Trinity College Dublin

The University of Dublin

School of Computer Science and Statistics

# Intrusion Detection for Kubernetes Based Cloud Deployments

Macdara Tinney

[redacted]

May 8, 2020

A Dissertation submitted in partial fulfilment
of the requirements for the degree of
M.Sc. (Computer Science)

# Declaration

I, Macdara Tinney, hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: Macdara Tinney Date: 08/05/2020

# Abstract

The modern internet remains a hostile environment, with various malicious actors launching attacks against any internet facing system that shows any sign of vulnerability. While the former naïve approaches to computer security are being abandoned, the rate at which attacks grow in complexity continues to outpace developments in securing computer systems.

With the growth in cloud computing architectures such as Platform as a Service and Infrastructure as a Service, much of the progress made in securing traditional deployments is now not strictly applicable in the new paradigm. This risks opening a further gap between system implementations and the attackers attempting to break them. This could lead to a rise in security incidents in cloud environments unless something is done to mitigate the potential harm.

Cloud service providers are aware of this threat, and are actively working on securing deployments from external threats. This research proposes a system that can help secure a deployment from internal threats, via use of honeypots in containerised cloud deployments and listener containers that can detect when suspicious traffic such as port scanning is circulating within the cloud deployment.

Existing work in this area has already explored the use of honeypots and honeynets in this role in traditional deployment scenarios. It has also demonstrated them in a research role in the cloud. This work separates itself from this prior work by investigating the use of these intrusion detection systems in a containerised cloud environment, in particular a Kubernetes based cloud deployment, in combination with a port scanner detector.

# Acknowledgements

I would like to give a massive thank you to my supervisor, Dr. Stefan Weber, for all of the assistance he has given me throughout this process. He was always willing to give detailed explanations whenever I had an issue, and it is really appreciated. His patience and help has enabled me to complete this research.

To my friends, thanks for all of the support this year and every other year.

Finally, Mum, Dad and Cormac, thank you for keeping me sane and putting up with my nonsense for the past year. I couldn't have completed this dissertation without your support.

# Contents

# List of Figures

# 1 Introduction

The primary goal of this research is to explore methods to help secure Cloud Deployments, particularly from internal threats. The growth of cloud services and software as a service paradigms means that these services will inevitably face more and more attacks as they become more popular.

The end goal is to provide an easy to implement method to add an intrusion detection mechanism, that will detect intruders who have already gained access to a deployment and alert the relevant administrators.

## 1.1 Topic Area

An increasing number of modern computer systems are moving to cloud based hosting solutions. The growth in micro-service architectures, software as a service and related paradigms is helping to fuel this transition. Much has been said of the benefits these solutions provide to companies and developers, however the security implications of this shift have seen less discussion. With many businesses hyping such designs as the way of the future, while neglecting to discuss the changes this causes in security considerations, it is possible that we could see a regression in the security of internet facing services.

Such internet facing systems are already under the constant threat of an attack from malicious actors across the globe. The risk of state or state backed actors creating highly sophisticated attacks and exploits has also increased. Examples include the American NSA developed EternalBlue exploit, whose leak in 2017 led to the creation of several notorious pieces of malware including the WannaCry ransomware(1).

The cloud service providers such as AWS, Azure and Google Cloud are already focusing on helping their customers to secure their deployments from external intruders. The internal security of a deployment is less explored, however. If there are no internal security mechanisms in a deployment, then a malicious actor who has managed to gain access can remain undetected as they infiltrate the system. By the time administrators realise what is going on, it will already be too late to act. Adding some form of mitigation within a cloud deployment can buy administrators precious time to secure private data, or to pull the system down completely to prevent a leak.

The internal security of a traditional service deployment scenario has been explored in depth before. Concepts such as network sniffers, honeypots and honeynets have been in use in many existing deployments(2). It is worthwhile to see what elements of the traditional deployment scenario can be reused within a cloud deployment or a containerised environment.

