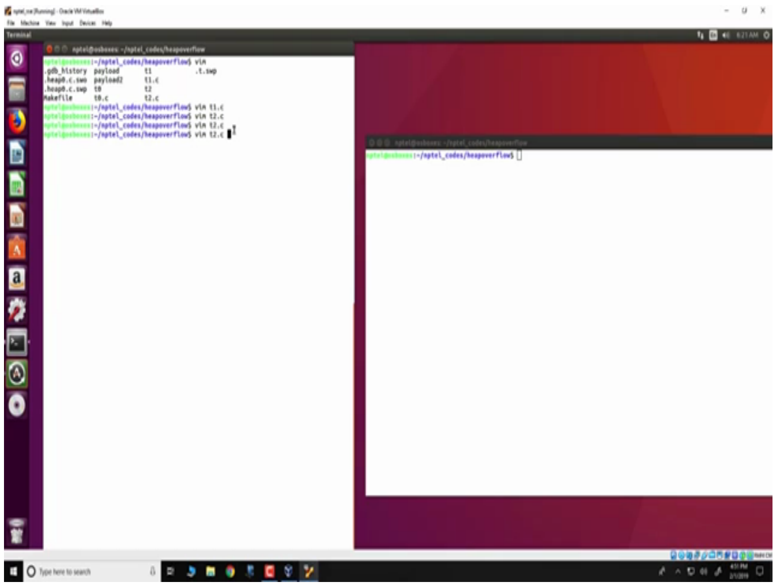
Information Security - 5 - Secure Systems Engineering
Professor Chester Rebeiro
Indian Institute of Technology, Madras
Heap_Demo3

Hello and welcome to this third demonstration for heaps, this is part of the secure system course as part of the NPTEL.

(Refer Slide Time: 0:25)

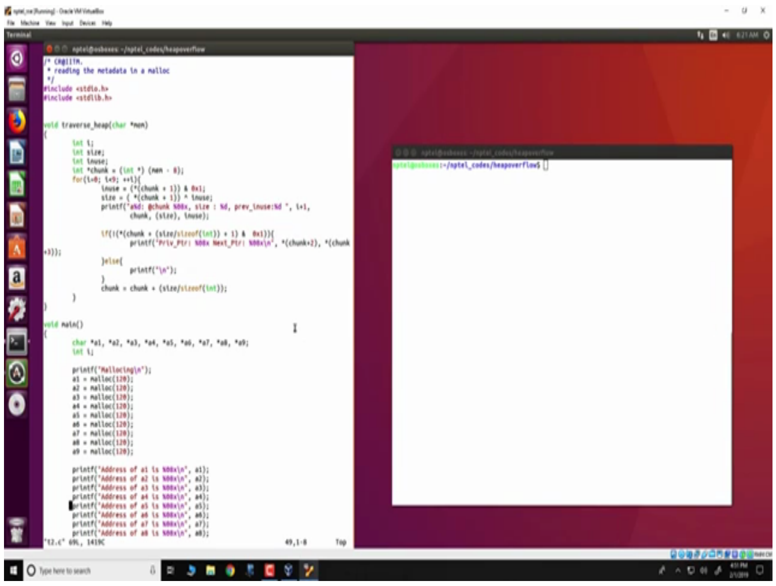


```
nptel@osboxes:~/nptel_codes/heapoverflow$ vim
.gdb_history  payload    t1         .t.swp
.heap0.c.swo  payload2   t1.c
.heap0.c.swp  t0         t2
Makefile      t0.c       t2.c
nptel@osboxes:~/nptel_codes/heapoverflow$ vim t1.c
nptel@osboxes:~/nptel_codes/heapoverflow$ vim t2.c
nptel@osboxes:~/nptel_codes/heapoverflow$ vim t2.c
nptel@osboxes:~/nptel_codes/heapoverflow$ vim t2.c
```

```
nptel@osboxes:~/nptel_codes/heapoverflow$
```

So this code can be downloaded preferably you can download it from your virtual machine which comes along with this course and we could then run these codes. So we are looking at t2 dot C today.

(Refer Slide Time: 0:40)

