Information Security - 5 - Secure Systems Engineering Professor Chester Rebeiro Indian Institute of Technology, Madras Heap_Demo1

Hello, welcome to this demonstration, so this demonstration is on heaps, we have seen in the previous lectures about how we could overflow the stack and be able to do malicious things by essentially modifying the return address which is stored on the stack, we can also do something very similar by overflowing heap locations. So this very basic introduction to a heap exploit would first take us through how a heap is organized in the program and then we would actually use the exploit.

(Refer Slide Time: 0:52)



We will be sharing this particular code so you could download this code from your virtual machine box and run this code after compiling it, so in order to run this specific code you could do a make as make clean over here, then make. S what we see here is that there are in fact 3 programs T 0 dot C, T 1 dot C and T 2 dot C in this specific video we will be looking at T 0 dot C.

So let us open it out T 0 dot C, okay so this essentially is a very small C program where we have two malloc statements that gets invoked, one to allocate a structure data underscore T which is of 64 bytes and has name, the other one is F which essentially allocates in the heap this structure fp underscored T, so note that FB underscore T has just one element which is a function pointer fp.

So then what we do is we initialize the function pointer in f to nowinner so know that nowinner is here and it simply prints level has not been passed. Now there is another printf and then a string copy, now string copy uses d name that is d name and copies argument 1 that is a command line argument into d name and then there is a invocation to ffp, so what you would expect over here is first the string copy gets invoked and then the fp function which is pointing to nowinner that would get invoked and you would get this particular printf getting executed that is level has not been passed.







So let us see that happening so we run T 0 and give it a few inputs like this a few a short length argument and what we see is that it gets printed that level has not been passed. Okay, so before we go into how to actually use the exploit and by now if you have been following the course then you would have figured out that the exploit is due to this vulnerability in the string copy, note that the string copy copies something from argument 1 into d name so d name is a fixed length of 64 bytes and argument 1 is controlled by the user through the command line arguments, so the argument 1 can cause d name to overflow if it has a length which is greater than 64 bytes.

So let us just go through how this program executes with the right input first, so we will as you shall run gdb with T 0 as input list thing as usual put a breakpoint in line number 34 and we run this program giving the same shot input string, there are five to six A over here. So what we have stopped as we have seen before in the previous videos is that we have stopped at line number 34 and importantly right now there is no malloc has been called as yet.

As we have seen in the theory classes since the malloc has not been invoked as yet, the initial size of the heap is 0, in fact there is no heap present in this particular program at this particular point during the execution.

(Refer Slide Time: 5:02)



So we can see this by this info propmap command and what we look at over here as we seen in the theory is that we have the virtual address space for this process, we see the various segments in the process like which are defined front by the start address and the end address.

(Refer Slide Time: 5:22)



So we have the various segments for the T0 program we have the libc, stack, vdso and so on. So what is missing here you would notice is that at this particular time since we have not execute any malloc as yet there is no heap that is present. So we will single step to another line and see that the malloc has actually been executed, we can now run this info propmap again and we would see that a heap has now been assigned to the program. So what has happened internally is that rather since this was the first malloc that is invoked in this particular program the PT malloc code which has got linked with this particular program through libc has invoked the operating system and it has requested for a large chunk of memory of 132 kilobytes. So this memory has got attached to the locations 804b000 to 806c000 and the size is 132 kilobytes, so you can verify that this size 21000 which is in hexadecimal notation over here is in fact 132 kilobytes.

(Refer Slide Time: 6:38)



So now we will also look at what the address of d is xd and what we see is that d has a value 804b008. So what PT malloc has done internally is that since you have invoked the function requesting a structure data underscore T to be allocated in the heap, so PT malloc code once it has obtained the 132 kilobytes of memory would split this memory into two components, one component which would be slightly larger than 64 bytes because data underscore T is 64 bytes and the remaining component is the free chunk of memory which has not yet been utilized, what we see over here is the address of d we see that it is an offset of 804b008.

Now if you compare this with a start location of heap, you see that it is 8 bytes from the starting location of the heap, the bytes 804b000 to 804007 contains the metadata for this particular chunk of memory.

