Ethical Student Hackers

Docker

The Legal Bit

- The skills taught in these sessions allow identification and exploitation of security vulnerabilities in systems. We strive to give you a place to practice legally, and can point you to other places to practice. These skills should not be used on systems where you do not have explicit permission from the owner of the system. It is VERY easy to end up in breach of relevant laws, and we can accept no responsibility for anything you do with the skills learnt here.

- If we have reason to believe that you are utilising these skills against systems where you are not authorised you will be banned from our events, and if necessary the relevant authorities will be alerted.

- Remember, if you have any doubts as to if something is legal or authorised, just don't do it until you are able to confirm you are allowed to.

Code of Conduct

- Before proceeding past this point you must read and agree to our Code of Conduct - this is a requirement from the University for us to operate as a society.
- If you have any doubts or need anything clarified, please ask a member of the committee.
- Breaching the Code of Conduct = immediate ejection and further consequences.
- Code of Conduct can be found at https://shefesh.com/downloads/SESH%20Code%20of%20Conduct.pdf

What is docker?

“an open-source project that automates the deployment of software applications inside ***containers*** *by providing an additional layer of abstraction and automation of* ***OS-level virtualization*** *on Linux.”*

But what does that really mean?

Docker is a tool that allows you to deploy sandboxed applications (known as *containers*) that can be run on the host operating system.

This means that you can package an application alongside all of its dependencies into a standardized unit.

In simple terms, instead of shipping the application, you ship them the computer.

Why should I use docker?

“Well it works on my machine” - So ship them your machine! Docker ensures that the packages, libraries and other softwares that your application relies upon are the same between your machine and other peoples

Easier deployment - Docker transforms the 15 pre-deployment commands you need to run to make sure things are up to date, migrations are run and files are moved to the right locations into 1 file and 1 command, making it easier to get deployments out quicker to your users, it also means that if you need to update your deployment steps, you can do that and have them used without having to make sure everyone knows the new way of doing it.

Containers allow for fine control over resource allocation which can result in lower costs overall

Containers allow the host system to remain safe even when a container is compromised (Most of the time)

Starting a container from an image

Firstly, we need to find an image to run! We will be using the official https://hub.docker.com/ website to find images that we can run.

Most docker containers on Docker Hub have instructions on how to properly run the container, as the instructions can differ depending on the application.

For example, `docker run -it -v $(pwd)/data:/data -p 25565:25565 -e EULA=true --name mc_server cmunroe/bukkit`

-it	- Use an interactive terminal, it allows us to type commands
-v	- Use a volume, allowing us to share a folder between the host and the docker container
-p	- Translate the port from container port to host port (So we can access from localhost)
-e	- Pass an environment variable to the container
--name	- Name the image to something we can remember
cmunroe/bukkit	- The actual name of the container on Docker Hub