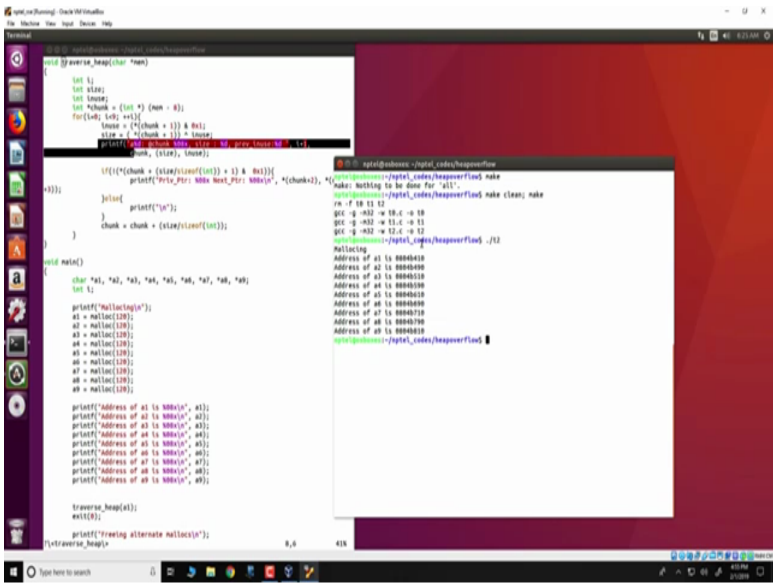


```
/* CR@IITM.
 * reading the metadata in a malloc
 */
#include <stdio.h>
#include <stdlib.h>


void traverse_heap(char *mem)
{
        int i;
        int size;
        int inuse;
        int *chunk = (int *) (mem - 8);
        for(i=0; i<9; ++i){
                inuse = (*(chunk + 1)) & 0x1;
                size = ( *(chunk + 1)) ^ inuse;
                printf("a%d: @chunk %08x, size : %d, prev_inuse:%d ", i+1,
                        chunk, (size), inuse);

                if(!(*(chunk + (size/sizeof(int)) + 1) &  0x1)){
                        printf("Priv_Ptr: %08x Next_Ptr: %08x\n", *(chunk+2), *(chunk
+3));
                }else{
                        printf("\n");
                }
                chunk = chunk + (size/sizeof(int));
        }
}

void main()
{
        char *a1, *a2, *a3, *a4, *a5, *a6, *a7, *a8, *a9;
        int i;

        printf("Malloc'ing\n");
        a1 = malloc(120);
        a2 = malloc(120);
        a3 = malloc(120);
        a4 = malloc(120);
        a5 = malloc(120);
        a6 = malloc(120);
        a7 = malloc(120);
        a8 = malloc(120);
        a9 = malloc(120);

        printf("Address of a1 is %08x\n", a1);
        printf("Address of a2 is %08x\n", a2);
        printf("Address of a3 is %08x\n", a3);
        printf("Address of a4 is %08x\n", a4);
        printf("Address of a5 is %08x\n", a5);
        printf("Address of a6 is %08x\n", a6);
        printf("Address of a7 is %08x\n", a7);
        printf("Address of a8 is %08x\n", a8);
"t2.c" 69L, 1419C                                       49,1-8        Top
```

```
nptel@osboxes:~/nptel_codes/heapoverflow$
```

Now (this would) this particular code again it is not an attack or it does not actually show a vulnerability, but it is essentially used to tell you the some aspects about the internal workings of the heap allocator, there are various parts in this heap allocator.

(Refer Slide Time: 0:58)



```
void traverse_heap(char *mem)
{
        int i;
        int size;
        int inuse;
        int *chunk = (int *) (mem - 8);
        for(i=0; i<9; ++i){
                inuse = (*(chunk + 1)) & 0x1;
                size = ( *(chunk + 1)) ^ inuse;
                printf("a%d: @chunk %08x, size : %d, prev_inuse:%d ", i+1,
chunk, (size), inuse);

                if(!(*(chunk + (size/sizeof(int)) + 1) &  0x1)){
                        printf("Prev_Ptr: %08x Next_Ptr: %08x\n", *(chunk+2), *(c
+3)));
                }else{
                        printf("\n");
                }
                chunk = chunk + (size/sizeof(int));
        }
}

void main()
{
        char *a1, *a2, *a3, *a4, *a5, *a6, *a7, *a8, *a9;
        int i;

        printf("Mallocing\n");
        a1 = malloc(120);
        a2 = malloc(120);
        a3 = malloc(120);
        a4 = malloc(120);
        a5 = malloc(120);
        a6 = malloc(120);
        a7 = malloc(120);
        a8 = malloc(120);
        a9 = malloc(120);

        printf("Address of a1 is %08x\n", a1);
        printf("Address of a2 is %08x\n", a2);
        printf("Address of a3 is %08x\n", a3);
        printf("Address of a4 is %08x\n", a4);
        printf("Address of a5 is %08x\n", a5);
        printf("Address of a6 is %08x\n", a6);
        printf("Address of a7 is %08x\n", a7);
        printf("Address of a8 is %08x\n", a8);
        printf("Address of a9 is %08x\n", a9);


        traverse_heap(a1);
        exit(0);

        printf("Freeing alternate mallocs\n");
/\<traverse_heap\>                                      8,6          41%
```

```
nptel@osboxes:~/nptel_codes/heapoverflow$ make
make: Nothing to be done for 'all'.
nptel@osboxes:~/nptel_codes/heapoverflow$ make clean; make
rm -f t0 t1 t2
gcc -g -m32 -w t0.c -o t0
gcc -g -m32 -w t1.c -o t1
gcc -g -m32 -w t2.c -o t2
nptel@osboxes:~/nptel_codes/heapoverflow$ ./t2
Mallocing
Address of a1 is 0804b410
Address of a2 is 0804b490
Address of a3 is 0804b510
Address of a4 is 0804b590
Address of a5 is 0804b610
Address of a6 is 0804b690
Address of a7 is 0804b710
Address of a8 is 0804b790
Address of a9 is 0804b810
nptel@osboxes:~/nptel_codes/heapoverflow$
```

The first thing we would do so what we will do is look at it one by one, we would break the program in various points like this and we will just run part of this program. So what we do in this particular program is that we define character pointers 9 of them a 1 to a 9 and allocate 120 bytes for each of them and then simply just print the addresses for each of these 9 pointers.

