Using Docker to Create Multi-Container Environments for Research and Sharing Lateral Movement

Shaun McCullough
Using Docker to Create Multi-Container Environments for Research and Sharing Lateral Movement Scenarios

GIAC (GXPN) Gold Certification

Author: Shaun McCullough, cybergoof@gmail.com
Advisor: Chris Crowley
Accepted Date: June 22, 2017

Abstract

Docker, a program for running applications in containers, can be used to create multi-container infrastructures that mimic a more sophisticated network for research in penetration techniques. This paper will demonstrate how Docker can be used by information security researchers to build and share complex environments for recreation by anyone. The scenarios in this paper recreate previous research done in SSH tunneling, pivoting, and other lateral movement operations. By using Docker to build sharable and reusable test infrastructure, information security researchers can help readers recreate the research in their own environments, enhancing learning with a more immersive and hands on research project.
1 Introduction

Information Security professionals research and publish new ideas, concepts, and approaches to exploitation, defense, public awareness, and forensics. The SANS website promotes eleven different blogs that regularly publishes research on all aspects of information security (SANS Blogs). Researchers focusing on penetration testing and exploitation techniques can spend countless hours building scenarios in their labs to demonstrate and understand an innovative concept or hack (McJunkin, 2014). However, it can be difficult for readers to reproduce the research without having explicit instructions on how to replicate the environment from scratch. For the casual reader, who does not have a ready-to-go testing infrastructure, it can be an even steeper mountain to climb. Recent advances in Docker products provides new ways of reproducing network environments with just a few simple commands.

2 Docker

Docker is a software container platform that attempts to eliminate a common problem; applications tend to run differently, and unexpectedly, on development environments than in production environments (What is Docker). The idea of containers can be traced to the chroot system call introduced to Unix V7 in 1979 (History of Containers, p. Chroot). Docker was first developed by Solomon Hykes, in March of 2013, as an internal project within dotCloud, a platform-as-service company. Since then, this open source software has grown a significant following with contributors from Microsoft, Dropbox, RedHat, IBM, Huawei and Google (Docker Contributors Graph).

2.1 Docker Basics

The Docker platform builds upon Linux containers with self-contained execution environments in their isolated CPU, memory, block I/O and network resources. They share the kernel of the host operating system, which gives the feeling of a separate virtual machine to the applications running in the container (Wang, 2016). Although containers might appear to be virtual machines, they are only small, streamlined execution environments.
environments. It is possible to run multiple containers on the same system without the overhead typical in virtual machine architectures. In Figure 1 below, the stack on the left uses a traditional virtual machine model to isolate the applications. The stack on the right depicts a Docker container-based system.

![Figure 1: Docker Versus Virtual Machine Architecture](image)

Although the applications operate as if they are functioning in a full machine stack, the container engine on the right is providing all the functionality of the OS that the applications need, without the full guest operating system. Not only does the container-based system save resources on the host machine, but it significantly reduces the complexity of running those apps across other container-based infrastructures.

### 2.2 Docker Image, Containers, and Storage Drivers

A Docker image is made up of separate read-only images that represent filesystem differences. Each layer is stacked on top of each other to form the base for that particular image’s root filesystem (Docker Docs). A container takes those read-only image layers and adds a writeable top. Effectively, a container is a running instance of a Docker image. A single image is used to spin up multiple Docker containers, each built with a different writeable layer, but with most of the information the same across all containers.

Shaun McCullough, cybegoof@gmail.com
2.3 The Docker Registry

Docker can launch multiple applications on the same hardware with ease, but it is the use of registries which allows the distribution of images (Mouat, 2016). Out of the box, Docker will pull and build images straight from Docker’s supported Docker Hub at https://hub.docker.com/, which can house official repositories of images as well as personally created ones. The registry is central to sharing Docker built infrastructures and providing built-in commands to pull and push images to the Docker public registry.

A repository, not to be confused with a registry, is a collection of related Docker images that typically provides multiple versions of the same application or service. For instance, Figure 2 shows the Offensive Security team’s official Kali Linux images in the “kali-linux-docker” repository at the Docker Hub registry at hub.docker.com/r/kalilinux/kali-linux-docker/.