Tags

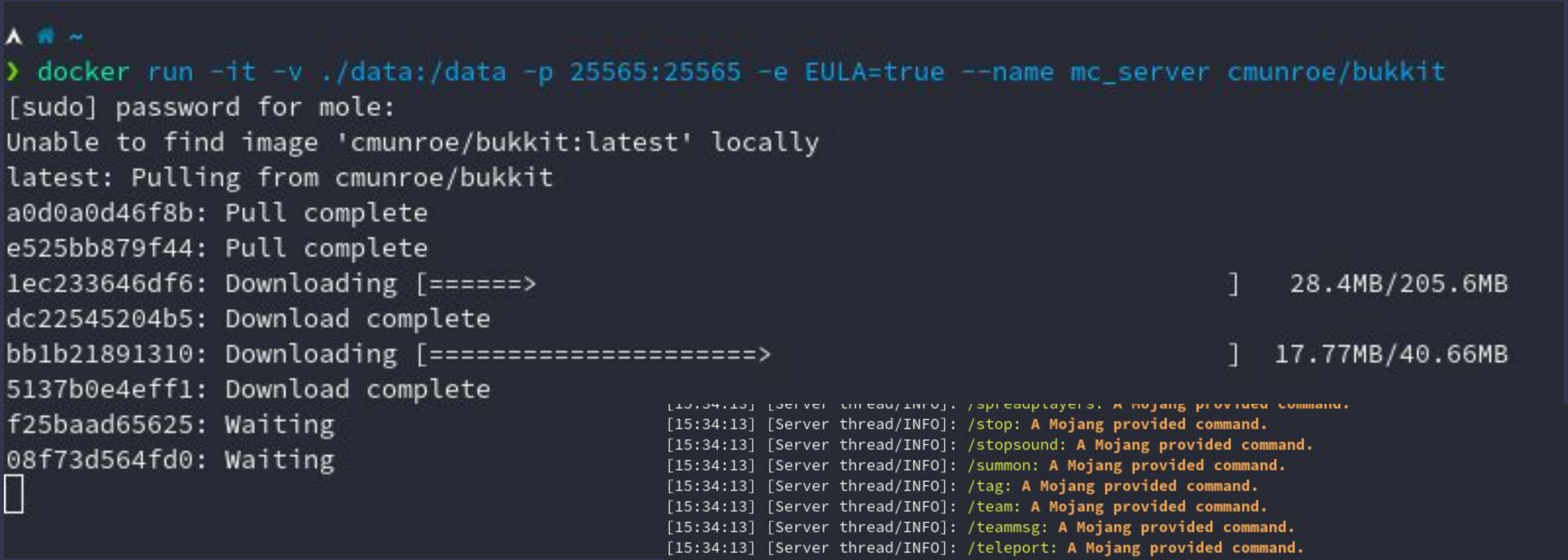
Docker uses a tag system, similar to Git's branches

They allow us to use different versions of the same project

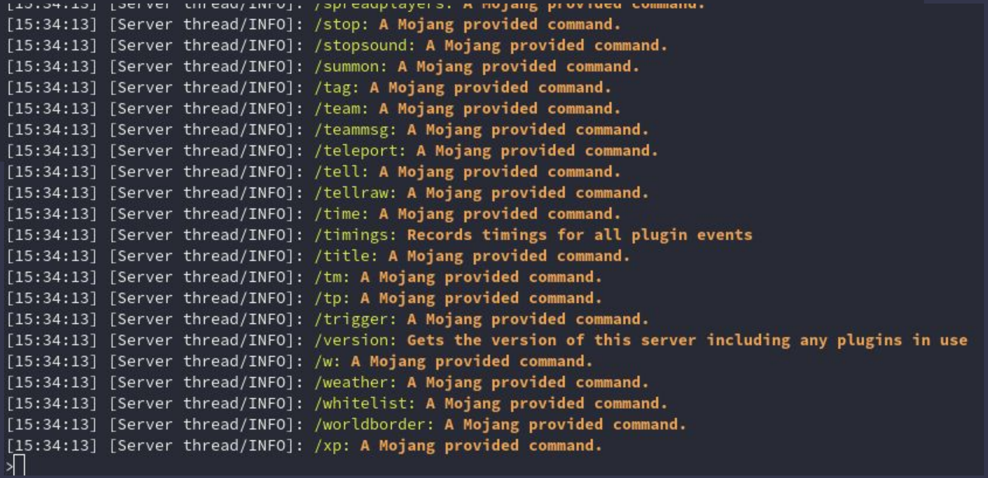
In terms of the `cmunroe/bukkit` image, we can use `cmunroe/bukkit:1.17` and `cmunroe/bukkit:1.16` to specify the 1.17 and 1.16 versions of the Minecraft container that we want to use.

When no tag is specified, the 'latest' tag is used by default (`cmunroe/bukkit:latest`). 'Latest' doesn't necessarily mean it's the newest version, the developer can tag any version as the 'latest'.

