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

Mairi MacLeod
School of Design and Informatics
Abertay University
DUNDEE, DD1 1HG, UK

ABSTRACT

Context: Virtualisation, especially containers, is becoming increasingly popular as more businesses are moving their products to the cloud. This is in a large part due to the fact that new microservice and smaller legacy applications suit containerisation more than being put into a virtual machine. However with container based environments the big question is; where do the vulnerabilities lie and how can they be mitigated against?
Aim: This paper aims to compare different methods of escaping into the host environment from a container and will discuss the methodology along with an assessment of their ease of exploitation.
Method: A range of containers with different base images and security measures, from the CIS benchmark, will be created using Docker. These will then be used to demonstrate a range of container escape techniques.
Results: It is expected that the exploits demonstrated will allow for access to the host operating system from within the container. These techniques will then be evaluated, as well as the security measures on their success in preventing a container escape.
Conclusion: This project will attempt to demonstrate that containers in their current state are insecure and vulnerable to breakout exploitation. It will also assess security measures that can be taken and their effectiveness against a potential breakout attack.

Keywords

Docker, Container-Based Virtualisation, Container Breakouts, Virtualisation

1. INTRODUCTION

Virtualised environments have existed since the late 1960's and have only increased in popularity since. This can be, in some part, attributed to the creation of container-based virtualisation. The appeal of containers lies in their ability to provide a lightweight and portable solution for deploying applications. Most organisations that offer cloud hosting solutions now provide their own container-hosting service, making containers more attractive and accessible for businesses. Docker itself boasts the patronage of Netflix, Paypal, Verizon and many other large corporations. Although they are convenient to use, containers are not as secure as some might think and are vulnerable to a large array of potential exploits (Yasrab, 2018). This makes it imperative that the vulnerabilities are understood and security countermeasures implemented wherever possible.

Before discussing containers further, virtualised environments must first be defined. These allow users to run an operating system and or pieces of software in an entirely isolated partition of the host machines system (Bui, 2015). One benefit of virtualisation is that it allows for software to be transported with all its dependencies, which allows for a simple and easy deployment on any machine.

1.1 Hypervisor-Based Virtualisation

Hypervisor-Based virtualisation involves a piece of software called a hypervisor managing a range of emulated hardware that hosts an entirely separate operating system and kernel to the host machine's. As a result of this, Virtual Machines tend to be much larger than their container-based counterparts.

Hypervisors often have much greater hardware requirements and are considerably less portable than containers. However as Virtual Machines are completely isolated from the host machine, breakout vulnerabilities are less common and more difficult to exploit.

1.2 Container-Based Virtualisation

Container-Based virtualisation, the focus of this paper, places a virtualised environment on top of the kernel. Although containers exist within the host machine and share the kernel they are isolated using control groups, cgroups, and namespacing. Cgroups are a feature on the Linux kernel that controls and tracks resource allocation for processes. As a result containers can be configured to share resources with the host machine. If complete isolation from the host OS is desired however the container will need to be given its own resources in order to run. The main benefits of containers is the fact that they do not create their own kernel and so are much more lightweight than a traditional Hypervisor-Based Virtual Machine. As a result of this multiple containers can be stored in the same amount of memory as one virtual machine (Bui, 2015). Through the reading of papers on this topic it appears that as containers are connected to the host machine directly, they are considerably more vulnerable to breakouts and are widely considered the less secure option of the two. The figure below shows the structural differences between the two types of virtualisation.

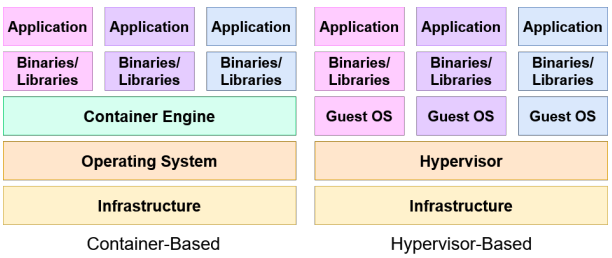

Figure 1: Structure of a Container vs a Virtual Machine

1.3 Docker

Created in 2013 and now synonymous with Container-Based virtualisation, Docker offers a command-line interface which allows the user to create and manage containers. Two main 'products' provided by this organisation are Docker Engine and Docker Hub. Docker Hub claims to be "the worlds largest repository of container images" and stores both public and private images. Docker containers are created by adding layers onto images so that the original image is not altered.

Although originally created to run exclusively on the Linux kernel Docker now provides container support Windows. These containers can be run through the hypervisor Hyper-V or Windows Subsystem for Linux (WSL) to create a faux Linux environment for the container to run in. These both produce the same results and which one to use depends on the users version of Windows.