![Figure 2: Kali Linux Docker Hub Page](image-url)

Shaun McCullough, cybergoof@gmail.com
Figure 3 shows the Docker Hub dashboard which gives one the ability to tag each version of the image, show build details, and link to source code repositories such as GitHub.

![Figure 3: Kali Linux Tag Page on Docker Hub](image)

This public Docker registry, with repositories of publically available Docker images, creates an ecosystem of images shared across different target environments. Any infrastructure that fully implements Docker can support Docker containers. A Docker container can truly be portable since the networking, storage and OS details are abstracted away from the application (Wang, Containers 101: Linux Containers and Docker Explained). Information Security researchers publishing research can build a fully functioning Docker image and store it in their Docker Hub repositories. Anybody interested in replicating the research can access the exact environment setup the researchers have built.

The operating system is abstracted away from containers, so it is easy to move containers across any system that supports the container runtime environment. In addition to being able to run Docker on Windows, OSX and other Linux environments, many cloud hosting services such as Amazon’s AWS (Amazon), Microsoft’s Azure (Iain Foulds), Google Container Engine (Google), and DigitalOcean (DigitalOcean) support running Docker containers. A 2`container running in one place should work in every container supported hosting service. This portability allows Docker containers to be widely distributable and sharable. Coupled with the Docker Hub registry for self-

Shaun McCullough, cybergoof@gmail.com
discovery, it is simple to grab Docker images and start a running infrastructure with a few short commands.

3 Running Docker

It is relatively easy to being running a prepackaged Docker container from Docker Hub after installing the Docker Engine (Docker Engine, p. Install Docker). Recently, Docker split its offerings into a Community Edition and an Enterprise Edition (Bhartiya, 2017). As the name implies, the Community Edition is free for anyone to use and supports most major operating systems. The Enterprise Edition provides additional capabilities to help manage critical containers running in production environments (Docker Enterprise). This research uses the Community Edition because it is free to anyone, can be run on most versions of Mac, Windows and Linux, and does not require a cloud service to support.

3.1 First Docker Container

Once Docker has been installed on a system, the command line can be used to interact with all Docker containers. Whalesay, a “Hello World” type application from store.docker.com/community/images/docker/whalesay, will demonstrate basic Docker commands (Docker Store). Docker pulls from the Docker registry to grab the right image as seen in Figure 4.

![Figure 4: Docker Pull Command for Whalesay](image-url)

Shaun McCullough, cybergoof@gmail.com
Each layer of the Docker image has a twelve-digit hexadecimal identification number. These identification numbers are used to represent image layers, the image as a whole, and containers. Users can assign text names to images and containers, but these identification numbers are assigned and unique as seen in Figure 5.

Docker pull will go out to the Docker registry, look for the docker/whalesay repository, and pull down the latest image. The “docker” part of docker/whalesay is to differentiate namespaces. From the “docker pull” command, Docker Engine pulled down the latest whalesay image and loaded it into the local Docker registry running on the local system. As seen in Figure 5, the command “docker images” will display the image information.

![Figure 5: Looking at Loaded Docker Images](image1)

Figure 6 shows how to start the whalesay container, running the command `cowsay boo-boo`.

![Figure 6: Running Whalesay from Docker](image2)

Shaun McCullough, cybergoof@gmail.com
The whalesay image started a running container, ran the *cowsay boo-boo* application, then exited the application. Once the application exited, the whalesay Docker container also stopped.

### 3.2 Customizing the Docker Image

In some cases, a Docker image can be used from Docker Hub without any changes. More customized Docker images will require new layers to be added to Docker. Since a Docker image is a set of read-only layers, it is possible to add custom read-only layers on top with a Dockerfile. A Dockerfile is a text document that contains the commands Docker needs to provision or build a new Docker image (Dockerfile Reference). The first command in a Dockerfile is *FROM*, which identifies the base Docker image. A base Ubuntu image could be used to run an application (Docker Hub Ubuntu), or start with a standard service built to run an application such as the Nginx web server (Docker Hub Nginx), or even a complete Docker image of the Kali distribution (Docker Hub Kali).

The Dockerfile will run shell commands, set up networking, create environment variables, create shared volumes between the container and the host computer supporting, 18 different commands in total. The Dockerfile in Figure 7, used to build Whalesay, is documented on the Whalesay Docker Hub project.