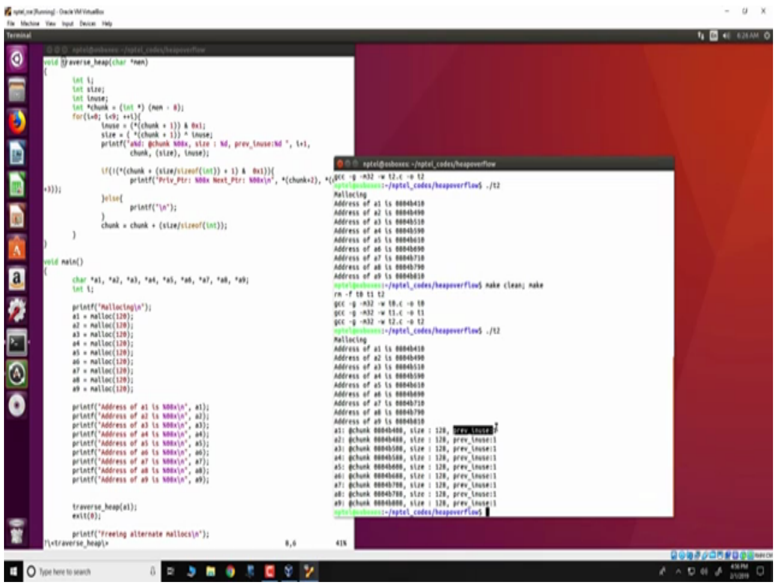
So this is nothing unusual about it, so we would make clean make and run t2, okay and one thing what we need to see is that these 9 pointers have been allocated different addresses. So note that each allocation request is for 120 bytes and if you actually take the difference between any two consecutive pointers, for example a 2 and a 1, a 5 and a 4 or a 4 and a 3 you would see that the difference in these addresses is 128 bytes, the extra 8 bytes comes due to the presence of the metadata.

So for example for the chunk of memory a 1 the metadata is present (no okay) so for example a 2 which starts at 0804B490 the metadata (comprised) which holds the size and other information starts at the location 8 bytes before this, similarly for all other pointers. The next we will see is a very important function known as the traverse heap function. So what we are going to do is that we are going to traverse the heap and look at the various metadata contents present in the heap. So this function so before we go there we can just put an exit here soon after traverse heap just to ensure that we get another partial output.

So this particular traverse heat function takes a memory location. For example a 1 over here which incidentally is the first malloc chunk of memory and then it would obtain a pointer to the metadata for a 1 chunk which is at a location 8 bytes from where the pointer is actually obtained. So the metadata comprises of two important fields, one is the size which is of 31 bits starting from the most significant bit which is the 31st bit to the first bit and the in use bit which is of which is the 0th it or the least significant bit. A value of 1 is in use while a value of 0 over here indicates that the previous chunk is free, the size of the chunk that has been allocated.

So since we have requested for 120 bytes and there is an additional 8 bytes allocated for every malloc request, so what we would expect to see is that the size of this chunk is 128 bytes. So this is a printf which then prints the various aspects it prints a pointer, it prints the size and it determines whether there is it is in use or not in use.

(Refer Slide Time: 4:42)

