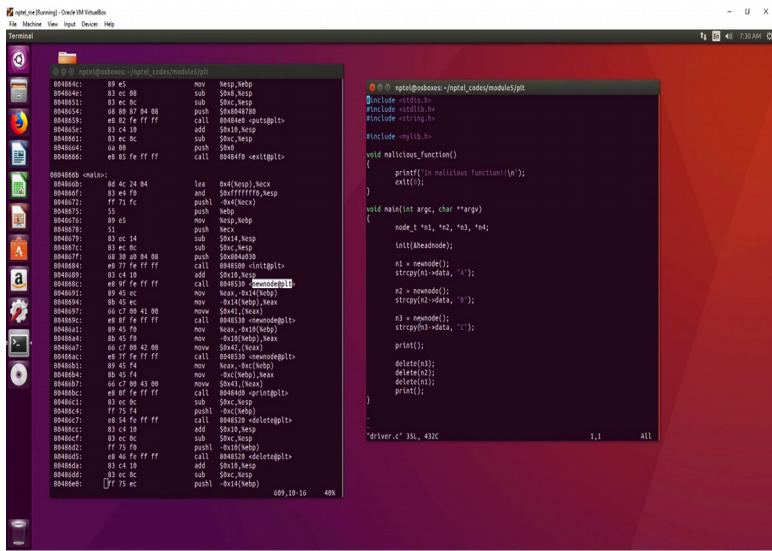
Disassembly of section .note.gnu.build-id:

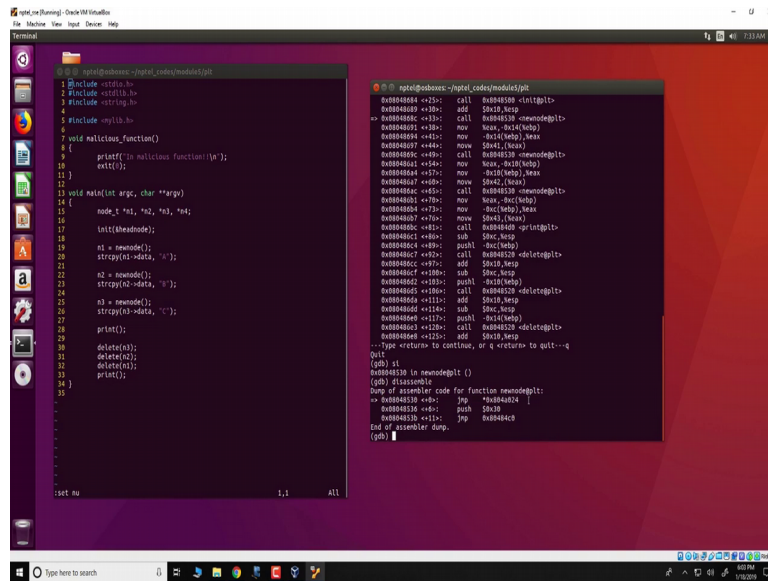
00040188 <.note.gnu.build-id>:
00040188: 04 00          add  $0x04,hex
00040189: 00 00          add  hex,hex
0004018a: 00 00          add  hex,hex
0004018b: 00 00          add  hex,hex
0004018c: 00 00          add  hex,hex
0004018d: 00 00          add  hex,hex
0004018e: 00 00          add  hex,hex
0004018f: 47          lsc   hexl
00040194: 47          lsc   hexl