## 1.2 Aims

This research began in the wake of many years of significant security breaches in the IT space. Once, many security incidents were caused by enthusiasts or people attempting to prank others. However, in today's world these groups constitute a very small minority. The modern threat landscape is dominated by criminals, criminal organisations and state actors, all of whom were already present on the scene, but who are now posing an even greater threat.

Due to the growth in the cloud computing paradigm, this research attempts to improve the security of such deployments. Motivated by this aim, the following goals were set.

- Explore the implications of an attacker gaining access to the internals of a cloud deployment
- Investigate possible mitigations for such a scenario, to lessen the negative impact of such events
- Propose an implementation which improves the security of cloud deployments against internal intruders, compared with the baseline implementation
- Have the system log and notify administrators of such suspicious events
- Have the proposed implementation be quick and easy to deploy into a cloud environment

## 1.3 Note

The nature of the security field means that academic sources often lag behind the cutting edge of the topic. Many of the leading security experts and researchers update their findings in online publications such as blogs or media articles, rather than academic journals. For this reason, many of the relevant sources cited in this dissertation are not cited from such journals.

Wherever online locations are used as a source, the date of the most recent access to the page in question is included.

## 1.4 Structure of the Document

The subsequent chapters cover the bulk of this research. Chapter 2 reviews the current state of virtualisation, cloud computing and the security threat landscape, and also looks at related work in the area. Chapter 3 describes how the research problem arose from these issues. Chapter 4 describes the design of the proposed system. Chapter 5 contains the details of the implementation process. Chapter 6 Evaluates the system and the research, and proposes areas of further research. Finally, Chapter 7 reviews what conclusions can be made as a result of this work.

# 2 State of the Art

This chapter discusses the modern state of cloud computing and containerisation. It also explores the current security landscape, threats, actors and how these relate to cloud computing in particular.

## 2.1 Virtual Machines

A virtual machine is a virtualised instance of a computer system (3). A typical virtual machine is launched from an image containing an Operating System and all of the libraries and binaries that come with a standard version of that Operating System. The nature of the virtualisation means that each virtual machine is isolated from other instances, and can only interact with the host machine and hardware peripherals via the hypervisor. The result of this is that any program running in a virtual machine should be unable to tell the difference between its environment and a non-virtualised one.

A virtual machine can be allocated varying amounts of computing resources. This makes them a good choice for cloud computing tasks, where a service provider can offer its users the ability to only pay for the resources that they use. Many VMs can be hosted on the same infrastructure, with customisable specifications. VM instances are the primary vehicle by which cloud computing is carried out.

The typical anatomy of a VM is described in Figure 2.1. Each virtual machine instance contains an image of its guest OS, along with any libraries and applications it requires to run. The hypervisor starts and stops these VMs and acts as an intermediary to between the host OS and the guests. The Host OS then has direct access to the underlying infrastructure.

### 2.1.1 Hypervisor

The hypervisor is the key interface between virtual machines and their host system. It can run on bare hardware, or in a host OS depending on the use case. In either case, it is the hypervisor which runs the guest OSes and translates the requests and instructions from