(Refer Slide Time: 8:00)



Now we can look at this metadata by executing something like this we are just dumping the heap location and we could specify the address 0x804b000, so in this particular time we have the metadata which will be stored over here currently it has a value of 49, so 49 is in hexadecimal 01001001, the LSB of this is 1 indicating that the previous is in use and 100 would indicate the size that is allocated for this chunk so this would be slightly greater than 64 bytes and this you could actually validate later, okay.

(Refer Slide Time: 9:06)



So once we single step further we would also see that the f would get allocated, so we look at the address part f e slash x or to obtain the thing for f and we would see that f is at a location

804b050, so note the offset of f from the base of the heap and we will compute actually what happens internally.

(Refer Slide Time: 9:30)

🜠 syste, na Russing) - Oacle Wi Vistuelles		- u x
File Machine Yeek Hope Devices Help Cellification		ta CE 4E SSEAM Ø
000 Methodenek-Jupid_indexponenties	000 rptnl@oubares: -/nptnl_codes/haspoverflow	
Ainclude exection by	0xf7f59000 0xf7f5c000 0x3000 0x0 0xf7f64000 0xf7f65000 0x1000 0x0	
#Include stdlo.hs	0xf7f65000 0xf7f68000 0x3000 0x8 (vvar) 0xf7f68000 0xf7f64000 0x2000 0x8 (vdso)	
ericcuse «sys/types.m»	0xf7fGa000 0xf7ffC000 0x22000 0x8 /11b32/16-2,23,50 0xf7ffc000 0xf7ff6000 0x1000 0x2000 /11b32/16-2,23,50	
NOPTINE PLACELIZE, NOX 64	0x77ff0000 0x77ff0000 0x1000 0x2000 /11b32/16-2.23.50 0xfff00000 0xffff0000 0x21000 0x8 (stack)	
(her risg(rissiz_wa))	(gdb) p/x d 51 = 0x8040408	
char nane[64];	(p0) x/32x 0x8040000 0x8040000: 0x00000000 0x0000049 0x00000000 0x00000000	
	0x80400101 0x80000000 0x80000000 0x80000000 0x80000000 0x80000000	
int (+fp)();	0x8040030 0x0000000 0x0000000 0x0000000 0x000000	
() with with the state of the s	0x2040000 0x0000000 0x0000000 0x0000000 0x000000	
printf("Heap Overflow Successfull\n");	(ph) s 17 fuite = nuclear()	
	(p0) p/x f 2 = 0x800000	
a. ((ph) [
printf("level has not been passed(n");	10-0.001-8 + 2048008	
(int main(int argc, char **argv)	8048008-40 + 8048048	
struct data_t *d; struct fp_t *f;		
<pre>d = nalloc(sizeof(struct data t)); f = nalloc(sizeof(struct fp_t));</pre>	8048048	
f->fp = now(sner;	Hexadecimal 1001130110x=134525000x	
printf("data is at Np, fp is at Np\n", d, f);	0 000 0000 0000 0000 0000 0000 0000 0000	
strcpy(d->name, argv[1]);	0 000 1000 0000 0100 1011 0000 0100 1000 31 15 0	
f->fp();	48 18 . XV () XV XV A	
	C D E F = mod ones twos (4	
	B S A B + MO NOT 4 x1	
	4 5 6 7 - 08 G log in lot	
÷	0 1 2 3 + X08 + fet # fet	
10.6 446, 595C		and the second second
	8 9	
		PART 0 PASH

So starting from the base so we first move this thing into the programming mode and move to the hexadecimal notation and let us start out from the base and we see that the base of the heap is 804b000 and the initial eight bytes comprises of the metadata for the first memory allocation that is for d that is so it would be 8 bytes so 804B008 gives you the memory address for d.

Now since the size of struct data underscore T is 64 bytes, so therefore the PT malloc would have added 64 bytes to this, so 64 in hexadecimal notation is 40, so this is 804B048, so this location onwards signifies the end of d.

