CEG 4420 5: Rootkit

Daniel Kosey
CEG442001
u00448953
kosey.2@wright.edu

Introduction

This is the 5th lab in CEG 4420, focus on Rootkits.

Purpose of the Experiment

To become familiar with rootkits and the tools to remove them.

Experiment Setup

To perform this lab, two computers running KNOPPIX were used. A third, running Backtrack, was used for internet access, isolated from the other two. The computers used were the three in the back corner of the room- default IP's ending with 19, 14, and 16- with 16 being the computer in backtrack. Rootkits and rootkit cleaning kits were found using the internet and moved to IP 14 (henceforth M1) using a flash drive. Exact programs used are listed in the following sections. From here on, IP 19 (back corner of the lab) will be referred to as M2.

Experimental Procedures

The intended procedure for the lab is as follows (2):

1. Download onto a clean Linux machine (M1) the source code of two rootkit detection packages among many available on the net. See links below. Build the binaries.
2. Boot another machine (M2) into a Linux Live DVD. Corrupt this system with a rootkit. Setup a new user named "intruder".
3. Login from M1 to M2 as user "intruder". Use the machine M2. Be creative.
4. Which of the activity of user intruder are you (as root on M2) able to observe with standard Unix utilities? Re-read the section "Analysis of N.F.O Incident" above.
5. Download from M1 the check rootkit binaries. Run these. Write a report on how you would clean M2 up.
6. Suppose you are the attacker. What changes would you make to M2 so that these rootkit detection tools become ineffective if they were downloaded and built on M2 instead of M1?

In my case, I was unable to build and install any rootkit on M2.

I started by building the chkrootkit(2) and rkhunter(3) programs on M1. These were saved on the flash drive. I later discovered both utilities are available by default in KNOPPIX. Next I tried to build a rootkit named KBeast(4). I discovered that the rootkit would not build under KNOPPIX due to a check for linux kernel versions. I tried another rootkit, Jynx Kit (5), and was not able to build that either.

I decided to try and tweak KBeast to see if it would work. I modified 'setup' (shell script) to skip version checking (commented out function call). I also modified getdents() in ipsecs-kbeast-v1.c to use struct linux_dirent instead of struct dirent. I then modified /usr/src/linux-headers-64/include/linux/dirent.h to describe struct linux_dirent as well as struct linux_dirent_64 by copying the struct for x64 and changing name.

Later, I tried to use SuperKit, but was unable to build in KNOPPIX due to multiple issues. I tried to use Dica, as recommended by a classmate, but the result was underwhelming. Finally, I attempted to build Linux Root Kit V5 (LRK5). Again, I was unable to compile due to numerous build failures in compilation.

Recorded Observations

I was unable to successfully install a rootkit. When I tried KBeast, the result was a handful of compilation issues. This theme stayed with all of the attempted installations, except for Dica. When I used the install script in Dica, it “did its thing” without error checking. The result was a partial installation that left basic functions crippled or broken. For example; ls would produce an error on EVERY use and ps would result in an error indicating there was no /bin/ps.

At one point, I also glanced at a shell script that was labeled as rkhunter, and recognized the values it had in record for identifying an installation of KBeast- namely the /usr/_h4x_ folder and contained files.

Interpretation of Recorded Observations

The only rootkit I was able to make close to success with was Dica, and it failed to install properly. The installation resulted in a rootkit that very obviously revealed its presence, which defeats the reason for using a rootkit in the first place. If this was the intended result, this is a bad rootkit. I believe that the result was an improper installation, because it is unlikely that someone would write a rootkit to perform in the manner observed.

Remarks

In total, I used about 7 hours on this lab. I was able to learn about rootkits to some degree. In specific I was able to look at the internal of KBeast, which uses syscall table hooks to redirect the normal functions to the _h4x_ versions specified in the c code and installed in an insidious kernel module. The single greatest difficulty in this lab was building and installing a rootkit. I don’t know if it is possible for

someone with my limited understanding of linux systems to install a rootkit successfully on a modern OS given limited access to rootkit sources. In the future it may be better to require students to use an older OS, or to provide a more functional rootkit for use.

I'm not sure what section is best to record this, so I will record it here. Upon closer inspection, the lab procedure listed in the Rootkit lab does not include steps for all of the graded fields on the linked Grading sheet; namely the sections for logging into Ubuntu and for "making it real".

For both of these cases, I think the best solution is to use a known exploit for privilege escalation. Once root access is established, installation of an already developed rootkit should be trivial.

Conclusions

In conclusion, I was unable to complete the lab as directed, but I was able to study a rootkit in some detail, and I am familiar with the concept of how a rootkit would function.

References

1. Mateti, P. *Course Web Site.* Root Kit lab. http://www.cs.wright.edu/~pmateti/Courses/4420/RootKits/index.html
2. *Reznor.com*. Download mirror for chkrootkit. http://www.reznor.com/tools/#chkrootkit
3. *Sourceforge.com.* Download of rkhunter. http://rkhunter.sourceforge.net/
4. *Packetstormsecurity.org.*Download of KBeast. http://packetstormsecurity.org/files/download/108286/ipsecs-kbeast-v1.tar.gz
5. *Packetstormsecurity.org.*Download of Jynx-Kit. *http://packetstormsecurity.org/files/110942/Jynx-Kit-Release-2.html*