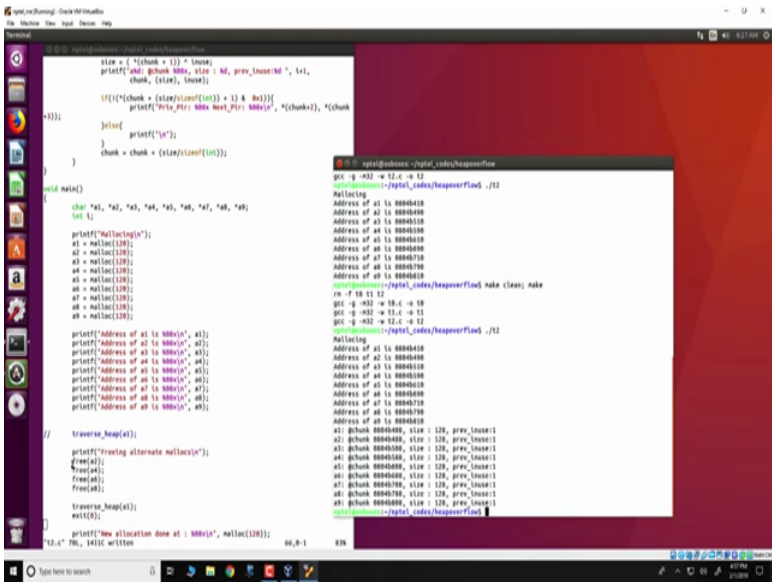


```
void traverse_heap(char *mem)
{
        int i;
        int size;
        int inuse;
        int *chunk = (int *) (mem - 8);
        for(i=0; i<9; ++i){
                inuse = (*(chunk + 1)) & 0x1;
                size = ( *(chunk + 1)) ^ inuse;
                printf("a%d: @chunk %08x, size : %d, prev_inuse:%d ", i+1,
                        chunk, (size), inuse);

                if(!(*(chunk + (size/sizeof(int)) + 1) &  0x1)){
                        printf("Priv_Ptr: %08x Next_Ptr: %08x\n", *(chunk+2), *(
+3));
                }else{
                        printf("\n");
                }
                chunk = chunk + (size/sizeof(int));
        }
}

void main()
{
        char *a1, *a2, *a3, *a4, *a5, *a6, *a7, *a8, *a9;
        int i;

        printf("Mallocing\n");
        a1 = malloc(120);
        a2 = malloc(120);
        a3 = malloc(120);
        a4 = malloc(120);
        a5 = malloc(120);
        a6 = malloc(120);
        a7 = malloc(120);
        a8 = malloc(120);
        a9 = malloc(120);

        printf("Address of a1 is %08x\n", a1);
        printf("Address of a2 is %08x\n", a2);
        printf("Address of a3 is %08x\n", a3);
        printf("Address of a4 is %08x\n", a4);
        printf("Address of a5 is %08x\n", a5);
        printf("Address of a6 is %08x\n", a6);
        printf("Address of a7 is %08x\n", a7);
        printf("Address of a8 is %08x\n", a8);
        printf("Address of a9 is %08x\n", a9);


        traverse_heap(a1);
        exit(0);

        printf("Freeing alternate mallocs\n");
/\<traverse_heap\>                                        8,6           41%
```

```
gcc -g -m32 -w t2.c -o t2
nptel@osboxes:~/nptel_codes/heapoverflow$ ./t2
Mallocing
Address of a1 is 0804b410
Address of a2 is 0804b490
Address of a3 is 0804b510
Address of a4 is 0804b590
Address of a5 is 0804b610
Address of a6 is 0804b690
Address of a7 is 0804b710
Address of a8 is 0804b790
Address of a9 is 0804b810
nptel@osboxes:~/nptel_codes/heapoverflow$ make clean; make
rm -f t0 t1 t2
gcc -g -m32 -w t0.c -o t0
gcc -g -m32 -w t1.c -o t1
gcc -g -m32 -w t2.c -o t2
nptel@osboxes:~/nptel_codes/heapoverflow$ ./t2
Mallocing
Address of a1 is 0804b410
Address of a2 is 0804b490
Address of a3 is 0804b510
Address of a4 is 0804b590
Address of a5 is 0804b610
Address of a6 is 0804b690
Address of a7 is 0804b710
Address of a8 is 0804b790
Address of a9 is 0804b810
a1: @chunk 0804b408, size : 128, prev_inuse:1
a2: @chunk 0804b488, size : 128, prev_inuse:1
a3: @chunk 0804b508, size : 128, prev_inuse:1
a4: @chunk 0804b588, size : 128, prev_inuse:1
a5: @chunk 0804b608, size : 128, prev_inuse:1
a6: @chunk 0804b688, size : 128, prev_inuse:1
a7: @chunk 0804b708, size : 128, prev_inuse:1
a8: @chunk 0804b788, size : 128, prev_inuse:1
a9: @chunk 0804b808, size : 128, prev_inuse:1
nptel@osboxes:~/nptel_codes/heapoverflow$
```

So let us compile and run this program again and we see the internal details of the metadata. Now corresponding to each of these pointers which we have defined we get the location of the chunk. So for example a 1 is at a location 0804B410 and the metadata corresponding to a 1 is at the location 0804B408. Now the size of this metadata as we have mentioned is 128 bytes, (120 bytes for the requested metadata and the remaining 8 bytes) 120 bytes for the requested allocation and 8 bytes for the metadata, we also note that previous in use flag is set to 1, so this indicates that the previous chunk of memory is not in use.

(Refer Slide Time: 5:46)