'driver.diss' 1436c, 03009C
1,0-1    Top

```





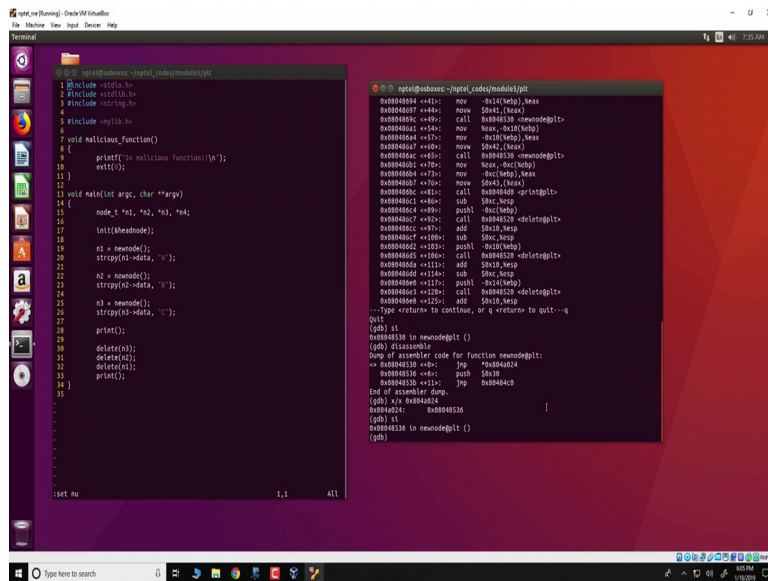


So let us look at a disassembly of the driver code, so this can be obtained in the file `driver.disk` and we will see it over here and will go to the main function, so this is what, how the main function looks like in assembly code, now become this with the actual main function written in C and what we see is that the various function calls to the library like in it, `new node`, `print`, `delete` and so on are all invoked over here through the called function.

However what we see over here is that the compiler has actually modified it slightly and instead of calling, calling `new node` directly it calls something like `new node at PLT`, so let us see what actually happens in this `new node at PLT`, so you could refer to the previous lecture also to see what this `PLT` actually means, so let us use GDB to run driver and we would breakpoint, so we driver, so let us see okay, so we open the `driver.c` code, find out the address, find out the line `new node` is call that is line number 19 and we put a breakpoint at line number nineteen like this and run the code using R, what we see is that there is the breakpoint at line number 19.

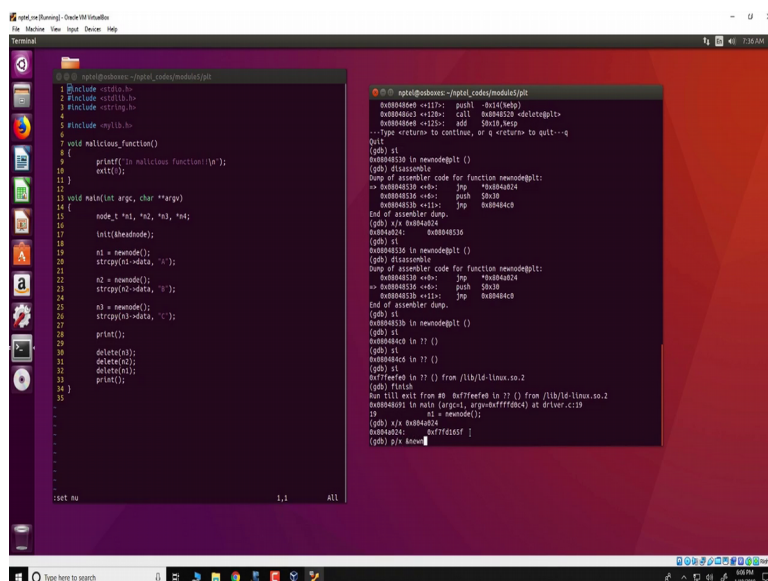
So let us disassemble the code here, so we see that we are at the call to `new node at PLT` and this `new node at PLT` is present at location `8048530`, so this corresponds to a `PLT` section in the code, so we can single step through that and see that we are now in the `new node at PLT`, we can disassemble this and see that it comprises of three instructions an indirect jump, a push and a jump instruction, so during the first call of `new node` which is at this call, so this first jump works as follows, it essentially jumps to the address which is specified in this location over here that is the location `804A024`.

(Refer Slide Time: 7:19)



So let us see the contents of this location 0X804A024 and we see that it contains 08048536, so what it means is that this jump would jump to the address 08048536, now in the first invocation of new node which is at line number 19 what we notice is that this address is in fact the next line in the PLT or the next instruction in the PLT, so essentially what happens is that during the execution this jump essentially jumps to the next line in the PLT code, then here there is a push instruction, a push 0X30 and a jump to this location 0XA0484C0, this location would be in the loader which essentially resolves the actual address of new note and these were would then fill in this location over here with the real or the correct address of new node.

(Refer Slide Time: 8:46)



