With the rising integration and adoption of Docker containers, there is a growing need to ensure their security.

The purpose of this book is to provide techniques and enhance your skills to secure Docker containers easily and efficiently. The book starts by sharing the techniques to configure Docker components securely and explore the different security measures/methods one can use to secure the kernel.

Furthermore, we will cover the best practices of reporting Docker security findings and will show you how you can safely report any security findings you come across. Toward the end, we list the internal and third-party tools that can help you immunize your Docker environment.

By the end of this book, you will have a complete understanding of Docker security and will be able to protect your container-based applications.

Who this book is written for
This book is for developers who wish to use Docker as their testing platform as well as security professionals who are interested in securing Docker containers. You must be familiar with the basics of Docker.

What you will learn from this book
- Find out how to secure your Docker hosts and nodes
- Secure your Docker components
- Explore different security measures/methods for Linux kernels
- Install and run the Docker Bench Security application
- Monitor and report security issues
- Familiarize yourself with third-party tools such as Traffic Authorization, Summon, sVirt, and SELinux to secure your Docker environment
- Stay up-to-date on Docker-related security issues that arise with its version updates

Learn how to secure your Docker environment and keep your environments secure irrespective of the threats out there.
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Securing Docker Hosts'
- A synopsis of the book’s content
- More information on *Securing Docker*
Scott Gallagher has been fascinated with technology since he was in elementary school, when he used to play Oregon Trail. His love continued through middle school, working on more Apple IIe computers. In high school, he learned how build computers and program in BASIC! His college years were all about server technologies such as Novell, Microsoft, and Red Hat. After college, he continued to work on Novell, all while keeping an interest in all the technologies. He then moved into managing Microsoft environments and eventually into what he is the most passionate about, Linux environments, and now his focus is on Docker and cloud environments.
 Preface

Docker is the hottest buzzword in technology these days! This book helps you to ensure that you are securing all the pieces in the Docker ecosystems of tools. Keeping your data and systems safe is of utmost importance these days, and with Docker, it's the same exception. Learn how Docker is inherently secure and how to secure the pieces around it even more and be on the lookout for potential vulnerabilities as they take place.

What this book covers

Chapter 1, Securing Docker Hosts, starts off the book by discussing how to secure the first part of getting your Docker environment up and running, and that is by focusing on your Docker hosts. The Docker hosts are the platform that your containers will run on. Without securing these first, it's like leaving the front door to your house wide open.

Chapter 2, Securing Docker Components, focuses on securing the components of Docker, such as the registry you can use, the containers that run on your hosts, and how to sign your images.

Chapter 3, Securing and Hardening Linux Kernels, explains hardening guides that are out there as well as different security measures/methods you can use to help secure the kernel that is being used to run your containers as it's important to secure it.

Chapter 4, Docker Bench for Security, informs how well you have set up your Docker environment with the Docker Bench Security application, get recommendations for where you should focus your efforts to fix right away, and what you don't really have to fix right now, but should keep yourself aware of.
Chapter 5, Monitoring and Reporting Docker Security Incidents, covers how to stay on top of the items that Docker has released regarding the security findings to help keep you aware of your environments. Also, we will take a look at how to safely report any security findings you come across to ensure that Docker has a chance to alleviate the concern before it becomes public and widespread.

Chapter 6, Using Docker's Built-in Security Features, introduces the use of Docker tools to help secure your environment. We will go over all of them to give you a baseline of what you can use that is provided by Docker itself. You can learn what command-line and GUI tools you can use for your security needs.

Chapter 7, Securing Docker with Third-party Tools, covers the third-party tools that are out there to help you keep your Docker environment secure. You will learn about command line, but we'll focus on third-party tools. We will take a look at traffic authorization, summon, and sVirt with SELinux.