![Dockerfile](image)

**Figure 7: Whalesay Dockerfile**

Shaun McCullough, cybergoof@gmail.com
FROM shows it is starting from an Ubuntu image, version 14.04. The first RUN command updates the installed packages and installs the cowsay package from the Ubuntu image. ENV adds /usr/games to the PATH environment variable. COPY moves a docker.cow file into the container’s directory. The final RUN command creates a symbolic link to overwrite the default.cow with the docker.cow. Lastly, CMD is telling the Docker image what to run when the container starts. These sets of commands are pretty standard for most Dockerfiles.

### 3.3 Multi Containers

Building a single Docker container to run a single application is simple. The maintainers of the REMnux Linux toolkit provide a set of Docker images that each have a different malware analysis toolset ready to be started in a single command (Docker Hub REMnux). It is easy to fire up a PEScanner (docker pull remnux/pescanner) or a Volatility (docker pull remnux/volatility) container that can later be torn down. More complicated setups, however, require multiple containers that interact together. A NodeJS web application might rely on MongoDB for its database. It would be better to have the NodeJS stack in one container, highly customized for the desired application, while the MongoDB is in a separate container built from the official image in Docker Hub. The Docker Compose application can orchestrate the instantiation of multiple Docker containers as a group of related containers (Docker Compose). Docker Compose uses a YAML file to build and run each Docker container, reproducing the run time instructions from the command line when running Docker (Compose YML). The following section will go into the details of Docker Compose to demonstrate how it can be used to build, test, and distribute environments for anyone to replicate penetration testing research.

### 4 Docker Compose Labs

Leveraging SANS published papers, Section 4 will build three scenarios that replicate at least part of their research using Docker. These papers focus on multi-system attacks, on Linux-based machines, and use publically available and free software. This

Shaun McCullough, cybergoof@gmail.com
section will also go into detail of Docker Compose and how it can be used to orchestrate multiple containers.

4.1 SSH and the “Konami Code” Blog Post

SANS Instructor, and Senior Technology Analyst, Jeff McJunkin, published a blog post on SSH Control Sequences useful for established SSH connections (McJunkin, Konami Code, 2015). Mr. McJunkin identified ten different escape sequences and control commands that can be used in an open SSH connection.

To replicate this research, a Docker Compose file will be used to build two Linux-based Docker containers. One of the containers will run an SSH Server and the other will run an SSH client. To simplify the setup, an already created Docker image will be used with all the needed software installed. Docker Compose will do all the magic.

Docker Hub hosts has an image called `phusion/baseimage` available at `hub.docker.com/r/phusion/baseimage` that is a basic Ubuntu container configured to work in Docker with an SSH client. Docker Hub has a Docker image built with an SSH server at `hub.docker.com/r/keto/ssh` that can be configured, through environment variables, to start an SSH server with a given password. It only requires a docker-compose.yml file to start these two containers, and link them. The basics of the YAML Language are at `http://yaml.org/`.

Shaun McCullough, cybergoof@gmail.com
The YAML file is used to define services, networks, and volumes. Each service is a separate Docker container. In Figure 8 above, the attacker container is from the image phusion/baseimage. Using build tells Compose to pull the image from the remote repository on Docker Hub. The container_name value will name the running container. The links will tell Docker to link this container to another container allowing attacker to connect to the victim container. The command entrypoint tells which application to run when the container starts. Docker containers are designed to stop once it’s main application has stopped running. As a server, the Docker container will run as long as the server software is running. A Docker container running as a client, however, does not really have a continuous running application to keep the container live. The my_init script will keep the container running.

Shaun McCullough, cybergoof@gmail.com
The victim container uses the keto/ssh image and names it victim. One feature of Compose is the ability to set environment variables at run time. The SSH_PASSWORD variable will be the SSH password for the user called user on the victim container.

All files necessary to run the labs in this paper are available at https://github.com/cybergoof/gxpn. Every Docker compose file, scripts, and support files can be found to reproduce these labs.

For the first lab, run Docker Compose with the YAML file, lab1.yml, and bring up the containers with the up command. Figure 9 shows the command and Docker pulling down the images needed for attacker and victim.

Figure 9: Starting Lab 1 with Docker Compose

The –d at the end of the command tells Docker to make the containers run in the background. Running docker ps from Figure 10 shows that the containers are running.