```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/socket.h>
7 #include <netinet/in.h>
8 #include <arpa/inet.h>
9 #include <netdb.h>
10 #include <sys/time.h>
11 #include <sys/resource.h>
12 #include <sys/wait.h>
13 #include <sys/stat.h>
14 #include <sys/mman.h>
15 #include <sys/uio.h>
16 #include <sys/xattr.h>
17 #include <sys/fsuid.h>
18 #include <sys/prctl.h>
19 #include <sys/ptrace.h>
20 #include <sys/procfs.h>
21 #include <sys/epoll.h>
22 #include <sys/eventfd.h>
23 #include <sys/signalfd.h>
24 #include <sys/timerfd.h>
25 #include <sys/rtld.h>
26 #include <sys/auxv.h>
27 #include <sys/auxv.h>
28 #include <sys/auxv.h>
29 #include <sys/auxv.h>
30 #include <sys/auxv.h>
31 #include <sys/auxv.h>
32 #include <sys/auxv.h>
33 #include <sys/auxv.h>
34 #include <sys/auxv.h>
35 #include <sys/auxv.h>
36 #include <sys/auxv.h>
37 #include <sys/auxv.h>
38 #include <sys/auxv.h>
39 #include <sys/auxv.h>
40 #include <sys/auxv.h>
41 #include <sys/auxv.h>
42 #include <sys/auxv.h>
43 #include <sys/auxv.h>
44 #include <sys/auxv.h>
45 #include <sys/auxv.h>
46 #include <sys/auxv.h>
47 #include <sys/auxv.h>
48 #include <sys/auxv.h>
49 #include <sys/auxv.h>
50 #include <sys/auxv.h>
51 #include <sys/auxv.h>
52 #include <sys/auxv.h>
53 #include <sys/auxv.h>
54 #include <sys/auxv.h>
55 #include <sys/auxv.h>
56 #include <sys/auxv.h>
57 #include <sys/auxv.h>
58 #include <sys/auxv.h>
59 #include <sys/auxv.h>
60 #include <sys/auxv.h>
61 #include <sys/auxv.h>
62 #include <sys/auxv.h>
63 #include <sys/auxv.h>
64 #include <sys/auxv.h>
65 #include <sys/auxv.h>
66 #include <sys/auxv.h>
67 #include <sys/auxv.h>
68 #include <sys/auxv.h>
69 #include <sys/auxv.h>
70 #include <sys/auxv.h>
71 #include <sys/auxv.h>
72 #include <sys/auxv.h>
73 #include <sys/auxv.h>
74 #include <sys/auxv.h>
75 #include <sys/auxv.h>
76 #include <sys/auxv.h>
77 #include <sys/auxv.h>
78 #include <sys/auxv.h>
79 #include <sys/auxv.h>
80 #include <sys/auxv.h>
81 #include <sys/auxv.h>
82 #include <sys/auxv.h>
83 #include <sys/auxv.h>
84 #include <sys/auxv.h>
85 #include <sys/auxv.h>
86 #include <sys/auxv.h>
87 #include <sys/auxv.h>
88 #include <sys/auxv.h>
89 #include <sys/auxv.h>
90 #include <sys/auxv.h>
91 #include <sys/auxv.h>
92 #include <sys/auxv.h>
93 #include <sys/auxv.h>
94 #include <sys/auxv.h>
95 #include <sys/auxv.h>
96 #include <sys/auxv.h>
97 #include <sys/auxv.h>
98 #include <sys/auxv.h>
99 #include <sys/auxv.h>
100 #include <sys/auxv.h>

```

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/socket.h>
7 #include <netinet/in.h>
8 #include <arpa/inet.h>
9 #include <netdb.h>
10 #include <sys/time.h>
11 #include <sys/resource.h>
12 #include <sys/wait.h>
13 #include <sys/stat.h>
14 #include <sys/mman.h>
15 #include <sys/uio.h>
16 #include <sys/xattr.h>
17 #include <sys/fsuid.h>
18 #include <sys/prctl.h>
19 #include <sys/ptrace.h>
20 #include <sys/procfs.h>
21 #include <sys/epoll.h>
22 #include <sys/eventfd.h>
23 #include <sys/signalfd.h>
24 #include <sys/timerfd.h>
25 #include <sys/rtld.h>
26 #include <sys/auxv.h>
27 #include <sys/auxv.h>
28 #include <sys/auxv.h>
29 #include <sys/auxv.h>
30 #include <sys/auxv.h>
31 #include <sys/auxv.h>
32 #include <sys/auxv.h>
33 #include <sys/auxv.h>
34 #include <sys/auxv.h>
35 #include <sys/auxv.h>
36 #include <sys/auxv.h>
37 #include <sys/auxv.h>
38 #include <sys/auxv.h>
39 #include <sys/auxv.h>
40 #include <sys/auxv.h>
41 #include <sys/auxv.h>
42 #include <sys/auxv.h>
43 #include <sys/auxv.h>
44 #include <sys/auxv.h>
45 #include <sys/auxv.h>
46 #include <sys/auxv.h>
47 #include <sys/auxv.h>
48 #include <sys/auxv.h>
49 #include <sys/auxv.h>
50 #include <sys/auxv.h>
51 #include <sys/auxv.h>
52 #include <sys/auxv.h>
53 #include <sys/auxv.h>
54 #include <sys/auxv.h>
55 #include <sys/auxv.h>
56 #include <sys/auxv.h>
57 #include <sys/auxv.h>
58 #include <sys/auxv.h>
59 #include <sys/auxv.h>
60 #include <sys/auxv.h>
61 #include <sys/auxv.h>
62 #include <sys/auxv.h>
63 #include <sys/auxv.h>
64 #include <sys/auxv.h>
65 #include <sys/auxv.h>
66 #include <sys/auxv.h>
67 #include <sys/auxv.h>
68 #include <sys/auxv.h>
69 #include <sys/auxv.h>
70 #include <sys/auxv.h>
71 #include <sys/auxv.h>
72 #include <sys/auxv.h>
73 #include <sys/auxv.h>
74 #include <sys/auxv.h>
75 #include <sys/auxv.h>
76 #include <sys/auxv.h>
77 #include <sys/auxv.h>
78 #include <sys/auxv.h>
79 #include <sys/auxv.h>
80 #include <sys/auxv.h>
81 #include <sys/auxv.h>
82 #include <sys/auxv.h>
83 #include <sys/auxv.h>
84 #include <sys/auxv.h>
85 #include <sys/auxv.h>
86 #include <sys/auxv.h>
87 #include <sys/auxv.h>
88 #include <sys/auxv.h>
89 #include <sys/auxv.h>
90 #include <sys/auxv.h>
91 #include <sys/auxv.h>
92 #include <sys/auxv.h>
93 #include <sys/auxv.h>
94 #include <sys/auxv.h>
95 #include <sys/auxv.h>
96 #include <sys/auxv.h>
97 #include <sys/auxv.h>
98 #include <sys/auxv.h>
99 #include <sys/auxv.h>
100 #include <sys/auxv.h>