Layers



```
^ ~
> docker run -it -v ./data:/data -p 25565:25565 -e EULA=true --name mc_server cmunroe/bukkit
[sudo] password for mole:
Unable to find image 'cmunroe/bukkit:latest' locally
latest: Pulling from cmunroe/bukkit
a0d0a0d46f8b: Pull complete
e525bb879f44: Pull complete
1ec233646df6: Downloading [======>                                            ]   28.4MB/205.6MB
dc22545204b5: Download complete
bb1b21891310: Downloading [=====================>                             ]  17.77MB/40.66MB
5137b0e4eff1: Download complete
f25baad65625: Waiting
08f73d564fd0: Waiting
```



```
[15:34:13] [Server thread/INFO]: /spreadplayers: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /stop: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /stopsound: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /summon: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /tag: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /team: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /teammsg: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /teleport: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /tell: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /tellraw: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /time: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /timings: Records timings for all plugin events
[15:34:13] [Server thread/INFO]: /title: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /tm: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /tp: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /trigger: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /version: Gets the version of this server including any plugins in use
[15:34:13] [Server thread/INFO]: /w: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /weather: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /whitelist: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /worldborder: A Mojang provided command.
[15:34:13] [Server thread/INFO]: /xp: A Mojang provided command.
>
```

Starting an existing container

We can view running containers with `sudo docker ps`, and view all containers (including stopped) with `sudo docker ps -a`.

Starting a stopped container is as simple as running `sudo docker start [container]`. This will run the container in the background.

- -a - Attach STDOUT/STDERR and forward signals
- -i - Interactive, allows us to type commands to the container

If the the application process we tell docker to run, e.g. java for minecraft, terminates or dies then the docker container stops. We can, however, tell docker to restart if the process dies.

We can also stop a container with the command line, by simply using `sudo docker stop [container]`.

Backgrounded containers

If we have a backgrounded container and we want to have a look around the file system or perform an upgrade, we can!

Using the `sudo docker attach [container]`, we can drop into the main process of the docker container. In the case of the minecraft container, it means we can issue minecraft commands.

Using `sudo docker exec -it [container] [terminal]`, we can drop into a new terminal on the container.

-	-it	- Specify for the terminal to be interactive
-	-u	- The user to login as, e.g. root
-	-e	- Environment variables to pass in

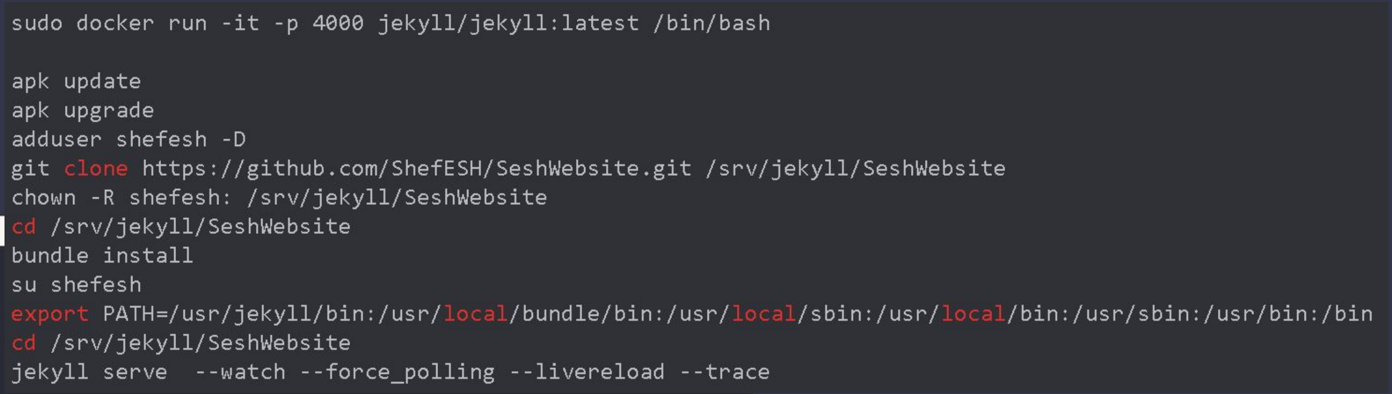
The Dockerfile!

The Dockerfile is the centerpiece of Docker, it is one of the scripts that tells the Docker daemon how to construct our containers.

Docker uses a combination of its own syntax and also bash to setup the environment.