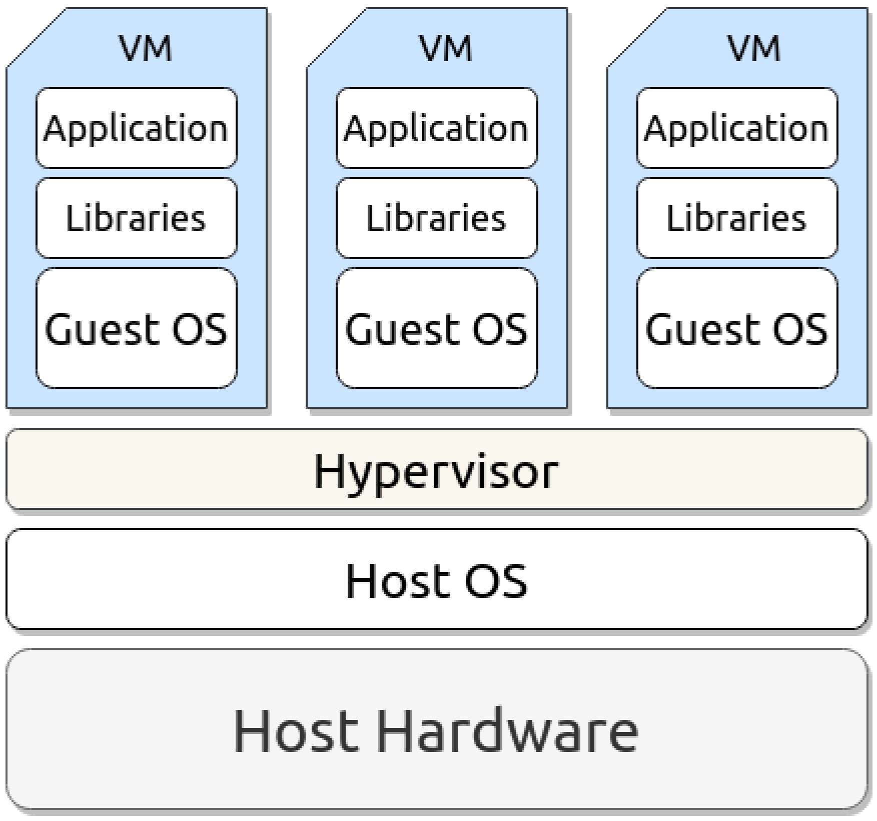


Figure 2.1: Anatomy of a typical virtual machine deployment

those guests to the host, thus enabling the virtualisation of the system.

### 2.1.2 Virtual Machine Security

One of the benefits in using virtual machines is that each instance is isolated from every other instance, with its own storage, OS and system binaries. This allows multiple guests to run on the same infrastructure without one guest being able to access another or corrupt its host. This is the principle by which virtual machines can securely enable the concept of a platform as a service.

However, if an attacker can escape the virtual machine and gain access to the host, or if they can access one VM from another, this advantage is completely lost. In fact, the system is now less secure than seperate non-virtualised deployments.

Attacks of this form have been shown to be possible. Such attacks have been dubbed a virtual machine escape. CVE-2009-1244 named "Cloudburst", is a vulnerability in VMware in which a malicious user could create a specially crafted video file, which would give them access to the host if executed inside a virtual machine(4). CVE-2018-12126 is a vulnerability called a Microarchitectural Data Sampling attack. This is a side channel attack targeting the CPU cache, which could enable the reading of data from a host or another virtual machine on the same host on a VM(5).

## 2.2 Containers

A container is a virtual, isolated computing environment. Similar to virtual machines, programs running within a containerised environment will believe that they are running on a full computer. Containers are virtualised at the OS level, unlike virtual machines. This means that containers running on the same host share the same kernel while being unaware of the other containers running on the same host. From each container's perspective, it is a stand alone user space environment with limited access to hardware or other devices(6).

Containers are bundled with any necessary libraries and other dependencies their applications require. This allows for ease of deployment without worrying about whether the target environment can run the client application or not. Container images themselves can be built upon other container images, this makes it easy for a developer to extend an existing container to suit their specific needs. The new container can be published, and other developers are now able to use it as a base to extend upon themselves.

The benefits of running applications in containers include many of the benefits of virtual machines. Programs can be run in isolation from their host environment and each other. An additional benefit is that they are more lightweight than an equivalent virtual machine(7). The container does not have to handle any of the kernel space tasks, nor does it need to include them. This reduces the amount of resources utilised. Additionally, multiple containers can share data across a shared file system if necessary. Containerisation offers improved portability of applications across different underlying operating systems and cloud infrastructure.

Figure 2.2 contains a diagram of a standard containerised deployment. Each container is more lightweight than a corresponding VM, as they only contain their applications and the necessary libraries to run them. The container engine manages the starting and stopping of the containers. All of the containers share access to the Host OS's kernel for kernel level tasks.

### 2.2.1 Container Engine

A container engine is the software which starts, stops and manages running container instances. It holds the responsibility of handling containers over their whole life-cycle. Often such software is bundled with features that have the ability to create a container image from a provided specification file or a running instance, and the ability to pull container images from an external repository to the host machine.

