Rootkit Lab

Setup:

It is best to do this lab on a fresh VM since you will be hooking library calls which although unlikely, may potentially cause your VM to misbehave. Some debugging tips are indicated at the bottom of this lab handout to aid in weird behaviour that may result from hooking libc library calls.

1. Download and install a **fresh** Kali Linux VM. If you want to use an existing VM, make sure to **take a snapshot** of it (VMWare) (VirtualBox).
2. Clone the LD_PRELOAD rootkit: https://github.com/brandonjaipersaud/CSC427-LD_PRELOAD-Rootkit

Submission:

Submit a pdf with all your answers clearly labelled to MarkUs. Answer all **TODOs** and provide **SCREENSHOTS** where indicated.

Part 1: Exploiting LD_PRELOAD to create a backdoor using a reverse shell

In the demo, we showed how LD_PRELOAD can be exploited to create a backdoor into the victim machine using a *bind shell*. We will start by doing the same thing but instead use a reverse shell instead of a bind shell. Note that for simplicity, we do this on a single VM. In reality, the attacker and victim would both have their own machines.

Explain the difference between a bind and reverse shell. In a bind shell who (the attacker or the victim) is the server and who is the client? Likewise, in a reverse shell, who is the server and who is the client?

TODO:

1. Start by compiling the shared library rootkit by doing: `gcc rootkit.c -fPIC -shared -D_GNU_SOURCE -o write-hook.so.6 -ldl`

2. Next, move the compiled rootkit to a directory the linker uses to search for shared libraries: `sudo mv write-hook.so.6 /lib/aarch64-linux-gnu/`

   (this path may vary depending on the architecture you are using. In my case, aarch64 stands for ARM64 architecture. I suspect most of you would be using x86_64 architecture so it may look like: `/lib/x86_64-linux-gnu/` instead)

3. Now, we must make the ld.so.preload file point to our rootkit so it can be dynamically injected into all running processes. To do this, you must be the root user. Start by issuing the command: “su” and enter the root password to switch to the root user.

   Hint: If this doesn’t work you may need to do `sudo passwd root` to enable the root user account

4. Use the echo command to copy the path to your shared library into the ld.so.preload file.

   Hint1: If you are stuck, look in the slides to find where the ld.so.preload file is located (around slide 47)

   Hint2: `echo <something-here> > <something-here>`

   Hint3: The first <something-here> is a path string terminating in “/write-hook.so.6”

5. Verify that your rootkit (shared library) will be dynamically linked to all executables by executing: `ldd /bin/ls`. Paste a screenshot of the output of this below. You should see write-hook.so.6 in the list of shared libraries that /bin/ls depends on.

   PASTE SCREENSHOT HERE:

6. Now that our rootkit has been deployed, we can enter the victim system using the backdoor created by the rootkit.

   Ensure the sshd daemon is running by doing: `sudo systemctl status ssh`. If it is not active then do: `sudo systemctl start ssh`.

7. Open 2 terminals. One terminal (terminal 1) will be used for triggering the backdoor to the victim machine via a reverse shell. The other terminal (terminal 2) is what the reverse-shell on the victim machine will attempt to connect to. Gain entry to the backdoor on the victim machine by issuing the appropriate commands on terminals 1 and 2 respectively!

   Hint1: On terminal 1 do: `ssh <something-here>@localhost`
   On terminal 2 do: `nc <something-here>`

Hint2: The ordering in which you execute these commands matters. Think about what a reverse shell is and what makes sense here.

Hint3: Take a look at the code in rootkit.c, specifically the definitions at the top, and recall what we did in the demo.

Paste a screenshot of terminal 1 and terminal 2 to show you have successfully triggered the reverse shell and gained entry to the backdoor on the victim machine.

PASTE SCREENSHOT HERE:

Part 1.5: Rootkit Aftermath

Notice that your reverse shell is not hidden from netcat. Namely, the TCP connection the victim machine forms with the attacker machine is not hidden. Prove this by issuing the command: `netstat | grep "localhost"`

NETCAT OUTPUT SCREENSHOT HERE:

You should also notice that the ld.so.preload file does not appear when doing: `ls /etc`. However, we can still use "ls" to detect whether the ld.so.preload file exists. Show how we can do this:

PASTE COMMAND HERE (It is pretty straightforward. Not a trick question):

Part 2: Modifying the rootkit to hide from netstat

To hide from netstat, we can hook the fopen()/fopen64() library calls in our rootkit. The idea behind this is that netstat will call fopen()/fopen64() on /proc/net/tcp to read information about the system network connections which get printed to stdout. We can hook fopen()/fopen64() so that when it tries to read the tcp file, we filter out the TCP connection created by our reverse shell.

Fill in the TODOs in rootkit.c under fopen()/fopen64() to complete the hooks.

Hint: The TODOs you fill in for fopen() and fopen64() should be exactly the same with the exception of the first TODO.

Note: Please see debugging tips at the end of this document for help if you get stuck.

Verify that your hook works by following the same steps you did in Part 1:

1. Recompiling the rootkit: `gcc rootkit.c -fPIC -shared -D_GNU_SOURCE -o write-hook.so.6 -ldl`

2. Moving it to the shared libraries directory: `sudo mv write-hook.so.6 /lib/aarch64-linux-gnu/`

3. Ensuring that the ld.so.preload file points to it.

4. Creating another reverse shell like you did in Part 1

5. Verify that the reverse-shell TCP connection is now hidden from netstat.

INCLUDE 2 SCREENSHOTS. 1 Containing the reverse-shell and 1 containing the netcat output with the TCP connection now hidden.

Part 3: Short Answer and True/False

a) Explain the difference between user level and kernel level rootkits. What are their strengths and weaknesses respectively?

TODO:

b) Do you think that rootkits are a greater threat on Windows or Linux? There's no right or wrong answer, as long as you justify it. Give at least 2-3 points.

TODO:

c) True or False: Wiping all drives and reinstalling the OS is enough to remove a rootkit.

TODO:

d) What stage of the compilation process do LD_PRELOAD rootkits exploit?

TODO:

e) True or False: LD_PRELOAD rootkits take advantage of dynamic linking as opposed to static linking.

TODO:

f) Is a rootkit just a kit (software) that runs with root privileges? Explain why or why not?

TODO:

Some Debugging Tips

- Programs like ls. cat, vim, etc. may use the write() or other libc library calls. So, hooking them may cause these programs to behave abnormally. You may find clearing out the ld.so.preload file to be useful which will allow these programs to reference the original libc library calls and not the hooked versions. To do this you can:
    - Sudo vim the ld.so.preload file and delete the path to your shared/dynamic library
    - If vim doesn't work since it may depend on a hooked libc call, you can try doing echo 1 > <path to ld.so.preload> which will remove the reference to your shared/dynamic library.
    - If there is still weird behaviour, you may need to kill any vim processes (try going to applications > task manager > sort by task > kill each instance of vim)
    - The sshd daemon might have crashed as well, try running `sudo systemctl restart ssh`, and then try checking the status
    - If all else fails, you may need to revert to a **fresh VM** or **snapshot**