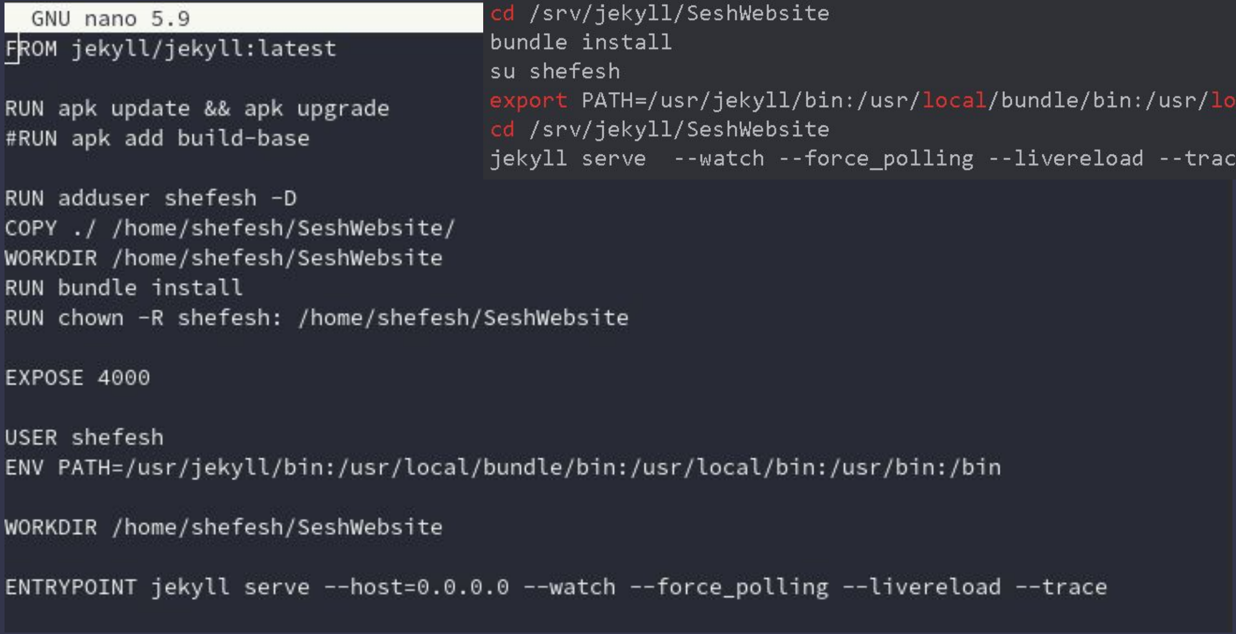
FROM	- Specify the base image to use, e.g. ubuntu or flask
COPY	- Copy files from the host to the container
RUN	- runs a command as a user, by default it runs as root in bash
WORKDIR	- Specify the working directory
USER	- Specify the user to change to
ENV	- Set an environment variable
EXPOSE	- Allow a port to be connected to
ENTRYPOINT	- The script to run once the container has started

An example of a Dockerfile



```
sudo docker run -it -p 4000 jekyll/jekyll:latest /bin/bash

apk update
apk upgrade
adduser shefesh -D
git clone https://github.com/ShefESH/SeshWebsite.git /srv/jekyll/SeshWebsite
chown -R shefesh: /srv/jekyll/SeshWebsite
cd /srv/jekyll/SeshWebsite
bundle install
su shefesh
export PATH=/usr/jekyll/bin:/usr/local/bundle/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/bin
cd /srv/jekyll/SeshWebsite
jekyll serve  --watch --force_polling --livereload --trace
```



```
  GNU nano 5.9
FROM jekyll/jekyll:latest

RUN apk update && apk upgrade
#RUN apk add build-base

RUN adduser shefesh -D
COPY ./ /home/shefesh/SeshWebsite/
WORKDIR /home/shefesh/SeshWebsite
RUN bundle install
RUN chown -R shefesh: /home/shefesh/SeshWebsite

EXPOSE 4000

USER shefesh
ENV PATH=/usr/jekyll/bin:/usr/local/bundle/bin:/usr/local/bin:/usr/bin:/bin

WORKDIR /home/shefesh/SeshWebsite

ENTRYPOINT jekyll serve --host=0.0.0.0 --watch --force_polling --livereload --trace
```

Building a Dockerfile

Once we have a Dockerfile available to us, we can use it to generate an image that we can run. To do this, we use the `docker build` command.