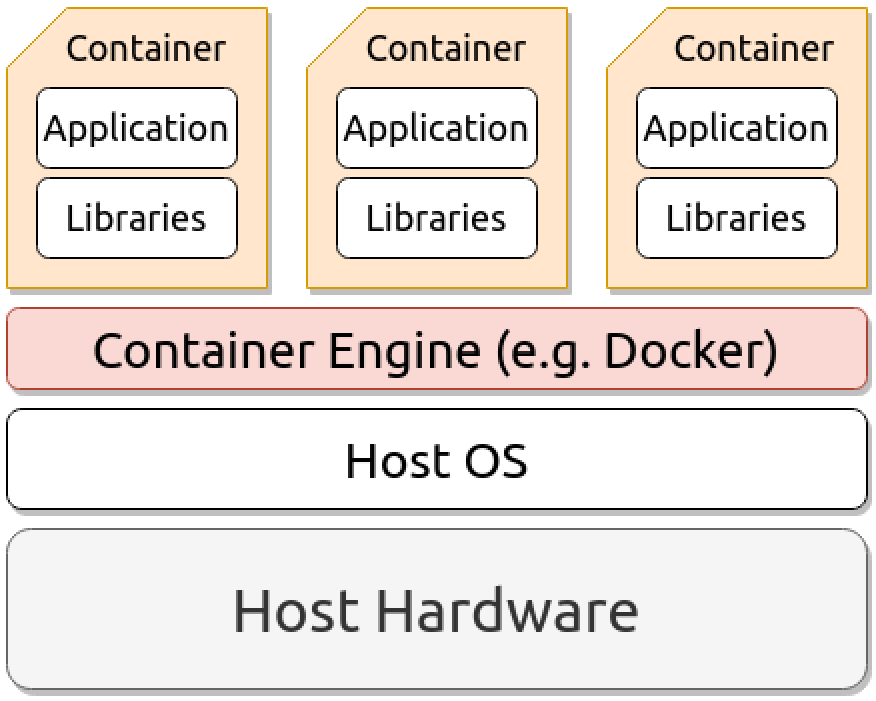


Figure 2.2: Anatomy of a containerised deployment

### 2.2.2 Container Security

The fact that containers are considerably more lightweight than VMs means that there is less of an issue in winding down an infected container and launching a new instance than there would be with a VM.

A potential negative to choosing to use a container over a virtual machine is the question of security. Containers are less removed from the host OS than virtual machines are, they have access to the shared kernel. Vulnerabilities in the parts of the kernel used by a container could provide a malicious actor a mechanism to launch a privilege escalation attack, and gain access to the host and every other container running on it.

## 2.3 Container Orchestration

Due to the complexities in managing large groups of containers in a deployment, container orchestration tools were created to assist with the process. Orchestration refers to the automation of key parts of the management of containerised systems(8). These are:

- Deploying new containers depending on configuration files. Containers do not have to be manually started but can instead be automatically started from a script. Multiple interdependent containers can be launched simultaneously in this way.
- Coordinating deployed containers. Containers can be grouped into logical units based on whether they require each others services or not.

- Network provisioning for containers, enabling them to access and be accessed via a network. This can include access over the internet.
- Flexible Scaling of resources to meet demand. New groups of containers can be created if the system load is exceeding a certain threshold. This can help prevent slow down if a large number of users attempt to access a containerised system.
- Availability and fault tolerance, achieved by adding some form of redundancy to the system. One container going down should not bring the whole system down.
- Stopping container instances when they are no longer needed. This category includes restarting containers which have entered an erroneous or corrupted state.

### 2.3.1 Kubernetes

Kubernetes is a popular, open source container orchestration provider. It was originally designed by Google engineers to manage the automation of container deployments. It can be run on a physical machine, cloud infrastructure or a hybrid scheme. In Kubernetes, the smallest deployable instance is called a pod, a single application instance. Kubernetes can handle systems consisting of many multiples of pods(9).