(Refer Slide Time: 10:41)

🜠 systel, mir (Running) - Oracia VM Vistualliss		- 0 X
	CO: appliqueness: -type1_cudex/happenting bc/7710008 bc/70008 bc/8 bc/770008 bc/70008 bc/8 bc/770008 bc/70008 bc/8 bc/77008 bc/70008 bc/9 bc/77008 bc/900 bc/900 bc/900 bc/900 bc/900 <	¶, ∰ 40 600A4 Q
interfug(tus)(t_m(s)) interfug(tus)(t_m(s)) interfug(tus) interfu		
wid sendors:) printf('Irest has not been passef(n'); printf('Irest has not been passef(n'); in sets(int args, char "segs)	(20)	
	Monadelimit HUTURIDA = 13402000a 0 20 0400 0400 0400 0400 0400 0400 0400	
	C D E F x mol mol <thmol< th=""> <thmol< th=""> <t< td=""><td></td></t<></thmol<></thmol<>	
10.0 10.0 <td< td=""><td>2 7 1 1</td><td></td></td<>	2 7 1 1	

Now the next allocation is f, now f also has metadata the it has two words of metadata and therefore it will also have 8 bytes of metadata. So if we add 8 to this, we will get 804B050 which precisely is what is the address of f. So in this way what we see is that we have seen how the memory is organized at this particular point in time, it has contiguous chunks of d and f which has placed side by side and the only thing which separates them is some metadata information for the f.

(Refer Slide Time: 11:47)