Chapter 8, Keeping up on Security, explains the means that you can use to keep up to date with Docker-related security issues that are out there for the version of the Docker tools you might be running now, how to stay ahead of any security issues, and keep your environments secure even with threats out there.
Securing Docker Hosts

Welcome to the *Securing Docker* book! We are glad you decided to pick up the book and we want to make sure that the resources you are using are being secured in proper ways to ensure system integrity and data loss prevention. It is also important to understand why you should care about the security. If data loss prevention doesn't scare you already, thinking about the worst possible scenario—a full system compromise and the possibility of your secret designs being leaked or stolen by others—might help to reinforce security. Throughout this book, we will be covering a lot of topics to help get your environment set up securely so that you can begin to start deploying containers with peace of mind knowing that you took the right steps in the beginning to fortify your environment. In this chapter, we will be taking a look at securing Docker hosts and will be covering the following topics:

- Docker host overview
- Discussing Docker host
- Virtualization and isolation
- Attack surface of Docker daemon
- Securing Docker hosts
- Docker Machine
- SELinux and AppArmor
- Auto-patching hosts

**Docker host overview**

Before we get in depth and dive in, let's first take a step back and review exactly what the Docker host is. In this section, we will look at the Docker host itself to get an understanding of what we are referring to when we are talking about the Docker host. We will also be looking at the virtualization and isolation techniques that Docker uses to ensure security.
Securing Docker Hosts

Discussing Docker host
When we think of a Docker host, what comes to our mind? If you put it in terms of virtual machines that almost all of us are familiar with, let's take a look at how a typical VM host differs from a Docker host. A VM host is what the virtual machines actually run on top of. Typically, this is something like VMware ESXi if you are using VMware or Windows Server if you are using Hyper-V. Let's take a look at how they are as compared so that you can get a visual representation of the two, as shown in the following diagram:

The preceding image depicts the similarities between a VM host and Docker host. As stated previously, the host of any service is simply the system that the underlying virtual machines or containers in Docker run on top of. Therefore, a host is the operating system or service that contains and operates the underlying systems that you install and set up a service on, such as web servers, databases, and more.

Virtualization and isolation
To understand how Docker hosts can be secured, we must first understand how the Docker host is set up and what items are contained in the Docker host. Again, like VM hosts, they contain the operating system that the underlying service operates on. With VMs, you are creating a whole new operating system on top of this VM host operating system. However, on Docker, you are not doing that and are sharing the Linux Kernel that the Docker host is using. Let's take a look at the following diagram to help us represent this:
As we can see from the preceding image, there is a distinct difference between how items are set up on a VM host and on a Docker host. On a VM host, each virtual machine has all of its own items inclusive to itself. Each containerized application brings its own set of libraries, whether it is Windows or Linux. Now, on the Docker host, we don't see that. We see that they share the Linux Kernel version that is being used on the Docker host. That being said, there are some security aspects that need to be addressed on the Docker host side of things. Now, on the VM host side, if someone does compromise a virtual machine, the operating system is isolated to just that one virtual machine. Back on the Docker host side of things, if the kernel is compromised on the Docker host, then all the containers running on that host are now at high risk as well.

So, now you should see how important it is that we focus on security when it comes to Docker hosts. Docker hosts do use some isolation techniques that will help protect against kernel or container compromises in a few ways. Two of these ways are by implementing namespaces and cgroups. Before we can discuss how they help, let's first give a definition for each of them.

Kernel namespaces, as they are commonly known as, provide a form of isolation for the containers that will be running on your hosts. What does this mean? This means that each container that you run on top of your Docker hosts will be given its own network stack so that it doesn't get privileged access to another container's socket or interfaces. However, by default, all Docker containers are sitting on the bridged interface so that they can communicate with each other easily. Think of the bridged interface as a network switch that all the containers are connected to.
Securing Docker Hosts

Namespaces also provide isolation for processes and mount isolation. Processes running in one container can't affect or even see processes running in another Docker container. Isolation for mount points is also on a container by container basis. This means that mount points on one container can't see or interact with mount points on another container.

On the other hand, control groups are what control and limit resources for containers that will be running on top of your Docker hosts. What does this boil down to, meaning how will it benefit you? It means that cgroups, as they will be called going forward, help each container get its fair share of memory disk I/O, CPU, and much more. So, a container cannot bring down an entire host by exhausting all the resources available on it. This will help to ensure that even if an application is misbehaving that the other containers won't be affected by this application and your other applications can be assured uptime.