#### Kubernetes Design

Kubernetes design can be divided into the control plane and the nodes. The control plane contains the services which manage the nodes and the pods within the nodes. These are the cluster data store, etcd, the schedular which assigns pods to nodes that are free, kube-scheduler, the control management services kube-controller and cloud-controller manager, and kube-api-server which acts as a gateway to the control plane. This is shown in Figure 2.3[1].

Nodes can be run on physical machines or on virtual machines. They handle and control the creation and management of pods. They consist of the kube-proxy, which enables external network connections to pods in the node and kubelet which manages the containers running on the node.

#### Pods

Pods represent the basic executable instance of an application in Kubernetes(10). They are deployed onto the worker nodes. A pod consists of one or more containers, their underlying storage and an IP address. All containers in the same pod are assigned the same IP address, and can communicate with each other through their shared data storage or via localhost.

[1]Source: Kubernetes online documentation `kubernetes.io/docs/concepts/overview/components/`

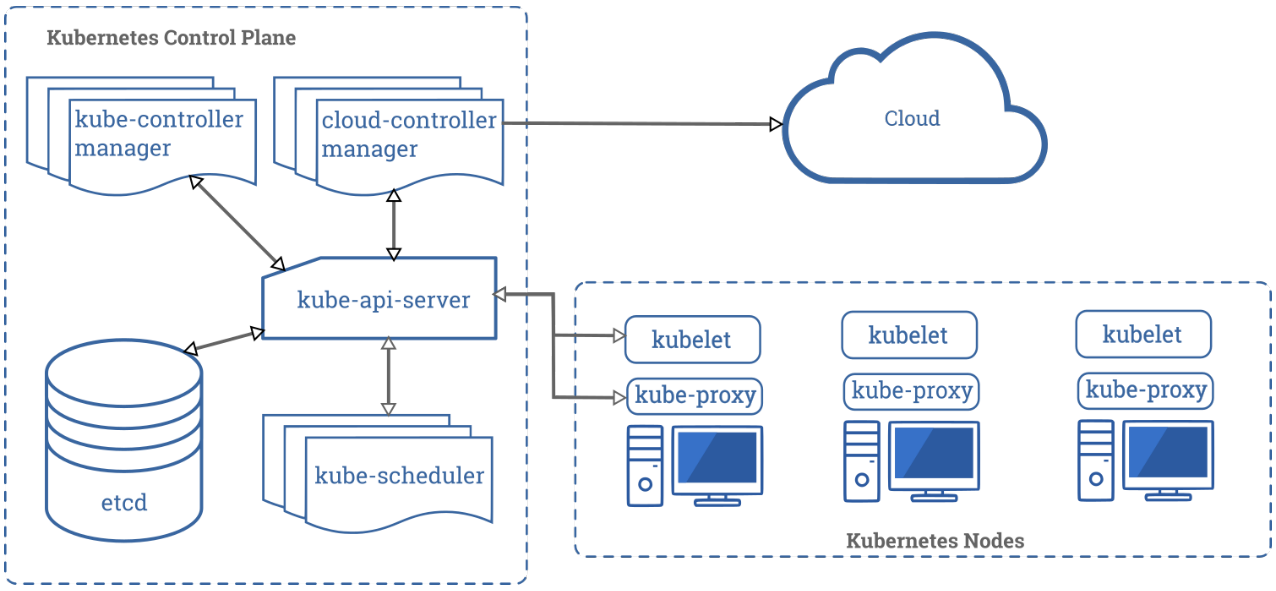


Figure 2.3: Kubernetes Components

### Cloud Integration

Kubernetes management and control planes are often directly supported by cloud service providers. Examples include Amazon Elastic Kubernetes Service and Google Kubernetes Engine.

## 2.4 Cloud Computing

Cloud computing describes the practise of utilising computing resources that you do not control to handle data storage, calculations, hosting or any other computing service(11). This is opposed to the traditional computing scenario where a company would manage hardware that they control to host services and servers, store data and compute calculations.

A major draw of cloud computing is the fact that it helps enable many modern IT business paradigms. These include many of the "X as a Service" concepts, such as Software as a Service and Platform as a Service. Another advantage is the removal of capital expenditure on machine hardware, which is replaced by a usage fee generally based on how many resources you are making use of.