Shaun McCullough, cybergoof@gmail.com
To test out the SSH control codes, it is necessary to run bash from inside *attacker* container, and SSH into the *victim* container. Docker’s preferred method of connecting to a running container is with the `exec` command as shown in Figure 11. The options for the `exec` are presented in Figure 12.

**Figure 11: Entering a Running Container**

Use of `-it` tells Docker to execute `/bin/bash` on the container in interactive mode with a pseudo-TTY. As shown in Figure 13, SSH was successful from the *attacker* container to the *victim* container with a password of `mypassword`. When setting up the

Shaun McCullough, cybergoof@gmail.com
container, Docker already assigned the hostnames through the `link` option in the `lab1.yml` file.

![Figure 13: SSH Command from Attacker to Victim](image)

Using the YAML configuration file, Docker brought up an infrastructure that is able to reproduce the research described in Mr. McJunkin’s blog post. Figure 14 displays all the SSH control sequences.

![Figure 14: List of SSH Control Commands](image)

The command sequence to setup a SOCKS proxy can be replicated with the commands in Figure 15.

Shaun McCullough, cybergoof@gmail.com
With just a single `lab1.yml` file and a single `docker-compose` command, infrastructure will be built to replicate Jeff McJunkin’s blog post. It is especially convenient that the Docker images necessary to repeat this blog post are already available on Docker Hub without any modifications. Publically available services, such as Docker Hub and GitHub, can provide all the resources needed to share and rebuild this research by any student. In the next lab, more containers will be used to replicate pivoting techniques.

### 4.2 SSH to Pivot through a Server

The concept of pivoting in a penetration test is to use a compromised system to attack another machine, using that first victim machine as the launch point for attacks against other computers. A pivot attack is particularly devastating if a network has a DMZ and a juicier inside network along with a gateway system that sits between both networks (superkojiman, 2012). If one has SSH access to the pivot computer, then they can use that SSH connection to tunnel through the gateway and into the real target network.

Figure 16 shows a diagram of the network setup Docker Compose will build for this lab. The container that will launch the attacks will be on the DMZ network, 172.16.200.0/24. The final targets will be on 172.16.201.0/24. The gateway machine will sit on both networks. Docker Compose will set all this up for us with the networks described in the YAML file (Docker Networking).
The first line of a Docker Compose YAML file identifies the version of the Compose file in use. As of this writing, Version 3 was the most recent; however, it seems that some IPAM configurations available in Version 2 are not yet available in Version 3 (IPAM). This scenario requires these IPAM configurations to setup the two subnetworks. It appears that the changes in Version 3 of the Docker Compose product will help align Docker Compose with the Docker Swarm product, which is Docker’s cluster management and orchestration feature (zaquestion, 2017).

It is occasionally necessary to clear out all old Docker artifacts such as images, containers, volumes and networks. A user created a guide for cleaning up the old Docker “stuff” that is lying around at https://gist.github.com/bastman/5b57ddb3c11942094f8d0a97d461b430. If there is an error in creating images, running containers, or creating networks, then run the commands from this site.

To build the subnets with specified IP addresses to segment the data, Docker allows custom networks in bridge mode. All networks created with the bridge driver use the Linux Bridge, or a virtual switch, which is why Linux-based host machines are used for the research in this paper (Marks, 2016). In the lab2.yml, a section called networks will build custom subnetworks with the proper gateway and subnet information. These

Shaun McCullough, cybergoof@gmail.com
subnets will be used to recreate part of the work done in Gordon Fraser’s 2015 GWAPT Gold Certification Paper titled *Tunneling, Pivoting, and Web Application Penetration Testing* (Fraser, 2015). Section 2.5 of Fraser’s paper describes how to create a SOCKS tunnel through a pivot server to run applications with ProxyChains.

From the *lab2.yml*, shown in Figure 17, two subnetworks are created: *attack-net* and *target-net*. The *attack-net* will live on 172.16.200.0/24 and *target-net* will live at 172.16.201.0/24. With the networks established, each host will be assigned to one or more networks.

```
  networks:
    DMX-net:
      driver: bridge
      driver_opts:
        com.docker.network.enable_ipv6: "false"
      ipam:
        driver: default
        config:
          - subnet: 172.16.200.0/24
            gateway: 172.16.200.3
    target-net:
      driver: bridge
      driver_opts:
        com.docker.network.enable_ipv6: "false"
      ipam:
        driver: default
        config:
          - subnet: 172.16.201.0/24
            gateway: 172.16.201.3
```

*Figure 17: Network Commands for Lab 2*

Figure 18 shows the *attacker* and *pivot* Docker containers that will be created for this lab.