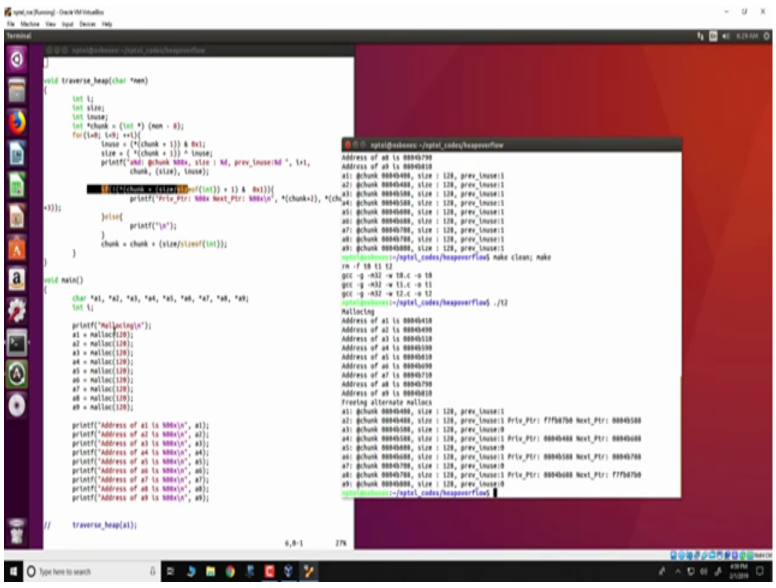


```
                size = ( *(chunk + 1)) ^ inuse;
                printf("a%d: @chunk %08x, size : %d, prev_inuse:%d ", i+1,
                        chunk, (size), inuse);

                if(!(*(chunk + (size/sizeof(int)) + 1) &  0x1)){
                        printf("Priv_Ptr: %08x Next_Ptr: %08x\n", *(chunk+2), *(chunk
+3));
                }else{
                        printf("\n");
                }
                chunk = chunk + (size/sizeof(int));
        }
}

void main()
{
        char *a1, *a2, *a3, *a4, *a5, *a6, *a7, *a8, *a9;
        int i;

        printf("Malloc\ing\n");
        a1 = malloc(120);
        a2 = malloc(120);
        a3 = malloc(120);
        a4 = malloc(120);
        a5 = malloc(120);
        a6 = malloc(120);
        a7 = malloc(120);
        a8 = malloc(120);
        a9 = malloc(120);

        printf("Address of a1 is %08x\n", a1);
        printf("Address of a2 is %08x\n", a2);
        printf("Address of a3 is %08x\n", a3);
        printf("Address of a4 is %08x\n", a4);
        printf("Address of a5 is %08x\n", a5);
        printf("Address of a6 is %08x\n", a6);
        printf("Address of a7 is %08x\n", a7);
        printf("Address of a8 is %08x\n", a8);
        printf("Address of a9 is %08x\n", a9);


//      traverse_heap(a1);

        printf("Freeing alternate mallocs\n");
        free(a2);
        free(a4);
        free(a6);
        free(a8);

        traverse_heap(a1);
        exit(0);
}
        printf("New allocation done at : %08x\n", malloc(120));
"t2.c" 70L, 1411C written                                        66,0-1        83%
```

```
gcc -g -m32 -w t2.c -o t2
nptel@osboxes:~/nptel_codes/heapoverflow$ ./t2
Mallocing
Address of a1 is 0804b410
Address of a2 is 0804b490
Address of a3 is 0804b510
Address of a4 is 0804b590
Address of a5 is 0804b610
Address of a6 is 0804b690
Address of a7 is 0804b710
Address of a8 is 0804b790
Address of a9 is 0804b810
nptel@osboxes:~/nptel_codes/heapoverflow$ make clean; make
rm -f t0 t1 t2
gcc -g -m32 -w t0.c -o t0
gcc -g -m32 -w t1.c -o t1
gcc -g -m32 -w t2.c -o t2
nptel@osboxes:~/nptel_codes/heapoverflow$ ./t2
Mallocing
Address of a1 is 0804b410
Address of a2 is 0804b490
Address of a3 is 0804b510
Address of a4 is 0804b590
Address of a5 is 0804b610
Address of a6 is 0804b690
Address of a7 is 0804b710
Address of a8 is 0804b790
Address of a9 is 0804b810
a1: @chunk 0804b408, size : 128, prev_inuse:1
a2: @chunk 0804b488, size : 128, prev_inuse:1
a3: @chunk 0804b508, size : 128, prev_inuse:1
a4: @chunk 0804b588, size : 128, prev_inuse:1
a5: @chunk 0804b608, size : 128, prev_inuse:1
a6: @chunk 0804b688, size : 128, prev_inuse:1
a7: @chunk 0804b708, size : 128, prev_inuse:1
a8: @chunk 0804b788, size : 128, prev_inuse:1
a9: @chunk 0804b808, size : 128, prev_inuse:1
nptel@osboxes:~/nptel_codes/heapoverflow$ 
```

The next thing we do is we go one step further, we can disable this traverse heap remove the exit from here and notice that we have actually freed a couple of these pointers. So we in fact we have freed a 2, a 4, a 6 and a 8 and we then traverse the heap again and put the exit just after this new traverse and what we are going to see is the metadata present in the heap changed due to the free invocations that are done.

(Refer Slide Time: 6:16)



```
}

void traverse_heap(char *mem)
{
        int i;
        int size;
        int inuse;
        int *chunk = (int *) (mem - 8);
        for(i=0; i<9; ++i){
                inuse = (*(chunk + 1)) & 0x1;
                size = ( *(chunk + 1)) ^ inuse;
                printf("a%d: @chunk %08x, size : %d, prev_inuse:%d ", i+1,
                        chunk, (size), inuse);

                if(!(*(chunk + (size/sizeof(int)) + 1) &  0x1)){
                        printf("Priv_Ptr: %08x Next_Ptr: %08x\n", *(chunk+2), *(ch
+3));
                }else{
                        printf("\n");
                }
                chunk = chunk + (size/sizeof(int));
        }
}

void main()
{
        char *a1, *a2, *a3, *a4, *a5, *a6, *a7, *a8, *a9;
        int i;

        printf("Mallocing\n");
        a1 = malloc(120);
        a2 = malloc(120);
        a3 = malloc(120);
        a4 = malloc(120);
        a5 = malloc(120);
        a6 = malloc(120);
        a7 = malloc(120);
        a8 = malloc(120);
        a9 = malloc(120);

        printf("Address of a1 is %08x\n", a1);
        printf("Address of a2 is %08x\n", a2);
        printf("Address of a3 is %08x\n", a3);
        printf("Address of a4 is %08x\n", a4);
        printf("Address of a5 is %08x\n", a5);
        printf("Address of a6 is %08x\n", a6);
        printf("Address of a7 is %08x\n", a7);
        printf("Address of a8 is %08x\n", a8);
        printf("Address of a9 is %08x\n", a9);


//      traverse_heap(a1);
                                                                 6,0-1         27%
```