Cloud service providers give their clients access to computing resources they control in return for this usage fee. For example, AWS give their clients access to a virtual machine instance which is hosted on Amazon infrastructure in a data centre. The physical locations of the infrastructure can be selected in order to lower latency for the end user. Many providers allow for the resources used by the virtual machine instances to be scaled up or down as the user demands.

A major potential negative of cloud computing is the fact that the client's data is now not fully under their control. If they lose internet access or if the cloud service provider is taken down for one reason or another, their resources are now inaccessible.

## 2.5 X as a Service

Offering to host computing resources externally has led to companies offering to host more and services that were traditionally run by each client on site. This has led to the term X as a Service, also called a service oriented architecture. This architecture abstracts away some of the complications inherent in accessing, running and hosting applications, which have limited the portability of software in the past. The new model also introduces new ways for companies to keep monetising software and hardware, replacing one time purchase costs with some variation of a subscription model.

The shift to externally provided services threatens to overturn the current security landscape, ameliorating some vulnerabilities while exacerbating others and introducing new ones.

### 2.5.1 Software as a Service

Software as a Service is a service oriented model in which applications are run by companies in the cloud, and the end users of those applications connect to the cloud environment to make use of their services. The end user has no control over the environment that the software runs on.

Because the application is not run on a client's machine, portability is increased as the operating system or hardware in use by the client should no longer matter. Additionally, technical issues which arise should be easier to solve, and should not require a technician to make an on site visit to the client. Maintaining and updating the application is made simpler too, as the developers can ensure that everyone is presented the latest version of the software at all times.

The end user may download a very simple client program, whose sole purpose is to connect to the application running in the cloud, or else connect via their web browser. The end user generally pays either on demand, or else pays a subscription fee to maintain access to the software.

### 2.5.2 Platform as a Service

Platform as a Service is the paradigm by which cloud providers allow a client to build and run their own applications in the cloud, by providing them with the underlying environment necessary to maintain such applications.

The main concept of PaaS is that the developer does not need to manage the underlying resources used by their system. This includes the OS, network environment and drivers.

Platform as a Service enables the client to build their applications without worrying about any of the underlying infrastructure. The cloud service provider supplies them with the (usually virtualised) underlying server infrastructure, databases and the necessary libraries to run programming language environments. This collection of offerings is often referred to as a computing platform.

### 2.5.3 Infrastructure as a Service

Infrastructure as a service is where the cloud service provider gives a client the ability to setup their own platform using the cloud infrastructure. They are renting the cloud infrastructure in place of maintaining their own on site.

Effectively, IaaS enables the client to set up their cloud deployment in the same manner in which they would set up a deployment on their own infrastructure. They have control over the operating system, databases, some networking options and the applications that will run in this environment.

This is one architecture used by cloud service providers to allow companies and developers to build their applications in the cloud. Their clients are given the ability to launch their virtual machine images on a hypervisor at the service provider's data centre. The developers in this example are paying to use the cloud infrastructure rather than maintaining their own hardware.

## 2.6 Threat Landscape

Attacks against internet facing computing infrastructure continue to be a serious threat in 2020. This is a threat both to the security of such systems, and to the privacy of their users.

The concept of computing security was not taken too seriously in the early days of the internet. However, a turning point for came in November 1988, when the infamous Morris worm was unleashed onto an unsuspecting internet. Created by a graduate student, Robert Morris, The worm spread quickly by exploiting vulnerabilities in sendmail, finger, Unix and BSD which had been known of beforehand, along with weak passwords. The consequences of the worm were quite severe, many machines were taken down and the internet was partitioned for a number of days while the worm was removed from infected systems(12).

While the possibility of a system being attacked is generally accepted today, such attacks remain commonplace. Despite the increasing number of online personal computers, IoT devices, smart devices and cloud based services, much of the general public remain unaware of the inner workings of such systems. This is particularly true for security related concepts such as cryptography. Many software engineers are also somewhat ignorant of the security implications of design choices they make, and development practises that they perform. The primary measure of whether an attack is successful or not today remains not a question of the skill of the attackers, but rather the naïvety of developers, users and system administrators.

