Daniel Oliveros

Garrett Bogart

Spring 2018 – Independent Study

Dirty COW Vulnerability Lab

**Background:**

This vulnerability existed in the Linux kernel for almost a decade, it seems it was part of the Linux distribution since the year 2007 and was only really patched out up until November 2017.

This vulnerability was also present on Android devices, and could allow rooting of these devices as well.

The way this vulnerability works is by exploiting a race condition found within the kernel’s memory-management subsystem.

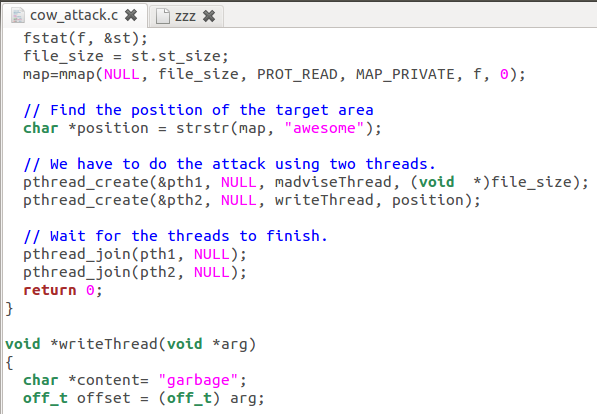
Other possible applications of this exploit include modifying binary files for applications like the Shell, such that whenever this was launched, malicious programs like keyloggers would be launched alongside them, allowing the attacker to gain valuable data from their victims.

**Task 1:**

We first created the file zzz in the home directory. This is what the file contained



The code we used to modify it can be seen below. For simplicity’s sake we changed the word “awesome” to the word “garbage” which has a similar length.

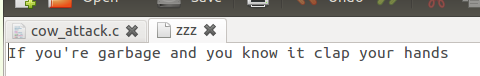


When compiling the program, use the -pthread command to inform gcc that the program is making use of these threads. After this, running the program should be very easy.

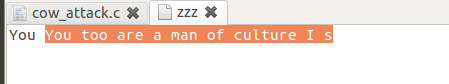


This program managed to perform the Dirty Copy On Write in less than 2 seconds. It is honestly impressive just how efficient it is at exploiting this vulnerability, and scary to realize it existed for so long.

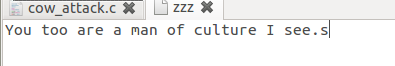
Here is the file after running this program for a short period of time.



An important thing to note is that the file that is being overwritten has a fixed size. In another test the code was modified to overwrite “You” with “You to are a man of culture I see.”.



The after image of the attack reveals that the attack only overwrites the file.



To show the limits of the attacks I modified the code to be

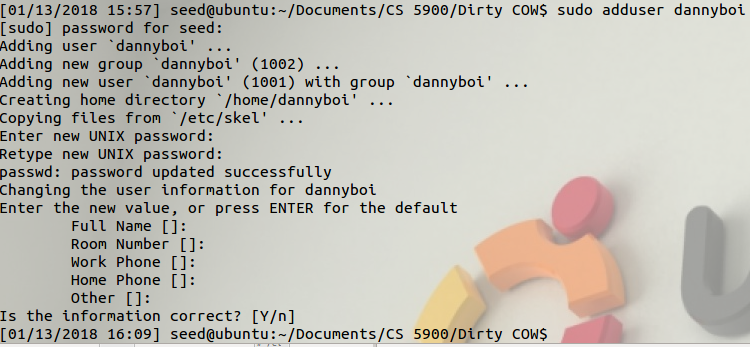


The file is of the same length as before but not all of the “\*” were added to the file.

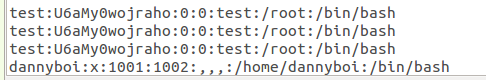


**Task 2:**

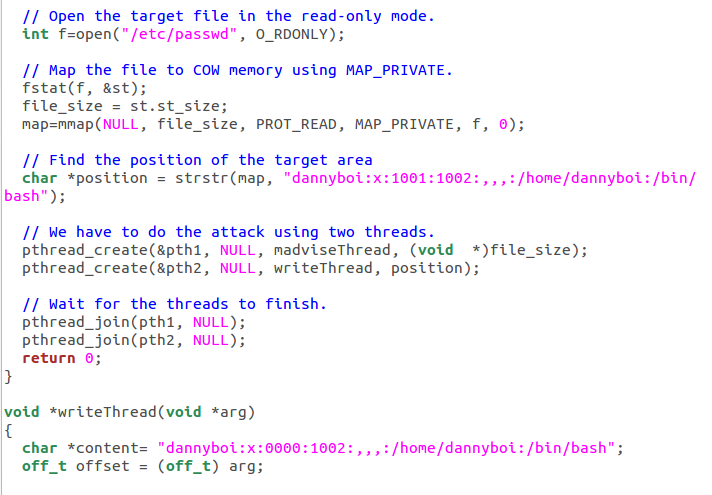
We begin by adding a new user, this user will end up having a very high uid, 1001 to be exact. They should have no power to modify root-level data. Here is the process of creating said user:



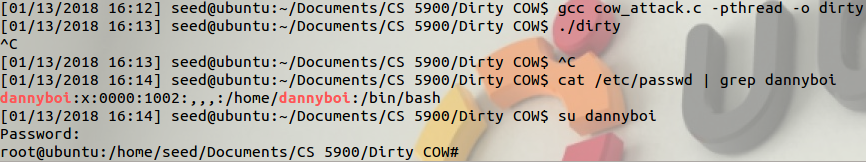
The user is stored in the passwd file like this. Notice that the third value is 1001, this is where the user’s ID number is stored. If we change this down to 0, we can gain root access when logging in as this user.



We made some changes to the code in order to accomplish this. In order to ensure nothing else gets touched we use the entire entry as the string to look for and proceed to replace it with its slightly modified equivalent. This caused some minor concerns since we were looking to exploit a narrow window, and adding extra characters caused minor runtime worries, where our program could perhaps get locked out halfway and somehow end up corrupting the file. This did not happen, so luckily it didn’t end up mattering.



If we were to change the 1001 to 0 we would be left with some nonsensical garbage at the end of our string. By changing 1001 to 0000 we preserve the original string length. After running the attack we checked the changed file again. We then checked to see if we had root access.



We successfully were able to login into our user with root access.

**Issues:**

1. This lab, although effective, felt too short to justify being a whole lab on its own, and it also felt like it guided you by the hand the entire time. The only coding sections of it really boiled down to changing some strings to fit what we wanted it to do.

**Ideas for Improvement:**

1. This lab would be good to have as a short follow up to the Race Condition Lab, it took probably about an hour of actual work to finish, and provides valuable insight into more recent attacks as well as well-structured exploits for race conditions