**Attack surface of Docker daemon**

While Docker does ease some of the complicated work in the virtualization world, it is easy to forget to think about the security implications of running containers on your Docker hosts. The largest concern you need to be aware of is that Docker requires root privileges to operate. For this reason, you need to be aware of who has access to your Docker hosts and the Docker daemon as they will have full administrative access to all your Docker containers and images on your Docker host. They can start new containers, stop existing ones, remove images, pull new images, and even reconfigure running containers as well by injecting commands into them. They can also extract sensitive information like passwords and certificates from the containers. For this reason, make sure to also separate important containers if you do need to keep separate controls on who has access to your Docker daemon. This is for containers where people have a need for access to the Docker host where the containers are running. If a user needs API access then that is different and separation might not be necessary. For example, keep containers that are sensitive on one Docker host, while keeping normal operation containers running on another Docker host and grant permissions for other staff access to the Docker daemon on the unprivileged host. If possible, it is also recommended to drop the setuid and setgid capabilities from containers that will be running on your hosts. If you are going to run Docker, it's recommended to only use Docker on this server and not other applications. Docker also starts containers with a very restricted set of capabilities, which works in your favor to address security concerns.
To drop the setuid or setgid capabilities when you start a Docker container, you will need to do something similar to the following:

```bash
$ docker run -d --cap-drop SETGID --cap-drop SETUID nginx
```

This would start the nginx container and would drop the SETGID and SETUID capabilities for the container.

Docker's end goal is to map the root user to a non-root user that exists on the Docker host. They also are working towards allowing the Docker daemon to run without requiring root privileges. These future improvements will only help facilitate how much focus Docker does take when they are implementing their feature sets.

### Protecting the Docker daemon

To protect the Docker daemon even more, we can secure the communications that our Docker daemon is using. We can do this by generating certificates and keys. There are are few terms to understand before we dive into the creation of the certificates and keys. A Certificate Authority (CA) is an entity that issues certificates. This certificate certifies the ownership of the public key by the subject that is specified in the certificate. By doing this, we can ensure that your Docker daemon will only accept communication from other daemons that have a certificate that was also signed by the same CA.

Now, we will be looking at how to ensure that the containers you will be running on top of your Docker hosts will be secure in a few pages; however, first and foremost, you want to make sure the Docker daemon is running securely. To do this, there are some parameters you will need to enable for when the daemon starts. Some of the things you will need beforehand will be as follows:

1. Create a CA.

```bash
$ openssl genrsa -aes256 -out ca-key.pem 4096
Generating RSA private key, 4096 bit long modulus
..................................................................
..................................................................
..................................................................
................++
..................................................................
..++
e is 65537 (0x10001)
```

Enter pass phrase for ca-key.pem:

Verifying - Enter pass phrase for ca-key.pem:
Securing Docker Hosts

You will need to specify two values, pass phrase and pass phrase. This needs to be between 4 and 1023 characters. Anything less than 4 or more than 1023 won't be accepted.

```bash
$ openssl req -new -x509 -days <number_of_days> -key ca-key.pem -sha256 -out ca.pem

Enter pass phrase for ca-key.pem:

You are about to be asked to enter information that will be incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank.

For some fields there will be a default value,
If you enter '.', the field will be left blank.

-----

Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Pennsylvania
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

There are a couple of items you will need. You will need pass phrase you entered earlier for ca-key.pem. You will also need the Country, State, city, Organization Name, Organizational Unit Name, fully qualified domain name (FQDN), and Email Address to be able to finalize the certificate.

2. Create a client key and signing certificate.

```bash
$ openssl genrsa -out key.pem 4096
$ openssl req -subj '/CN=<client_DNS_name>' -new -key key.pem -out client.csr
```

3. Sign the public key.