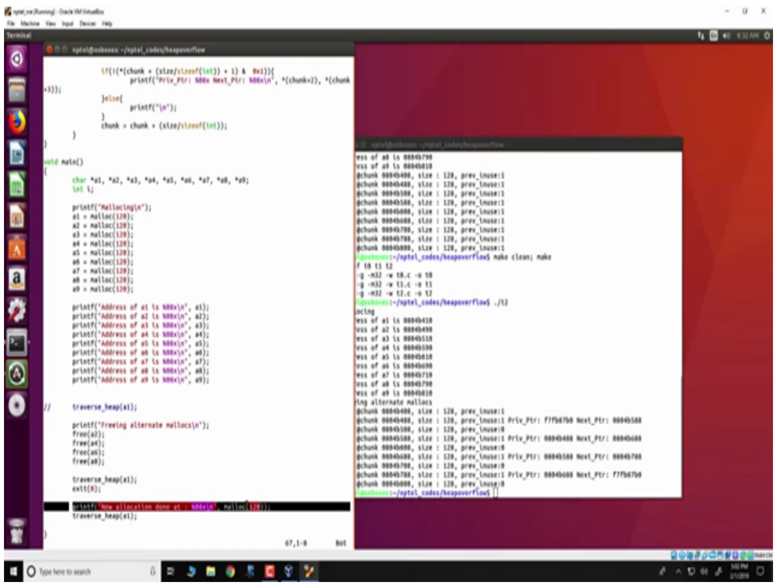
```
Address of a8 is 0804b790
Address of a9 is 0804b810
a1: @chunk 0804b408, size : 128, prev_inuse:1
a2: @chunk 0804b488, size : 128, prev_inuse:1
a3: @chunk 0804b508, size : 128, prev_inuse:1
a4: @chunk 0804b588, size : 128, prev_inuse:1
a5: @chunk 0804b608, size : 128, prev_inuse:1
a6: @chunk 0804b688, size : 128, prev_inuse:1
a7: @chunk 0804b708, size : 128, prev_inuse:1
a8: @chunk 0804b788, size : 128, prev_inuse:1
a9: @chunk 0804b808, size : 128, prev_inuse:1
nptel@osboxes:~/nptel_codes/heapoverflow$ make clean; make
rm -f t0 t1 t2
gcc -g -m32 -w t0.c -o t0
gcc -g -m32 -w t1.c -o t1
gcc -g -m32 -w t2.c -o t2
nptel@osboxes:~/nptel_codes/heapoverflow$ ./t2
Mallocing
Address of a1 is 0804b410
Address of a2 is 0804b490
Address of a3 is 0804b510
Address of a4 is 0804b590
Address of a5 is 0804b610
Address of a6 is 0804b690
Address of a7 is 0804b710
Address of a8 is 0804b790
Address of a9 is 0804b810
Freeing alternate mallocs
a1: @chunk 0804b408, size : 128, prev_inuse:1
a2: @chunk 0804b488, size : 128, prev_inuse:1 Priv_Ptr: f7fb87b0 Next_Ptr: 0804b588
a3: @chunk 0804b508, size : 128, prev_inuse:0
a4: @chunk 0804b588, size : 128, prev_inuse:1 Priv_Ptr: 0804b488 Next_Ptr: 0804b688
a5: @chunk 0804b608, size : 128, prev_inuse:0
a6: @chunk 0804b688, size : 128, prev_inuse:1 Priv_Ptr: 0804b588 Next_Ptr: 0804b788
a7: @chunk 0804b708, size : 128, prev_inuse:0
a8: @chunk 0804b788, size : 128, prev_inuse:1 Priv_Ptr: 0804b688 Next_Ptr: f7fb87b0
a9: @chunk 0804b808, size : 128, prev_inuse:0
nptel@osboxes:~/nptel_codes/heapoverflow$ 
```

So we compile again we see that unlike previously where all the in use bits were set to 1, here because we have actually freed (4 pointers) 4 chunks of memory we have 4 of these previous in use bits set to 1. So let us look at it a little bit more closely. So the first thing to notice is that all of these pointers a 1 to a 9 are contiguous with a 1 having the lowest address followed by a 2, a 3 and so on till a 9.

So when we have freed a 2 the free allocation adds the chunk corresponding to this a 2 pointer in a linked list and it marks then the in use bit in the next chunk as 0. So corresponding to a 2 over here for instance if you just follow these fields, we see that the next chunk of memory corresponding to a 3 the in use bit is set to 0. So we do this check over here in these few lines, what we do is that we first determine if the next chunk that is present in the adjacent chunk that is present has its in use bit set to 0, if so then we print some details about the link list that pt malloc maintains.

So in this particular case the linked lists would be present at the locations chunk plus 2 and chunk plus 3 respectively. So the these memory contains has previous and next pointers to the next nodes in the linked list.