### 2.6.1 Attackers

The modern world contains an ever increasing number of computer systems. These systems are of interest to many people looking to illicitly modify, break into, or simply break them in a number of ways. Any such person can be generally referred to as an attacker. Commonly in this space, malicious actors are referred to as black hats, while security professionals who probe systems for vulnerabilities with official permission are referred to as white hats. To the average citizen, black hats are often simply known as "hackers". There also exists the concept of a grey hat, someone who attacks a system without permission or engages in illegal activity, but without a malicious goal.

In addition to the professional or more sophisticated attackers, there are many who lack the necessary knowledge or skills to craft attacks of their own, but instead utilise tools or scripts created by others. Generally this group lacks even the technical knowledge of the attacks that they are carrying out. These attackers are often called by the pejorative term "script kiddies", and are perceived as a nuisance looking to show off to like minded individuals.

Attacks themselves can be sub-divided into those that are targeted at a specific individuals, companies or states, and those which are indiscriminate. Which style of attack is used often depends on the specific goals of the attacker.

The primary goal of a malicious actor can be divided into the following categories.

#### Financial Gain

The general goal of attackers engaging in criminality is their own financial gain. As such, common targets for such attacks are victim's credit card details and victim's bank details. These can either be directly stolen, or indirectly stolen.

Attackers can directly obtain these details through methods such as phishing emails, or by inserting credit card skimmers into vulnerable online shops. An example of this was the 2020 Tupperware credit card skimming incident. Indirect access to victim's financial data can be done via obtaining login details from a different site which the victim also uses for other important accounts, such as their primary email, banking or online shopping accounts. In this case, the malicious actor may steal these credentials from a site themselves, obtain them from a password dump or buy them from a different attacker.

Some attackers sell their services to others for profit. A common example of this is the renting out of botnets for the purpose of launching a DDoS attack at a target of their client's choosing.

### State or Corporate Espionage

State agencies or state sponsored actors represent some of the most advanced threats to any computer system. These groups often have access to lists of vulnerabilities which have not been publicly disclosed. In some instances, it is alleged that states have forced manufacturers to install backdoors into their systems, a recent example being the allegations that SuperMicro products contained Chinese backdoors. Concerns about the potential for governments to use backdoors to spy on other nation's citizens and businesses have increased in recent years. An example is the controversy over the decision to use Huawei's 5G infrastructure in the United Kingdom.

The goals of state actors are generally political or economic. They attempt to sabotage rival nations strategic infrastructure, or to disrupt their economy by targeting important private enterprises. They can also attempt to manipulate public opinion and discourse in other nations. The theft of intellectual property is another goal of these groups. The increasing threat of so called "cyber warfare" has led to the creation of state bodies dedicated to handling the role, such as USCYBERCOM in the United States.

Stuxnet was an example of malware used to target a state's strategic infrastructure(13). It targeted programmable logic controllers, particularly those manufactured by Siemens that were used by Iran in their uranium enrichment centrifuges. The malware caused some centrifuges to spin too fast and disintegrate. Stuxnet infected its targets via an infected usb stick, enabling it to attack air-gapped systems.

Many nations have been linked to such activities, including the United States, Russian Federation, China, Israel and Iran.

### Pettiness or Revenge

An increasing number of DDoS attacks are being linked to competitive online games(14), where a lost game results in a player losing some amount of ranking score. So called DDoS for hire services are used by some unscrupulous players to guarantee themselves a victory by preventing the opposing team from accessing the game servers. They can also prevent a defeat by causing enough network disruption that a game is abandoned. In these cases, the attacker is generally tech illiterate, and acquires the services of someone else to carry out the attack.

Revenge is another goal of some attackers. A disgruntled current or former employee can be particularly threatening to a company, as they can have insider access to company systems and an understanding of the security mechanisms currently in place.