Shaun McCullough, cybergoof@gmail.com
Figure 18: Attacker and Pivot Compose Configurations

The *pivot* host is part of both networks. Assigning *hostname* will set the containers hostname, and *extra_hosts* will assign those hosts to the `/etc/host` file. Figure 19 shows the build for the `target1` and `target2` containers. The `target1` machine will have two ports open, port 80 for HTTP and port 3306 for MySQL.

Shaun McCullough, cybergoo@gmail.com
The attacker container is being built locally from a Dockerfile, as shown in Figure 20. Rather than pulling the finished Docker image from Docker Hub as before, a base Docker image will be customized.

Shaun McCullough, cybergoo@gmail.com
Figure 20: DockerFile for Attacker Container

One of the useful features of Docker is ability to build off an already created image from Docker Hub. For this lab, the `phusion/baseimage` image will be the base image, but the DockerFile will tell Docker to install three additional programs: nmap (line 16), proxychains (line 17) and ping (line 21). Running the commands in Figure 21 will start the Lab 2 network, which includes all four containers.

Figure 21: Docker Compose Commands for Lab 2

Once the pivot, target1, target2, and attacker containers are running, the commands from Figure 11 can be used to connect to the attacker container. Once

Shaun McCullough, cybergoof@gmail.com
connected to the *attacker* container, try to ping the pivot machine and the *target1* machine. As depicted in Figure 22, the *pivot* container responds, but *target1* does not.

![Ping results](image)

**Figure 22: From Attacker, Ping the Pivot and Target1 container**

Using SSH, a SOCKS proxy will be setup on the *pivot* container. A SOCKS proxy is a general purpose proxy server that establishes a connection to another server and routes all traffic between the client and the server (SOCKS Proxy, 2012). Once the proxy is established, a SOCKS proxy aware application can use that SOCKS proxy to communicate through the pivot server to the other subnet. If a program is not SOCKS proxy aware, then the application *ProxyChains* is used as the client side of the SOCKS proxy.

From the *attacker* machine, look at the `/etc/proxychains.conf` file to find out which port ProxyChains is to use. The last line of the configuration file shows a default entry, using port 9050 over 127.0.0.1. Following the SSH command in Section 2.5 of the GWAPT paper, the SSH Socks creation commands are detailed in Figure 23.
The syntax of the commands is: `ssh -D local-address:port -f -N username@gateway-address`. Section 2.5.1 in the GWAPT paper demonstrated using NMAP to scan a host. First, Figure 24 shows that the `target1` container is not visible to NMAP from the `attacker` container.

Adding `proxychains` to the beginning of the command, the NMAP commands are run through the Proxy running on the `pivot` container, as seen in Figure 25 and Figure 26.
Figure 26: ProxyChains and SOCKS Proxy Allows NMAP to Scan Target1 from Attack

ProxyChains will proxy all network traffic through the SSH tunnel on the pivot container, successfully scanning the target1. Because target1 is running a vulnerable WordPress site, HTTP and MySQL Servers ports are open.

Using just a few commands, four containers mimicking four machines are running on two subnets. Using SSH and ProxyChains, NMAP can run through the pivot server and into the target network. This complex scenario was easy to setup with just a few commands. The final lab will replicate a more complex scenario, using Metasploit framework to do the pivoting.

4.3 Metasploit to Pivot

Section 2.6 of the GWAPT paper describes a more complex method of pivoting, using the Metasploit penetration testing framework. Meterpreter, which is part of Metasploit, is an extensible payload that uses in-memory injection stagers to support a vast array of penetration testing capabilities (Meterpreter). One capability of Meterpreter is to act as a SOCKS Proxy and route any Metasploit command through Meterpreter (Pivoting). Metasploit is the ultimate toolkit for pentesters and is the subject of many research posts. Docker Compose, with a few configuration files, can be used to replicate an environment to implement a SOCKS Proxy using Metasploit.