(Refer Slide Time: 8:07)



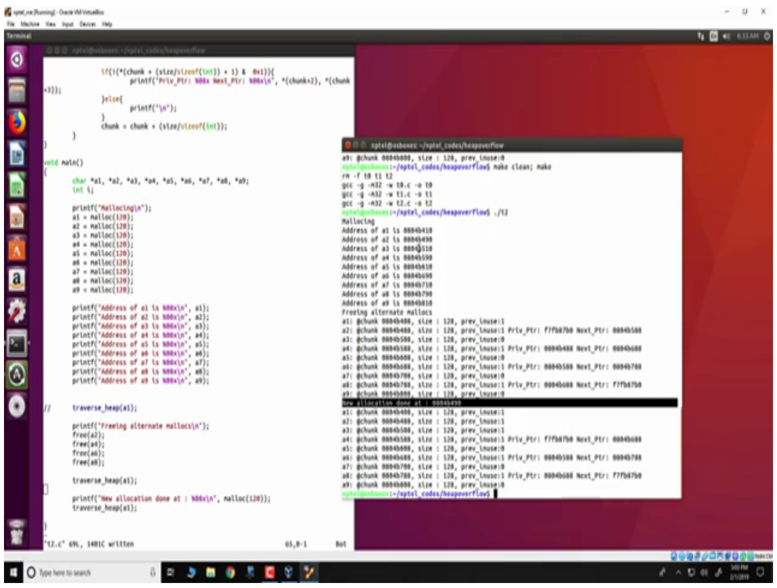
```
		if(!(*(chunk + (size/sizeof(int)) + 1) & 0x1)){
			printf("Priv_Ptr: %08x Next_Ptr: %08x\n", *(chunk+2), *(chunk
+3));
		}else{
			printf("\n");
		}
		chunk = chunk + (size/sizeof(int));
	}
}

void main()
{
	char *a1, *a2, *a3, *a4, *a5, *a6, *a7, *a8, *a9;
	int i;

	printf("Mallocing\n");
	a1 = malloc(120);
	a2 = malloc(120);
	a3 = malloc(120);
	a4 = malloc(120);
	a5 = malloc(120);
	a6 = malloc(120);
	a7 = malloc(120);
	a8 = malloc(120);
	a9 = malloc(120);

	printf("Address of a1 is %08x\n", a1);
	printf("Address of a2 is %08x\n", a2);
	printf("Address of a3 is %08x\n", a3);
	printf("Address of a4 is %08x\n", a4);
	printf("Address of a5 is %08x\n", a5);
	printf("Address of a6 is %08x\n", a6);
	printf("Address of a7 is %08x\n", a7);
	printf("Address of a8 is %08x\n", a8);
	printf("Address of a9 is %08x\n", a9);


//	traverse_heap(a1);

	printf("Freeing alternate mallocs\n");
	free(a2);
	free(a4);
	free(a6);
	free(a8);

	traverse_heap(a1);
	exit(0);

	printf("New allocation done at : %08x\n", malloc(120));
	traverse_heap(a1);

}
                                                  67,1-8        Bot
```

```
ess of a8 is 0804b790
ess of a9 is 0804b810
@chunk 0804b400, size : 128, prev_inuse:1
@chunk 0804b480, size : 128, prev_inuse:1
@chunk 0804b500, size : 128, prev_inuse:1
@chunk 0804b580, size : 128, prev_inuse:1
@chunk 0804b600, size : 128, prev_inuse:1
@chunk 0804b680, size : 128, prev_inuse:1
@chunk 0804b700, size : 128, prev_inuse:1
@chunk 0804b780, size : 128, prev_inuse:1
@chunk 0804b800, size : 128, prev_inuse:1
l@osboxes:~/nptel_codes/heapoverflow$ make clean; make
f t0 t1 t2
-g -m32 -w t0.c -o t0
-g -m32 -w t1.c -o t1
-g -m32 -w t2.c -o t2
l@osboxes:~/nptel_codes/heapoverflow$ ./t2
ocing
ess of a1 is 0804b410
ess of a2 is 0804b490
ess of a3 is 0804b510
ess of a4 is 0804b590
ess of a5 is 0804b610
ess of a6 is 0804b690
ess of a7 is 0804b710
ess of a8 is 0804b790
ess of a9 is 0804b810
ing alternate mallocs
@chunk 0804b400, size : 128, prev_inuse:1
@chunk 0804b480, size : 128, prev_inuse:1 Priv_Ptr: f7fb87b0 Next_Ptr: 0804b580
@chunk 0804b500, size : 128, prev_inuse:0
@chunk 0804b580, size : 128, prev_inuse:1 Priv_Ptr: 0804b480 Next_Ptr: 0804b680
@chunk 0804b600, size : 128, prev_inuse:0
@chunk 0804b680, size : 128, prev_inuse:1 Priv_Ptr: 0804b580 Next_Ptr: 0804b780
@chunk 0804b700, size : 128, prev_inuse:0
@chunk 0804b780, size : 128, prev_inuse:1 Priv_Ptr: 0804b680 Next_Ptr: f7fb87b0
@chunk 0804b800, size : 128, prev_inuse:0
l@osboxes:~/nptel_codes/heapoverflow$
```

So since we have deallocated or which since we have freed 4 chunks of memory a 2, a 4, a 6 and a 8 we have 4 nodes in the linked list. So we could actually follow this particular linked list, we would start from here now this particular pointer f7fb87b0 would actually point to some location within the heap management, so this would permit pt malloc to identify the head of the linked list.
]

The next pointer is 0804B588 which essentially points to the next free chunk which is this and you look at this particular the next free chunk that is this free chunk we see the previous pointer pointing to this one the first chunk that is freed and the next pointer pointing to the next chunk. So what we have seen over here is a doubly linked list we have this chunk which is pointing to the second chunk, the second chunk points to the third chunk, third chunk points the four chunk and the other way also. Also note that this is a circular linked list because the last freed chunk in the list also points to the same location as that of the first chunk that is f7fb87b0.