Although containers are most commonly associated with Docker it is important to note that they did not invent the concept nor do they own it. Other container building and management services exist as well as methods for creating them entirely from scratch without the use of any such service.

The goal of this project is to investigate container breakout techniques on different Docker images. This is to ascertain whether the standard container security measures on a plain image are effective and what steps users can take in order to better secure their applications with Docker.

2. BACKGROUND

Than Bui's 'Analysis of Docker Security' (Bui, 2015) investigates Docker's internal security as well as how it interacts with security features in the Linux kernel. Bui states that using containers a user could fit ten times as many virtual environments within the space that could store one full virtual machine. Within the section 'Docker Security Analysis' the author goes into detail about the internal security methods used by the container management service. These are; process isolation, filesystem isolation, device isolation, IPC (inter-process communication) isolation, network isolation and the limiting of resources using c-groups. These all demonstrate ways that Docker interacts with Linux in order to separate its containers as much as possible from the host operating system. C-groups, for example, allow for restriction of access to resources, which can prevent Denial of Service attacks. All of the isolation security measures, apart from device isolation which suggests the use of device whitelist c-groups, are achieved using various Linux namespace creations. The next part of this paper discusses the Linux Security Modules that are integrated into the official kernel, AppArmour and SELinux. SELinux uses labels for directories and files which allow the administrator to write policies for the following three categories; Type, Multi-category security and Multi-level security enforcement. These restrictions can prevent a process that has been exploited from compromising any others. AppArmor works in a similar way to SELinux but rather than setting rules for directories the rules are set for individual applications. Docker, unless the administrator sets their own, upon the creation of a container creates a default AppArmor profie for it.

'Automatic security hardening of Docker containers using Mandatory Access Control, (Dimou, 2019) specialized in defending isolation' by Fani Dimou is a thesis about the development and testing of their open-source tool SecureWilly. The tool is designed to automate the production of AppArmor profiles for Docker containers. As with Bui's paper, Dimou goes into detail about the importance of Linux namespacing, although they also mention that this can be disabled and that not everything within the kernel is namespaced. Mandatory Access Control, MAC, gives the OS power to restrict certain processes from performing specific actions. MAC is crucial for preventing container escapes as it can restrict sudo access. DAC or Discretionary Access Control is what AppArmor adds its profile rules to, so that access cannot be granted but further restricted. It works like MAC but rather than restricting actions based on levels of access it restricts them based on the user who is trying to perform them. Dimou then suggests that permitting the following within a container will leave it vulnerable to breakouts; SYS_ADMIN, SETUID, SETGID, SYS_CHROOT, SYS_PTRACE and DAC_OVERRIDE. When testing their application they focused mainly on attempting to disable namespacing for the container. This was done by disabling certain flags such as pid, which allows the container to see all of the processes currently running in the host, and uts which allows for the complete control of the host machines domain information. Another method is to mount the namespace files onto the container using volumes, whos contents persist even after the container is stopped. However using a bind mount instead of a volume is preferential to an attacker as you can decide where the created directory is stored, as opposed to a volume where it will always be created in /var/lib/docker/volumes where the user could find it easily. The final way of disabling or attacking namespaces was to use the tool Nsenter. It lets the user gain access to a processes namespace, although this requires sudo access. This means that it should be run alongside another attack which would give the container root access or that the container has been run as root by the user in the first place. In order to escalate the privilege level of the attacker Dimou uses a script that accesses a shell file within a mounted volume and changes the permissions there.

The paper 'Mitigating Docker Security Issues' by Robail Yasrab (Yasrab, 2018) discusses how it is a common misconception that due to the perceived separation from the host machine that all virtual environments are secure. Yasrab then writes about the origins of Docker, started in 2013 by Solomon Hykes as a project for the platform-as-a-service organisation dotCloud. Originally Docker used the Linux kernel as its execution environment but created its own Go library for this in 2014. The paper then goes into a security analysis of Docker, where the author writes about how container based applications have access to the host machine's kernel which can make the container vulnerable to breakouts. The potential security attacks that could be used from within a container are; kernel exploits, Denial of Service attacks, container breakouts, poisoned images, compromised secrets, Man in the Middle and ARP Spoofing. The author then suggests that a user can mitigate against these potential attacks with the following solutions: Image maintainers should provide an AppArmour profile and SELinux policy module with their image. Users should only download an image if they can verify their cryptographic signature, to avoid downloading a poisoned image. Processes should be logged and tracked to monitor any unwanted activity. SUID and GUID libraries should be dis-

abled to prevent buffer overflows. Run any sensitive containers inside a virtual machine or to fully virtualise the container using a service such as KVM.