								મ 🖬 લ
000 Aptel@osbekes:-/sptel_codes/belpoverflave	000	ngt el @osi	baxes: -/n	tel_codes	heapoverflow			
electude estdib.ho	0×8045000	91	0x000000	0 0	00000019	8x41414141	8x88884141	
Selectude estring.ha	6x804001	81	0100000000		00000000	6+00000000	6x0000000	
#include «stdio.h»	0x8040030	91	0×0000000	0 0	.00000000	8×86666668	0×0000000	
einclude «sys/types.n»	0x8040041	0; 6:	0x0000000		00000000	6x00000000	0x00000011	
#define FLAGSIZE_MAX 64	0x804006	0	8×6174614	4 8	20736920	8x30207461	0x34303878	
cher flasfflagstr Bax1:	0x804b071	01	0x3830304	2 0	(7866282C	8x28736928	0×30207461	
con collicourses with	53 = Vala	ue can't	The conve	rted to 1	Integer.			
struct data_t ((gtb) dL	Lassenbl	e nowlane	e				
))	0400 07 0	assemble 484b4 ca	# code re	with Kel	on nowinner:			
and to t I	0x0804	484b5 <+	1>:	ov Ke	ip,Xebp			
int (*fp)();	0x000	484b7 <+	3×1 1	10 50	(8,%esp			
3;	0x000	484bd <+	9>1	ush St	180485ea			
vold street()	0x000	48422 44	(14) (19) (all 8x	1848378 <put 18. Xeco</put 	uiplt.		
	0x000	484ca ci	22+1 1					
printf("Heap Overflow Successfull(n"))	0x000	484cb <4	23+1 1	eave				
	End of_a	ssembler	dump.	*1				
vold now(neer()	(gtb)	_						
printf("level has not been passed\n");	0.0 Calco	later -						
()	8040-000+8					048008		
	8048008+40					048048		
int main(int argc, char **argv)	8048048+8							
int main(int argc, char **argv) {	8040048+8							
<pre>int main(int argc, char **argv) { struct data_t *d; struct fp_t *f; }</pre>	8040048+8							
<pre>int meln(int argc, char **argv) (struct data_t *d; struct for *f; d_s = sloc(slogef(struct data *1); d_s = sloc(s</pre>	8040046+8					8048050		
<pre>int main(int argc, char **argv) { struct data_t *d; struct fp_t *f; d = malloc(sizeof(struct data_t)); f = malloc(sizeof(struct fp_t)); </pre>	8040048+8					8048050		
<pre>ts main(int argc, cher "margo) (struct data_t = d; struct data_t = d; struct fata_t = d; d = nallec(sizeof(struct data_t)); f = nallec(sizeof(struct fp_t)); fds = nallec(sizeof(struct fp_t)); fds = namles(sizeof(struct fp_t)); fds = namles(sizeo</pre>	8040046+8	4.5			10011000	8048050		
<pre>tet main(int argc, char "mapp) (trived data_t_="4") sinut fp_i = "fg" sinut fp_i = "fg" f = mailCoc(treef(intext data_t)); f = mailCoc(treef(intext data_t)); f = mailCoc(treef(intext fp_i)); f = formationer; }</pre>	Hexadecinu	el ¥			10011301	8048050 20. = 134525008.=		
<pre>tan mata(tint args, class "Margo) (stored data_1 * 4; stored data_1 * 4;</pre>	Hexadecime 0 0	al ¥ 200 000	0 0000 0	000 0000	10011301	8048050 20. = 134525000 = 000 0		
<pre>tin mata(int arg, char "maps) (reme that a "di; since fp_1" "fi d = malles(class(since(inter(targ, t)); f = malles(inter(inter(targ, t)); f = malles(inter(inter(targ, t)); f = malles(inter(targ, t)); f = malles(inter(targ, t)); f = malles(t); print("data is at up, fp is at up(n", d, f); strongd=maps, mp(1)); trongd=maps, mp(1); inter(targ, targ, t)); inter(targ, targ, t); inter(targ, t); inter(targ, targ, t); inter(targ, targ, t); inter(targ, t); inter</pre>	Hexadecima 0 0 63 0 0	al + 000 000	0 0000 0	000 0000 47 100 1011	10011301 0.0000 0.000 1.0000 0.101	8048050 20. = 134525008= 000 0 32 000 0		
<pre>tex mac(t) #pc, chr "map) struct #pc, "#p; d ~ all(c(lase('))(c(lase(t, b))); f ~ all(c(lase('))); f ~ all(c(lase('))); f ~ for a struct (b, b); f ~ fo</pre>	0040048-8 Hexadecima 0 0 63 0 0 31	al * 200 000	0 0000 0	000 0000 47 100 1 011 15	10011301 0 0000 0000 1 0000 0101	8048050 20. = 134525008= 000 0 32 000 0 0		
<pre>tix mad(t) #pc, chr "mp() stret fm, "4; stret fm,"*f; d = wilke(tive(stret dem.t)); f = wilk(tive(tive(stret dem.t)); f = wilk(tive(tive(tive) fm, 0)); f-fp = maximer; print('dem is at Np, fp is at Np(n', d, f); strengt-ennem.exp(1)); f-fp(); </pre>	0000048-8 Hexadecima 0 0 63 0 0 21 4 n 7 n	al * 200 0000 200 1000	0 0000 0 0 0000 0 x v	000 0 000 47 100 1 011 15 ()	10011301 0 0000 0000 1 0000 0101 1 v v v	8048050 20. = 134525008+ 000 0 32 000 0 4		
<pre>tax made(t) #pc, chw "map) struct #pc, "#p" struct #pc,"*f; d ~ all(c(t)#p("(t)*f(t)*f(t)); f ~ salle((t)#p("(t)*t); f(t,1)); f ~ for a construct; f(t,1)); f ~ for a construct; f(t,1); f ~ for a con</pre>	Hexadecins 0 6 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	al * 000 0000 000 1000	0 0000 0 0 0000 0 x v	000 0 000 47 100 1 0 11 15 ()	10011301 0 0000 0000 1 0000 0101 4 V > V	8048050 20.+134525008+ 0000 32 0000 0 4		
<pre>tin mac(ti #pc, chw "map) (street fm, "4; street fm," 4; d = wallec(tion((street deg.l)); f = wallec</pre>	Hexadecim 0 0 0 0 0 1 1 1 1 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	al * 000 0000 000 1000	0 0000 0 0 0000 0 X * F	000 0 000 47 100 1 0 11 15 () s mod	10011301 0 0000 0000 1 0000 0101 4 ¥ ¥ ¥ ¥ ones two	8048050 20. = 134525008+ 0000 32 0000 0 4 14		
<pre>tax made(t) #pc, chw "map) struct #pc, "#p" struct #pc," #p; d = all(c(tax#(f)(tax#(d,tax))); f - salle((tax#(f)(tax)(d,tax))); f - for a manage(tax)(f)(tax)(f)(tax)); f - for a manage(tax)(f)(</pre>	8040041-8 Headedin 0 0 63 0 0 31 4n 1 5 0 0 0 8 31 4n 1 5 9 9 9 9	al * 000 0000 000 1000 0 0 0 0	0 0000 0 0 0000 0 X V F B	000 0 000 47 100 1 0 1 1 15 () + mod + AND	10011301 0 0000 0000 1 0000 0101 1 0000 0101 1 0000 0101 1 0000 0101	8048050 20x+134525006+ 0000 32 0000 0 4 14 14 14 14 14		
<pre>tin mac(t) #pc, chr "mpp) stret dat, '4(; stret fp, '4'; stret fp, '4'; d = ma(ter;(inse(time(inset dat, 1)); f = ma(ter; print('dat is st Np, fp is at Np/n', d, f); stret('dat is st Np, fp is at Np/n', d, f); strep(datass, mpp(1)); f = fp(); }</pre>	000001-0 Hexadecinx 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		0 0000 0 0 0000 0 1 * * F B 7	000 0000 47 100 1 011 15 () = mod + AND - OR	10011301 0000 0000 0000 0101 0000 0101 0000 0101 0000 0101 0000 0101 0000 0101	8048050 20x+134525006+ 0000 32 0000 0 4 14 14 14 15 14 10 10 14		
<pre>tax made(t) #pc, chw "map) struct #pc, "Af set(class()); f = del(class()); f = del(class(</pre>	Headedmin Headedmin 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		0 0000 0 0 0000 0 X * F B 7	000 0 000 47 100 1 0 1 1 15 1 15 10 1 15 10 1 15 10 1 15 10 10 10 11 11 15 10 10 10 11 11 15 10 10 10 10 10 10 10 10 10 10 10 10 10	10011301 0 0000 0000 0000 0101 0 0000 0101 0 000 0101 0 000 0101 0 000 0101 0 000 0101	8048050 86+13452508+ 000 32 00 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		
<pre>tin mac(ti #pc, chw "map) store data, '4(; store fp_n'*f; d = ma(tec(timef(true(data, 1)); f = ma(tec(timef(true(data, 1)); f = ma(tec); print('data is at Np, fp is at Np(n', d, f); stored(d-same, map(1)); f = fp(); }</pre>	Headedmin Headedmin 0 1 0 1 0 1 0 1 0 1 0 1		0 0000 0 0 0000 0 X * F B 7 3	000 0 000 47 100 1 0 11 15 10 10 mod * AND * AND * XOR	10011301 0	8048050 20x = 134525000x 4000 0 4 14 14 1 10 11 11 11 11 11 11 11 11		