First, the attacker container must have Metasploit installed. Remnux already provides a Docker image with the full Metasploit framework (Zeltser, 2016). The Docker container for this lab can be run with no customization as seen in Figure 27.

Shaun McCullough, cybergoof@gmail.com
Figure 27: Attacker and Pivot Configuration for Lab 3

```
services:
  attacker:
    image: remnux/metasploit
    container_name: attacker
    ports:
      - "443:443"
    hostname: attacker
    volumes:
      - ~/msf4:/root/.msf4
      - /tmp/msf:/tmp/data
      - .attacker.lab3:/tmp
    command: "tail -F -n0 /etc/hosts"

networks:
  DMZ-net:
    ipv4_address: 172.16.200.10

extra_hosts:
  - "pivot:172.16.200.11"
  - "target1:172.16.201.12"
  - "target2:172.16.201.13"

pivot:
  build: ./pivot
  container_name: pivot
  hostname: pivot
  extra_hosts:
    - "attacker:172.16.200.10"
    - "target1:172.16.201.12"
    - "target2:172.16.201.13"
  networks:
    DMZ-net:
      ipv4_address: 172.16.200.11

  target-net:
    ipv4_address: 172.16.201.11
```

Shaun McCullough, cybergoo@gmail.com
The command to start the Metasploit server is a little odd. Docker containers are designed to run something and quit when it is done. The `tail` command is used to keep the Docker container running. For Meterpreter to run on the Pivot box, the Docker image needs a more standardized installation of Ubuntu. So, a custom Docker container will be built using the Dockerfile in Figure 28.

```bash
FROM ubuntu:16.04
# Labels just add Metadata to the image.
# Its like code comments for Docker images
LABEL maintainer="cybergoof"
LABEL description="This Dockerfile was build for a GXPN Gold Paper. It will be the

RUN apt-get update & apt-get install -y \
    git \
    libxml2-dev \
    python \
    build-essential \
    make \
    gcc \
    python-dev \
    locales \
    python-pip \
    net-tools \
    iputils-ping

RUN apt-get install -y \
    nmap

RUN apt-get update & apt-get install -y openssh-server
RUN mkdir /var/run/sshd
RUN echo 'root:mypassword' | chpasswd
RUN sed -i '/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd

# SSH login fix. Otherwise user is kicked off after login
RUN sed -i /@session\$required\@session\$session\$optional\@session\$optional\@session\@g

ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile

EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

Figure 28: Pivot Box Dockerfile to Install an SSH Server on Ubuntu

The file, `lab3.yml`, will be used by Docker Compose to start up all the four containers and the proper network setup, as seen in Figure 29.
Once the containers have started and the commands from Figure 30 have been run, the attacker server must be configured to properly run Metasploit. Running the script /tmp/init.sh will install Ruby, start Postgress, and update the Metasploit software. Once configured, starting the Metasploit console is with a single command /opt/msf/msfconsole from Figure 31:

Now that the Metasploit Framework is running on the attacker container, an attempt to scan ports 80 and 3306 of the target1 machine fails, as seen in Figure 32.

Shaun McCullough, cybergoof@gmail.com
As expected, Metasploit could not see ports open on 80 or 3306. The `ssh_login` Metasploit plugin will SSH into the pivot machine and return a shell as seen in Figure 33.

Once shell access on the pivot container is granted, the fully functioning Meterpreter should be installed onto the pivot container. User tom on daimonchild.com created a blog post on how to upload Meterpreter with just ssh_login shell access (tom, Shaun McCullough, cybergoo@gmail.com)
2015). Using the plugin `shell_to_meterpreter` will load Meterpreter on the pivot machine as seen in Figure 34:

```
msf auxiliary(ssh_login) > use post/multi/manage/shell_to_meterpreter
msf post(shell_to_meterpreter) > set session 1
session => 1
msf post(shell_to_meterpreter) > exploit

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 172.16.200.10:4433
[*] Starting the payload handler...
[*] Sending stage (797784 bytes) to 172.16.200.11
[*] Meterpreter session 2 opened (172.16.200.10:4433 -> 172.16.200.11:52712
[*] Command stager progress: 100.00% (668/668 bytes)
[*] Post module execution completed
msf post(shell_to_meterpreter) > sessions -i