3. METHOD

3.1 Research

Before starting the practical elements of this dissertation the author will first gather more information on the specifics of container vulnerabilities and how to exploit them in order to gain access to the host operating system. The main areas of research are; disabling the current security measures that have been implemented by Docker and Linux and how to exploit any CVEs and known breakout attacks. For bypassing security measures, Linux namespacing and AppArmour profiles appear to be the two main security measures that must be disabled to break out of the container in order to gain access to the host machine. Privilege escalation is very important at this stage. Methods to achieve this will also be researched and papers discussing the techniques will be read. Since Docker is always patching and updating the security of its service new vulnerabilities may be the authors best chance at achieving a breakout. To monitor new zero-day attacks an RSS feed for Docker CVEs will be setup. Although older breakout exploits will also be investigated in case the recent versions of Docker are still vulnerable to these methods.

To properly test these breakout techniques some security measures will need to be put in place to scientifically assess the attack's effectiveness and the containers vulnerability to it. Techniques for increasing security on a container and the kernel will be read up on in order to create a tier system for how secure a container is.

3.2 Disabling Namespaces

The techniques to be used in this section are heavily inspired by the testing done on the automated AppArmour profiler, SecureWilly. (Dimou, 2019) The first step will be to create multiple control containers, these will be created from different image types, for example from Ubuntu 18.04 and Python 3 images. The reason different image types will be used is to investigate if there are different vulnerabilities on an OS image compared to a programming language one.

The next step is to try and disable namespaces by making the following flags equal host on runtime; -pid to make the host's processes visible to the container, -net which allows the container to access the host's network resources and listen to ports that are being used by other containers, -uts allows the container to change the host domain information, -ipc which lets the container use the host's Interprocess Communication resources and -userns disables the user namespaces. The host filesystem will also be mounted using a bind mount in order to disable the mount namespace and give the container access to the host's filesystem.

3.2.1 Nsenter

A part of the package util-linux from 2.23, this tool allows for the alteration of other processes namespaces. Since this tool does require sudo access it will be used in conjunction with the section 3.4 Privilege Escalation. The author will create a bash script to be run from within the container that will run this tool and allow them to view and breakout into the host's root directory. This can be achieved by using Nsenter to create a new AppArmour profile with sudo privileges.

3.3 Common Vulnerabilities and Exposures

For this section of the methodology recent CVEs will be investigated and exploited where possible. One that the author is currently aware of is CVE-2014-3519 exploit known as Shocker.c which was discovered in 2014. A proof of concept attack for this vulnerability can be found in Gabe Monroy's github repository shocker. (GitHUb, 2014) Another breakout technique, discovered by Palo Alto in 2019, CVE-2019-14271 exploits the cp command to give the container root access to the host and other containers running on the host machine.

3.4 Privilege Escalation

A key component of a container breakout is privilege escalation to root in order to allow the attacker access to all areas of the host machine. For this paper this will be done by adding capabilities to the container using -cap-add alongside -privileged. The author is aware that this may not work on all containers and depends heavily on the security features put in place so other methods will be researched and attempted if this does not work.

3.5 Securing Containers

In order to fairly assess these vulnerabilities and attacks different security countermeasures will be enabled to create a tier system. The measures applied that will make up these tiers will be taken from the CIS Docker benchmark (Center for Internet Security, 2019) The top tier will contain containers that have been as secured as much as the author can and they will go above the recommended requirements as much as possible. The bottom tier will be purposefully made insecure, to replicate if a user had set up their container badly by accident. In between these tiers various levels of security will be applied to try and imitate as many real world scenarios as possible. Some methods to try and secure these containers will be; removing bash to prevent commands being ran inside, setting resource limits to prevent access to host resources and running the container as -u instead of root.

4. SUMMARY

With the ever increasing adoption of containerisation for applications, both commercially and privately, an assessment of their security is more important than ever. Especially an assessment of breakout techniques that could leave a user's host machine vulnerable to exploitation. As legislation around data protection is tightening, it is imperative that businesses take their container security seriously and consider the implications of a breakout.

5. REFERENCES

Bui, T.(2015)*'Analysis of Docker Security'*, Finland: Aalto University School of Science

Dimou, F. (2019) *'Automatic security hardening of Docker containers using Mandatory Access Control, specialized in defending isolation'*, Greece: National Technical University of Athens

Center for Internet Security (2019). *CIS Docker Benchmark v1.2.0* [online]. Available at: https://www.cisecurity.org/ benchmark/docker/

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

GitHub (2014). *shocker* [online]. Available at: https://github.com/gabrtv/shocker (Accessed 06 October)