```

So let us see this happening, so we will single step through this, see that we are in the next line, so we step again and see that we are at the jump, now we are going to jump into the resolver which is here and we can come out of this a resolver by entering a command finish and we see that we have come out of the resolver, so what the resolver has the done it has gone into this particular address, so this is the GOT entry for new node and it has filled in the correct address for new node in this location.

So let us verify that, so we can rerun this same command as follows and what we see is that this value present in this location has changed to F7FD165F, so if we, what we see is that this address is in fact the address of new node, so printing the address of new node we see that there is a match, so essentially what is happening is that during the first invocation of new node it jumps into the PLT and then obtains an offset from the GOT entry for that particular

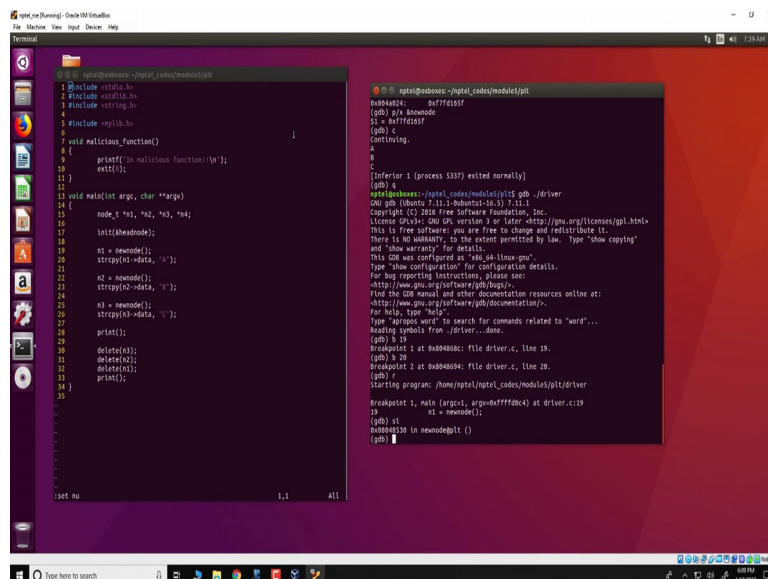


function new node and then goes into a resolver, the resolver resolves this address of this function and modifies the GOT entry over here as we see in this thing and then continues the execution.