(Refer Slide Time: 10:00)



```
                if(!(*(chunk + (size/sizeof(int)) + 1) & 0x1)){
                        printf("Priv_Ptr: %08x Next_Ptr: %08x\n", *(chunk+2), *(chunk
+3));
                }else{
                        printf("\n");
                }
                chunk = chunk + (size/sizeof(int));
        }
}

void main()
{
        char *a1, *a2, *a3, *a4, *a5, *a6, *a7, *a8, *a9;
        int i;

        printf("Mallocing\n");
        a1 = malloc(120);
        a2 = malloc(120);
        a3 = malloc(120);
        a4 = malloc(120);
        a5 = malloc(120);
        a6 = malloc(120);
        a7 = malloc(120);
        a8 = malloc(120);
        a9 = malloc(120);

        printf("Address of a1 is %08x\n", a1);
        printf("Address of a2 is %08x\n", a2);
        printf("Address of a3 is %08x\n", a3);
        printf("Address of a4 is %08x\n", a4);
        printf("Address of a5 is %08x\n", a5);
        printf("Address of a6 is %08x\n", a6);
        printf("Address of a7 is %08x\n", a7);
        printf("Address of a8 is %08x\n", a8);
        printf("Address of a9 is %08x\n", a9);

//      traverse_heap(a1);

        printf("Freeing alternate mallocs\n");
        free(a2);
        free(a4);
        free(a6);
        free(a8);

        traverse_heap(a1);

        printf("New allocation done at : %08x\n", malloc(120));
        traverse_heap(a1);

}
"t2.c" 69L, 1401C written                                   65,0-1        Bot
```

```
a9: @chunk 0804b808, size : 120, prev_inuse:0
nptel@osboxes:~/nptel_codes/heapoverflow$ make clean; make
rm -f t0 t1 t2
gcc -g -m32 -w t0.c -o t0
gcc -g -m32 -w t1.c -o t1
gcc -g -m32 -w t2.c -o t2
nptel@osboxes:~/nptel_codes/heapoverflow$ ./t2
Mallocing
Address of a1 is 0804b410
Address of a2 is 0804b490
Address of a3 is 0804b510
Address of a4 is 0804b590
Address of a5 is 0804b610
Address of a6 is 0804b690
Address of a7 is 0804b710
Address of a8 is 0804b790
Address of a9 is 0804b810
Freeing alternate mallocs
a1: @chunk 0804b408, size : 120, prev_inuse:1
a2: @chunk 0804b488, size : 120, prev_inuse:1 Priv_Ptr: f7fb87b0 Next_Ptr: 0804b588
a3: @chunk 0804b508, size : 120, prev_inuse:0
a4: @chunk 0804b588, size : 120, prev_inuse:1 Priv_Ptr: 0804b488 Next_Ptr: 0804b688
a5: @chunk 0804b608, size : 120, prev_inuse:0
a6: @chunk 0804b688, size : 120, prev_inuse:1 Priv_Ptr: 0804b588 Next_Ptr: 0804b788
a7: @chunk 0804b708, size : 120, prev_inuse:0
a8: @chunk 0804b788, size : 120, prev_inuse:1 Priv_Ptr: 0804b688 Next_Ptr: f7fb87b0
a9: @chunk 0804b808, size : 120, prev_inuse:0
New allocation done at : 0804b490
a1: @chunk 0804b408, size : 120, prev_inuse:1
a2: @chunk 0804b488, size : 120, prev_inuse:1
a3: @chunk 0804b508, size : 120, prev_inuse:1
a4: @chunk 0804b588, size : 120, prev_inuse:1 Priv_Ptr: f7fb87b0 Next_Ptr: 0804b688
a5: @chunk 0804b608, size : 120, prev_inuse:0
a6: @chunk 0804b688, size : 120, prev_inuse:1 Priv_Ptr: 0804b588 Next_Ptr: 0804b788
a7: @chunk 0804b708, size : 120, prev_inuse:0
a8: @chunk 0804b788, size : 120, prev_inuse:1 Priv_Ptr: 0804b688 Next_Ptr: f7fb87b0
a9: @chunk 0804b808, size : 120, prev_inuse:0
nptel@osboxes:~/nptel_codes/heapoverflow$
```

Now what we do again is that we would request for memory to be allocated again with this call to malloc and what we would see is that since the linked list is available and the link list is not empty. So pt malloc would in fact remove an free list chunk of memory present in this list and allocate this chunk of memory to this malloc request. So what we will do is remove the exit from here and let the program run to completion and we would see how the metadata stored in the malloc changes.

So this corresponds to the first traverse heap over here with 4 elements in the linked list of free memory chunks. Now with this allocation and other 120 bytes what pt malloc has done is used one of these chunks which are in the free list and therefore our free list now just comes down to having just three memory chunks present. Also note that the memory that gets allocated corresponds to the recently freed a 2. So for example a 2 was pointing to this location 0804B490 and it was freed and the new memory request was also given exactly the same memory chunk therefore this new request and a 2 point to the same chunk of memory, thank you.