Now we will see how we can actually overflow d using this string copy and passing an argument which is longer and how we could actually modify the contents of f. So first of all let us continue to a single step and since we have given a small input we can go through

string copy and we can now look at the heap again and what we see is that since our input is all is a couple of A's and A has ASCII value of 41 in hexadecimal, so those values are actually stored from this location onwards.

Now since data underscore t has a size of 64, so 64 bytes from here would end the structure data underscore t and then we would have f and f since we have initialized to no winner the value of no winner is present here. So we can see that 080484B4 in fact specifies the address of nowinner so that we can check like this v / x nowinner is assemble nowinner and what you would see is the start address of nowinner is indeed 080484B4.

(Refer Slide Time: 13:22)

	Popul Devices Perip													1. 🖾 4
			5 () au	el@ash			ender D		rfing					
	eloclude «stdlib.b»		*****	405 =+1	*1	nov	Xes	, xebp						
- 10	Finclude sumistd.hs		x08045	4b7 <+3	×1	sub	58xi	, tesp						
	Finclude «stdlo.h»		1x08048	4bd cr9	*1 *1	push	50x1	18485e						
	#include «sys/types.h»		x08048	462 ++1	de l	call	0x80	48370	quts	plt>				
	#define FLAGSIZE_MAX 64		1×08048 1×08048	4C7 ++1 4C8 ++2	9+1 2+1	800 NIQ	58X	ie,ses	p					
	cher flasfflactin sax)		******	Ach erz	3+1	Leave								
	con real constrained.	04	of ass	enbler	er: dump.	ret								
	struct data_t ((98) disa	ssenble	winner									
))		-0000		×1	push	Keby					1		
	struct fp.t (X08048	49c <+1	>1 >1	nov	Ness Seat	, Kebp						
	int (*fp)();		*****	401 <+6	*	sub	50xi	,Xesp						
	11		1×00045 1×00045	424 c+9 429 c+1	ini dia t	push call	50x8	10485 <i>0</i> 148378	0 100/15	mlt.				
	vold winner()		×08045	tae <+1	9+1	add	58x1	l0,%es		pre-				
	printf("Heap Overflow Successfullin");		x08048 x08048	401 <+2 402 <+2	Zini Bini	nop Leave								
)		×08048	463 ++2	4+1	ret								
	vold nowinner()	(98	of #55	enoter	ounp.									
	<pre>printf("level has not been passed(n");</pre>	0.0												
)	80-60	00+8						+ 80	48008				
	int main(int argc, char **argv)	80400	08+40 48+8							48048				
	attent data t td:													
	struct fp_t *f;										_			
	<pre>d = malloc(sizeof(struct data t));</pre>									8048	1050			
	f = malloc(sizeof(sizeof(sizeof(size)))													
	f->fp = nowinner;	Head	decimal					10	91130120	h=13452	5008-0			
	printf("data is at No. fo is at Noin", d. f):		0.05	0 0000	0000 1	0000	0 0 0 0	0000	00000	000				
	stand and section.		60				47			32				
	stropy(s-seame, args[1]);		31	0.1000		0100	15	4444	01011	0				
	f->fp();	1 m	76		3.9	(e ¥	3.4	4				
	þ	6	0				mod	-	Parts	M				
			2	A.	-		AND	NOT	4	2	12			
		4	5	6	7		OR	0	log	In	int			
			1	1	1		X08		fact		Inc			
			للفعار	الشا	1			ينت ا	1	الكبا	-			
	"10.C" 46L, 593C													