Active sessions

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>shell/linux</td>
<td>SSH root:mypassword (172.16.200.11:22)</td>
</tr>
<tr>
<td>2</td>
<td>meterpreter</td>
<td>x86/linux uid=0, gid=0, euid=0, egid=0 @ 172.16.200.11</td>
</tr>
</tbody>
</table>

msf post(shell_to_meterpreter) >
```

Figure 34: Using Metasploit Command to Upgrade Shell Session to a Meterpreter session

From Figure 34, there are now two sessions. The original shell in session 1, and now a Meterpreter session at session 2. The GWAPT paper in Section 2.6 describes how to use the command to add a route, or proxy, all commands through the Meterpreter shell running on pivot and is replicated in Figure 35.
Figure 35: Setting a Route Through Pivot Box to Target1 Subnet

As shown in Figure 35, after using the `route add` command, a scan against 172.16.201.12 is now successful, seeing open ports 80 and 3306. The scan went from the attacker container running the Metasploit server, through Meterpreter running a proxy server on the pivot container, and against the target1 container as if they were all on the same subnetwork. Any of Metasploit’s exploitation and survey commands can now be launched against the 172.16.201.0 subnetwork. This is a simple way to setup an infrastructure to replicate a complicated Metasploit scenario. A researcher could also build and publish a custom build Docker image with specific tools, vulnerabilities, or scripts to support even more immersive, complex, or sophisticated penetration testing research scenario. The Docker Hub registry makes sharing those custom images simple with just a few commands. As Docker becomes more feature rich to support developers and operators, it also opens up opportunities for researchers to share and distribute research scenarios, making them more effective for their audience.

Shaun McCullough, cybergoof@gmail.com
5 Conclusion

As demonstrated in the three labs of Section 4, Docker can replicate rich networked environments to test and share penetration testing research. With a few commands, all the files and images needed for a multi-subnet environment can be shared and rebuilt by students to run the research. Researchers, using these tools, can create a more immersive and educational experience for their readers. Docker Hub and GitHub can allow researchers to publish all source code, configuration files, and instructions for replicating these environments. This ease of sharing can significantly increase researcher’s outreach to their audience, allowing them to quickly set up and run experiments in their environments with limited overhead. Reducing this friction will make research more applicable, easier to build upon, and a more effective learning tool.

Shaun McCullough, cybergoo@gmail.com
6 Bibliography

Amazon. (n.d.). *Docker Basics*. Retrieved from Amazon Web Services:

http://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html


Compose YML. (n.d.). *Compose file version 3*. Retrieved from Docker Docs:

https://docs.docker.com/compose/compose-file/


https://www.digitalocean.com/community/tutorials/how-to-use-the-digitalocean-docker-application

Docker Compose. (n.d.). *Docker Compose*. Retrieved from Docker Docs:

https://docs.docker.com/compose/

Docker Docs. (n.d.). *Understand images, containers, and storage drivers*. Retrieved from Docker Docs:

https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/


https://docs.docker.com/engine/installation/

*Docker Enterprise Edition*. (n.d.). Retrieved from Docker Enterprise:

https://www.docker.com/enterprise-edition

Shaun McCullough, cybergoo@gmail.com

https://hub.docker.com/r/kalilinux/kali-linux-docker/

Docker Hub REMnux. (n.d.). *remnux*. Retrieved from Docker Hub:

https://hub.docker.com/r/remnux/


https://docs.docker.com/compose/networking/#specifying-custom-networks


https://store.docker.com/community/images/docker/whalesay

Docker. (n.d.). *What is Docker*. Retrieved from Docker Main Site:

https://www.docker.com/what-docker

*Dockerfile Reference*. (n.d.). Retrieved from Docker Docs:

https://docs.docker.com/engine/reference/builder/


GitHub. (n.d.). *Docker Contributors Graph*. Retrieved from Docker on Github:

https://github.com/docker/docker/graphs/contributors


https://cloud.google.com/container-engine/


https://github.com/docker/labs/blob/master/networking/A2-bridge-networking.md

Shaun McCullough, cybergoof@gmail.com
Docker To Build Multi-Container Labs


Shaun McCullough, cybergoof@gmail.com
Shaun McCullough, cybergoof@gmail.com