Now during the second invocation of new node we would directly jump using this indirect jump to this location, that is the location of new node itself and therefore only the first invocation of new node would actually require the resolver to be used in order to resolve the actual address of the new node function, all subsequent invocations of new node would directly jump to the new node a function using this indirect jump and we can continue the execution.

Now what we see here is that there is a malicious function, now what we can do is we can actually trick this entire thing into executing this malicious function, note that the main function we see over here there is no invocation of malicious function but what we will show is that we can trick this entire system into invoking the malicious function, let say this as follows.

(Refer Slide Time: 12:20)





```
gdb@ubuntu:~/nptel_codes/modules/gpl$
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #include <lib.h>
6
7 void malicious_function()
8 {
9     printf("In malicious function!\n");
10    exit(1);
11 }
12
13 void main(int argc, char **argv)
14 {
15     node_t *n1, *n2, *n3, *n4;
16     int i(headnode);
17
18     n1 = newnode();
19     strcpy(n1->data, "A");
20
21     n2 = newnode();
22     strcpy(n2->data, "B");
23
24     n3 = newnode();
25     strcpy(n3->data, "C");
26
27     print();
28
29     delete(n3);
30     delete(n2);
31     delete(n1);
32     print();
33 }
34
35
:set nu 1,1 All
```

```
gdb@ubuntu:~/nptel_codes/modules/gpl$
License (GPL): GNU GPL version 3 or later ->http://gnu.org/licenses/gpl.html
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show warranty"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./driver...done.
(gdb) b 19
Breakpoint 1 at 0x8048084: file driver.c, line 19.
(gdb) b 20
Breakpoint 2 at 0x8048094: file driver.c, line 20.
(gdb)
Starting program: /home/nptel/nptel_codes/modules/gpl/driver
Breakpoint 1, main (argc=1, argv=0xffffd0c4) at driver.c:19
19      n1 = newnode();
(gdb) at
0x8048030 in newnode@plt ()
(gdb) disassemble
Dump of assembler code for function newnode@plt:
=> 0x8048030 <+>: jmp *0x8048024
0x8048030 <+>: push $0x0
0x8048030 <+is: jmp 0x804803c
End of assembler dump.
(gdb) c
Continuing.
Breakpoint 2, main (argc=1, argv=0xffffd0c4) at driver.c:20
20      strcpy(n1->data, "A");
(gdb) /x 0x8048024
0x8048024: 0xfffff00f
(gdb) /x $newnode
$1 = 0xfffff00f
(gdb) set: [01]0x8048024=malicious_function
(gdb) /x 0x8048024
0x8048024: 0x8048040
(gdb) c
[Inferior 1 (process 5333) exited normally]
(gdb)
```

```
gdb@ubuntu:~/nptel_codes/modules/gpl$
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #include <lib.h>
6
7 void malicious_function()
8 {
9     printf("In malicious function!\n");
10    exit(1);
11 }
12
13 void main(int argc, char **argv)
14 {
15     node_t *n1, *n2, *n3, *n4;
16     int i(headnode);
17
18     n1 = newnode();
19     strcpy(n1->data, "A");
20
21     n2 = newnode();
22     strcpy(n2->data, "B");
23
24     n3 = newnode();
25     strcpy(n3->data, "C");
26
27     print();
28
29     delete(n3);
30     delete(n2);
31     delete(n1);
32     print();
33 }
34
35
:set nu 1,1 All
```

```
gdb@ubuntu:~/nptel_codes/modules/gpl$
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./driver...done.
(gdb) b 19
Breakpoint 1 at 0x8048084: file driver.c, line 19.
(gdb) b 20
Breakpoint 2 at 0x8048094: file driver.c, line 20.
(gdb)
Starting program: /home/nptel/nptel_codes/modules/gpl/driver
Breakpoint 1, main (argc=1, argv=0xffffd0c4) at driver.c:19
19      n1 = newnode();
(gdb) at
0x8048030 in newnode@plt ()
(gdb) disassemble
Dump of assembler code for function newnode@plt:
=> 0x8048030 <+>: jmp *0x8048024
0x8048030 <+>: push $0x0
0x8048030 <+is: jmp 0x804803c
End of assembler dump.
(gdb) c
Continuing.
Breakpoint 2, main (argc=1, argv=0xffffd0c4) at driver.c:20
20      strcpy(n1->data, "A");
(gdb) /x 0x8048024
0x8048024: 0xfffff00f
(gdb) /x $newnode
$1 = 0xfffff00f
(gdb) set: [01]0x8048024=malicious_function
(gdb) /x 0x8048024
0x8048024: 0x8048040
(gdb) c
```

```
gdb@ubuntu:~/nptel_codes/modules/gpl$
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #include <lib.h>
6
7 void malicious_function()
8 {
9     printf("In malicious function!\n");
10    exit(1);
11 }
12
13 void main(int argc, char **argv)
14 {
15     node_t *n1, *n2, *n3, *n4;
16     int i(headnode);
17
18     n1 = newnode();
19     strcpy(n1->data, "A");
20
21     n2 = newnode();
22     strcpy(n2->data, "B");
23
24     n3 = newnode();
25     strcpy(n3->data, "C");
26
27     print();
28
29     delete(n3);
30     delete(n2);
31     delete(n1);
32     print();
33 }
34
35
:set nu 1,1 All
```

```
gdb@ubuntu:~/nptel_codes/modules/gpl$
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./driver...done.
(gdb) b 19
Breakpoint 1 at 0x8048084: file driver.c, line 19.
(gdb) b 20
Breakpoint 2 at 0x8048094: file driver.c, line 20.
(gdb)
Starting program: /home/nptel/nptel_codes/modules/gpl/driver
Breakpoint 1, main (argc=1, argv=0xffffd0c4) at driver.c:19
19      n1 = newnode();
(gdb) at
0x8048030 in newnode@plt ()
(gdb) disassemble
Dump of assembler code for function newnode@plt:
=> 0x8048030 <+>: jmp *0x8048024
0x8048030 <+>: push $0x0
0x8048030 <+is: jmp 0x804803c
End of assembler dump.
(gdb) c
Continuing.
Breakpoint 2, main (argc=1, argv=0xffffd0c4) at driver.c:20
20      strcpy(n1->data, "A");
(gdb) /x 0x8048024
0x8048024: 0xfffff00f
(gdb) /x $newnode
$1 = 0xfffff00f
(gdb) set: [01]0x8048024=malicious_function
(gdb) /x 0x8048024
0x8048024: 0x8048040
(gdb) c
[Inferior 1 (process 5333) exited normally]
(gdb)
```