```bash
$ openssl x509 -req -days <number_of_days> -sha256 -in client.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -out cert.pem
```

   $ chmod -v 0400 ca-key.pem key.pem server-key.pem
   $ chmod -v 0444 ca.pem server-cert.pem cert.pem

Now, you can make sure that your Docker daemon only accepts connections from the other Docker hosts that you provide the signed certificates to:

   $ docker daemon --tlsverify --tlscacert=ca.pem --tlscert=server-certificate.pem --tlskey=server-key.pem -H=0.0.0.0:2376

Make sure that the certificate files are in the directory you are running the command from or you will need to specify the full path to the certificate file.

On each client, you will need to run the following:

   $ docker --tlsverify --tlscacert=ca.pem --tlscert=cert.pem --tlskey=key.pem -H=<$DOCKER_HOST>:2376 version

Again, the location of the certificates is important. Make sure to either have them in a directory where you plan to run the preceding command or specify the full path to the certificate and key file locations.

You can read more about using Transport Layer Security (TLS) by default with your Docker daemon by going to the following link:

http://docs.docker.com/engine/articles/https/

For more reading on Docker Secure Deployment Guidelines, the following link provides a table that can be used to gain insight into some other items you can utilize as well:


Some of the highlights from that website are:

- Collecting security and audit logs
- Utilizing the privileged switch when running Docker containers
- Device control groups
- Mount points
- Security audits
Securing Docker Hosts

Securing Docker hosts

Where do we start to secure our hosts? What tools do we need to start with? We will take a look at using Docker Machine in this section and how to ensure the hosts that we are creating are being created in a secure manner. Docker hosts are like the front door of your house, if you don't secure them properly, then anybody can just walk right in. We will also take a look at Security-Enhanced Linux (SELinux) and AppArmor to ensure that you have an extra layer of security on top of the hosts that you are creating. Lastly, we will take a look at some of the operating systems that support and do auto patching of their operating systems when a security vulnerability is discovered.

Docker Machine

Docker Machine is the tool that allows you to install the Docker daemon onto your virtual hosts. You can then manage these Docker hosts with Docker Machine. Docker Machine can be installed either through the Docker Toolbox on Windows and Mac. If you are using Linux, you will install Docker Machine through a simple curl command:

```
$ curl -L https://github.com/docker/machine/releases/download/v0.6.0/
docker-machine-"uname -s"-"uname -m" > /usr/local/bin/docker-machine && 
$ chmod +x /usr/local/bin/docker-machine
```

The first command installs Docker Machine into the /usr/local/bin directory and the second command changes the permissions on the file and sets it to executable.

We will be using Docker Machine in the following walkthrough to set up a new Docker host.

Docker Machine is what you should be or will be using to set up your hosts. For this reason, we will start with it to ensure your hosts are set up in a secure manner. We will take a look at how you can tell if your hosts are secure when you create them using the Docker Machine tool. Let's take a look at what it looks like when you create a Docker host using Docker Machine, as follows:

```
$ docker-machine create --driver virtualbox host1
```

Running pre-create checks...
Creating machine...
Waiting for machine to be running, this may take a few minutes...
Machine is running, waiting for SSH to be available...
Detecting operating system of created instance...
Provisioning created instance...
Copying certs to the local machine directory...
Copying certs to the remote machine...

Setting Docker configuration on the remote daemon...

From the preceding output, as the create is running, Docker Machine is doing things such as creating the machine, waiting for SSH to become available, performing actions, copying the certificates to the correct location, and setting up the Docker configuration, we will see how to connect Docker to this machine as follows:

```
$ docker-machine env host1
```

```
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/scottpgallagher/.docker/machine/machines/host1"
export DOCKER_MACHINE_NAME="host1"
# Run this command to configure your shell:
# eval "$(docker-machine env host1)"
```

The preceding command output shows the commands that were run to set this machine up as the one that Docker commands will now run against:

```
eval "$(docker-machine env host1)"
```

We can now run the regular Docker commands, such as `docker info`, and it will return information from `host1`, now that we have set it as our environment.

We can see from the preceding highlighted output that the host is being set up securely from the start from two of the export lines. Here is the first highlighted line by itself:

```
export DOCKER_TLS_VERIFY="1"
```

From the other highlighted output, `DOCKER_TLS_VERIFY` is being set to 1 or `true`. Here is the second highlighted line by itself:

```
export DOCKER_HOST="tcp://192.168.99.100:2376"
```

We are setting the host to operate on the secure port of 2376 as opposed to the insecure port of 2375.
We can also gain this information by running the following command:

```bash
$ docker-machine ls
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>ACTIVE</th>
<th>DRIVER</th>
<th>STATE</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWARM</td>
<td>*</td>
<td>virtualbox</td>
<td>Running</td>
<td>tcp://192.168.99.100:2376</td>
</tr>
</tbody>
</table>

Make sure to check the TLS switch options that can be used with Docker Machine if you have used the previous instructions to set up your Docker hosts and Docker containers to use TLS. These switches would be helpful if you have existing certificates that you want to use as well. These switches can be found in the highlighted section by running the following command:

```bash
$ docker-machine --help
```

Options:

- `-debug, -D` Enable debug mode
- `-s, --storage-path */Users/scottpgallagher/.docker/machine*` Configures storage path
- `--tls-ca-cert` CA to verify remotes against
- `--tls-ca-key` Private key to generate certificates
- `--tls-client-cert` Client cert to use for TLS
- `--tls-client-key` Private key used in client TLS auth
- `--github-api-token` Token to use for requests to the Github API
- `--native-ssh` Use the native (Go-based) SSH implementation.
- `--help, -h` show help
- `--version, -v` print the version

You can also regenerate TLS certificates for a machine using the `regenerate-certs` subcommand in the event that you want that peace of mind or that your keys do get compromised. An example command would look similar to the following command:

```bash
$ docker-machine regenerate-certs host1
```

Regenerate TLS machine certs? Warning: this is irreversible. (y/n): y
Regenerating TLS certificates
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...

SELinux and AppArmor
Most Linux operating systems are based on the fact that they can leverage SELinux or AppArmor for more advanced access controls to files or locations on the operating system. With these components, you can limit a container's ability to execute a program as the root user with root privileges.

Docker does ship a security model template that comes with AppArmor and Red Hat comes with SELinux policies as well for Docker. You can utilize these provided templates to add an additional layer of security on top of your environments.

For more information about SELinux and Docker, I would recommend visiting the following website:
https://www.mankier.com/8/docker_selinux

While, on the other hand, if you are in the market for some more reading on AppArmor and Docker, I would recommend visiting the following website:
https://github.com/docker/docker/tree/master/contrib/apparmor

Here you will find a template.go file, which is the template that Docker ships with its application that is the AppArmor template.

Auto-patching hosts
If you really want to get into advanced Docker hosts, then you could use CoreOS and Amazon Linux AMI, which perform auto-patching, both in a different way. While CoreOS will patch your operating system when a security update comes out and will reboot your operating system, the Amazon Linux AMI will complete the updates when you reboot. So, when choosing which operating system to use when you are setting up your Docker hosts, make sure to take into account the fact that both of these operating systems implement some form of auto-patching, but in a different way. You will want to make sure you are implementing some type of scaling or failover to address the needs of something that is running on CoreOS so that there is no downtime when a reboot occurs to patch the operating system.
Summary

In this chapter, we looked at how to secure our Docker hosts. The Docker hosts are the first line of defense as they are the starting point where your containers will be running and communicating with each other and end users. If these aren’t secure, then there is no purpose of moving forward with anything else. You learned how to set up the Docker daemon to run securely running TLS by generating the appropriate certificates for both the host and the clients. We also looked at the virtualization and isolation benefits of using Docker containers, but make sure to remember the attack surface of the Docker daemon too.

Other items included how to use Docker Machine to easily create Docker hosts on secure operating systems with secure communication and ensure that they are being set up using secure methods when you use it to set up your containers. Using items such as SELinux and AppArmor also help to improve your security footprint as well. Lastly, we covered some Docker host operating systems that you can use for autopatching as well, such as CoreOS and Amazon Linux AMI.

In the next chapter, we will be looking at securing the components of Docker. We will focus on securing the components of Docker such as the registry you can use, containers that run on your hosts, and how to sign your images.
Where to buy this book
You can buy Securing Docker from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.