Some common flags for building are:

-	-t	- Assign a tag (identifier/name) to the image so we can easily reference it
-	-f	- The location of the Dockerfile, if -f isn't specified then the one in the current directory is used

There are also some flags that allow us to specify the maximum system resources utilised by the container.

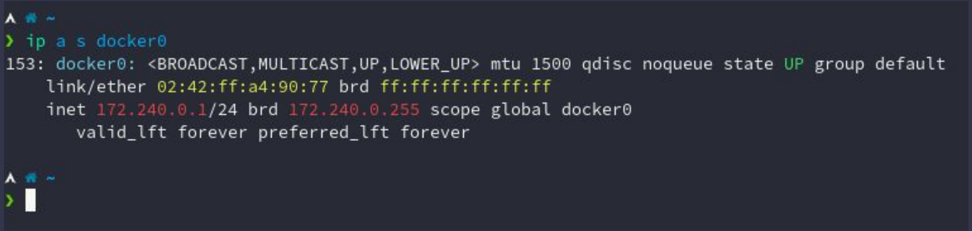
Docker Networks

Docker utilises its own networking system, as all the containers are effectively independent operating systems.

Each container is placed on a subnet setup by docker, meaning the containers are able to speak to each other on this subnet. Earlier we mentioned that we can use the -p flag when running a container to forward the port to our localhost network.

We can specify the network for a container to use when using the `docker run` command with --network.

Docker by default has 3 networks, a host only adapter, a bridge adapter and a 'none' adapter. The bridge adapter is used by default by all containers, but we can create new adapters with the `docker network create` command.



```
λ ~
› ip a s docker0
153: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ff:a4:90:77 brd ff:ff:ff:ff:ff:ff
    inet 172.240.0.1/24 brd 172.240.0.255 scope global docker0
       valid_lft forever preferred_lft forever

λ ~
› 
```

Demo time!

Moving a project to Docker

We are going to move an existing git repository into a docker container, so that everyone on committee can easily run the website when developing. https://github.com/ShefESH/SeshWebsite

Firstly, we will use a Dockerfile to create a script to move all the code to a Docker container, install the environment.

Then we'll show how you can use docker-compose to spin up a couple of containers at the same time.

Vulnerabilities when using Docker

Docker, by default, is relatively secure. However there are some misconfigurations that can cause Docker to become insecure.

As Docker utilises the host's kernel within the containers, if there is a vulnerability in the host kernel then it means that a container can potentially exploit the host and gain remote code execution on the host.

This is very bad :/

A simple solution to this is to keep your software updated!

More vulnerabilities

There are also some configurations that can make Docker vulnerable.

If you are a member of the docker group, then you can easily gain root! Simply exploit the volumes ability to mount the host operating system to a container. *Note:* this is done from *outside* the container

There are also ways to *break out* of a container:

- Running a docker container with --privileged can allow the container to interact with host ports, capabilities and overall lead to code execution on the host as root. This is VERY bad!
    - To mitigate this, resist using the --privileged flag. Instead, individually assign the minimum amount of capabilities that the container needs to function with --cap-add.
- Various docker capabilities are dangerous and can be used for privilege escalation, such as CAP_SYS_ADMIN, CAP_SYS_MODULE, and DAC_OVERRIDE

More Docker Breakout Techniques

If the docker socket (find / -name docker.sock 2>/dev/null) is mounted inside the container you can use it to escape

You may also be able to find dockerfile or docker-compose.yml files that leak credentials

Finally, some versions of Docker are vulnerable to CVE-2019-5736, which can be abused if Docker can be run as root

More details:
https://book.hacktricks.xyz/linux-unix/privilege-escalation/docker-breakout/docker-breakout-privilege-escalation

Places to practice: Laboratory, Ready, TheNotebook and Monitors on hackthebox.eu

Upcoming sessions

What's up next?

www.shefesh.com/sessions

CompSoc:

17 Nov 2021 - YII Pt. 2 Workshop

24 Nov 2021 - YII Pt. 3 Drop In Session

02 Dec 2021 - C++ Workshop

SESH:

22 Nov 2021 – Shells

29 Nov 2021 – Privilege Escalation

06 Dec 2021 – Hack the Box

Any Questions?

SHEFFIELD ETHICAL STUDENT HACKERS

www.shefesh.com

Thanks for coming!