Now what we want to do is that we want to modify this program we want to subvert the execution so that this address in the heap which corresponds to nowinner is modified to that of winner. In order to do this we would first require to determine the address of winner so that would can be obtained by disassemble of the winner function and you would see that the winner function starts at the address 0804849B, so we could actually write this down somewhere.

(Refer Slide Time: 13:41)

nptet,ms/Running) - Oncle VM VetraBox		- U X
n Machina Yani ayut Devini Yang ayniladi		T≱ 🖬 €€ 603AM 0
eptel@odusies -/sptel_cedes/hespoverflow	a a a notelpostenen: -/notel_coden/haspoverflow	
Pinclude +unistd.h>	(pm) s printf("data is at Np, fp is at Npin", d, f);	
Pinclude stifte.hs	(pm) s	
Finclude «sys/types.h»	<pre>41 strcpy(d->name, argv[1]);</pre>	
Indefine FLAGSIZE_MAX 64	(pm) s 43 f→fp();	
char flag(FLAGSIZE_MAX))	(p/b) x/32x 0x8040000 0x8040000: 0x00000000 0x00000049 0x41414141 0x00004141	
ETruct data_t ((bur hame[64])	4x10430101 0x10000000 0x10000000 0x10000000 0x10000000 0x1040x101 0x10000000 0x10000000 0x0000000 0x00000000	
1 In the second second	8x8040401 8x8000000 8x8000000 8x8000000 8x800000011	
struct fp_t { int (+fp)();	bits Disconting Disconting <td></td>	
void winner()	(gdb) p/x nowinner 31 = Value can't be converted to integer. (gdb) disastemble nowinner	
printf("Heap Overflow Soccessfull(\n");	Dunp of assembler code for function nowinner: Bx88048864 «H»: push Kebp Bx8804885 «H»! now Kebp Kebp	
a. (0x0004807 <<3>: sub 50x5, kep 0x0044ba <<6>: sub 50x5, kep	
printf("level has not been passed(n");	D D CALORIN	
int main(int argc, char **argv)	0000004-0 + 8048048 8048048-4 + 80480 - 8048048 8048048-4 + 80480 - 804800 - 804800 - 804800 - 804800 - 804800 - 80480	
struct data_t *d; struct fp_t *f;	Be80048496 start of winner function	
<pre>d = nalloc(sizeof(struct data_t)); f = nalloc(sizeof(struct fp_t));</pre>	6	
f->fp = nowinner;	Hexadecimal * 1001130120-*	
printf("date is at Np. fp is at Npin", d. f);	0 000 0000 0000 0000 0000 0000 0000	
strepuld-seame, arov[1]);	6) 47 0 000 1000 0000 0103 1 011 0000 0101 000	
	31 15	
1. Martin	48 TR , X V () < V > V	
	C D E F e mod ones twos	
	B 9 A B + AND NOT + 1	
	4 5 4 7 - 08 G Ing E	
	0 1 2 3 + 100 + fed	
10. c* 66. 1996	- TREAT	2.1 411
	·· Deseries	
🖸 O Type here to search 🕴 🖻 🛤 🚳 🐰 🚺	2 7	~ 0 00 A 40 M

So what we have done is we have copied the start address of the winner function I know that we need the start address to override the nowinner function location present in the stack. So let us see how we do this from here things are quite similar to what we have done so far in the past in this particular course, we can first determine that the amount of in order to overflow this particular data underscore t structure we would require 72 inputs before all of this gets filled up and then we need to specify the address for our malicious function which in this case is winner.