So let restart the GDB and we will put a breakpoint in line number 19 and also a breakpoint in line number 20 and then run the program and this single step instruction and as before we have gone into the new node PLT, now what we will do is note down the address of the GOT entry, so we will see that the GOT entry as before is at the location here okay, this is all that we require, will continue executing and what we see as done before that this particular GOT entry would be fill with the actual address of new node, we can verify that again as x/x and d/x at percent new node.

So you see that as before the resolver has actually resolved new node during the first invocation, now what we can do is we could modify the GOT entry and we could make it a point to the malicious function as follows, so what we do is set int, GOT entry is this, so this again my clipboard equal to amphi cent and what we see is that the GOT entry has been modified and now points to the malicious function.

Now in the second invocation of new node it goes into this PLT and picks, and jumps to the address has specified this location which essentially is the malicious function and therefore it is going to execute the malicious function, let us see if we continue the execution what would happen, so right enough instead of executing new node it has indeed come into the malicious function.

So in this way what we have seen is that, we have seen how GDB can be used to actually change the execution pattern of this particular program, in a more malicious application in marijuana for example and an attacker would use a wonderbility in the program modify the GOT entry and thereby change its execution pattern, so we will putting up a few assignments or some challenges on the website and you could actually try to use the wonderbility to show how you could use a GOT entry to modify the execution pattern. Thank you.