(Refer Slide Time: 14:32)

Machine Yest	Input Devices Help											
inal											h	46 606
						Bonbaxe	s -/ngte					
	finclude «stdlib.h»			End of a	assent	iler dur	φ.	_				
	Finclude «unistd.h»			(gdb) d	154554	omble wi	nner					
	Finclude «stdio.h»			ourg of	#15#7	10147 CO	of for	Function Net	n winn	er 1		
	Finclude «sys/types.h»			0x08	04549	C +1>1	- 104	Tes	o,Xebo			
	Adefine FLACSIZE_MAX 64			0x00/ 0x00/	048494 0484a1	(43×) (46×)	sub	58x 58x	ill,Xesp ic,Xesp			
	char flag(FLAGSIZE BAX);			0x00	040424	t crypt]	pus	h 50x	804830		and the	
	struct data_t (0x00 0x00	0484a4 0484b1	++19+1 ++22+1	add	584	10, Nes	· ·	apro-	
	char name[04]1			0x08	6484b3	e+23+:	lei	we				
	struct fp_t (End of a	assent	oler due	φ. ^{ret}					
	int (*fp)();			A debug	ging s	session	ts acts	we.				
); vold winner()				Infer	ter 1 (process	15635]	w111	be k111	lled.	
	(Qu'it any	yway?	(y or a) y				1	
	printf("Heap Overflow Succes	aretriv.)		setel@o	(beite)	:-/npte	1_codes	/heapov	erflow	5 ./t0	0 \$(./payload2)	
	,			HEAD OV	at or erflow	 Succes 	aful!		040030			
	vold newinner()			apte ipe		1-/npte	1_codes	heapov	erflow			
	printf("level has not been a	ussed(n*):										
)		- 6	1040-000+8						+ 80	1048008	
	int mainful area, that tharea)		- 1	1048008+40	6					+ 80	1048048	
	(- 1	1040045+6						* 80	0.0.0 aptel@osboxes/vptel_coles/heapoverflow	
	struct data_t *d;		- 1								2x0004649b start of winner function	
	struct fp_t *r;		- 1									
	d = malloc(sizeof(struct dat	((());	- 1								C .	
	f = Malloc(streef(struct fp,	,000										
	f-stp = nowlenes		- 1	Hexadecie	nal 🕶				10	1130120	20	
	asiatilitate is at the fe is	at mint of the	- 1									
	proversi entre to et sp, sp ti	at 100 1 4, 171	- 1	- 63				47				
	<pre>strcpy(d->name, argv[1]);</pre>		- 1	0	000 1	1000.00	00 010	0 1011	0000	0101 0	664_	
	f. sfull:											
				10 1	۰.	· X	۷ (₹ ₩	2.4	JUC	
)		- 1	c 1		1 1	1.	mod	0.045	Parts		
			- 1									
				1 1		A 8	1.1	AND	NOT	1	1.4	
			- 1	4 1	1	6 7		08	0	les	1	
	-		- 1									
			- 1	0		1)		XOR		fect	J.L.*	
				_	-		-	-	-			
	"10.c" 46L, 595C	1,1	All								INLERT 2.	1

So we have already done this for you, so this is present in payload 2, so you see over here that what payload 2 does is that it creates 72 A's followed by the address 0804849B so this as you

can recollect is the little Indian notation. So we could run this particular program as follows, so we have simplified it for you all you all that you need to do is run T0 specify a dollar and then dot slash payload 2 and this would result in this payload 2 getting executed the string 72 A's followed by this address of winner would be then passed to T0 and we see that it has resulted in a heap overflow successful. So this happens because the winner function gets executed.

So what we have done is that we have essentially overflowed d with a lot of A's in fact 72 A's and then specified the address of the nowinner function which essentially replaces this invocation this fp thing with winner thus the winner function would get executed. So there are a lot of interesting attacks you can actually do with the heat, but the and some of them are actually uploaded to the website for this particular course and in the next video lectures we look at some more things specifically with the heap, thank you.