#### Activism

Some attackers are motivated by a desire to either champion a cause, or to detract from a movement or group they disagree with. This can range from targeting businesses they perceive as immoral, to targeting terrorist organisations.

These "Hacktivists" can act either alone, as part of a group, or as part of a decentralised collective with a common goal, but no command structure.

### 2.6.2 Port Scanners

A port scanner is a piece of software which is used to discover which ports are open or closed on a target system. They often have additional features such as determining what Operating System is in use at the target, or what specific service or version of an application is accessible at a port. Port scanners are legal pieces of software with legitimate uses, however they are also used by attackers to map out networks and look for potential weaknesses in the form of an open port.

#### Nmap

A popular example of a port scanner is nmap. Nmap is a piece of software which can map out networks, including open, closed and filtered ports, services available and firewall detection. Port scanning is just one portion of the functionality offered by nmap.

#### TCP Stealth SYN scan

The default scanning option used by nmap is a TCP SYN scan. This exploits the three way TCP handshake in order to rapidly test the ports at a host without establishing an actual TCP connection. The host's response to a TCP packet with the SYN bit set is used to classify the port into the open state, filtered state or closed state(15).

The scan works by sending the first part of the handshake, a TCP SYN packet, to the target port. If the port is closed, the target will respond by sending a TCP RST packet, closing the connection. This process is shown in Figure 2.4.

If the target port is open, the host will respond to the SYN packet with the second part of the handshake, a TCP packet with the SYN and ACK flags set. Upon receiving this packet,

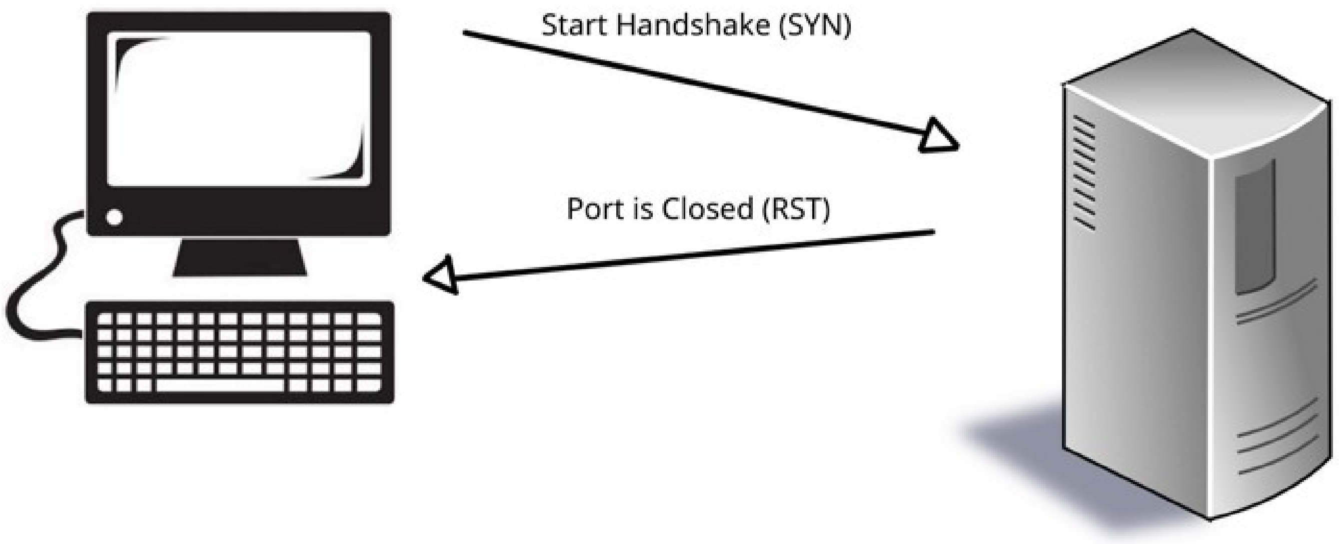


Figure 2.4: TCP SYN scan when target port is closed

we have confirmed that the port is open. To prevent establishing a connection, we respond with a RST packet.This is shown in Figure 2.5. Nmap does not have to send this packet as the OS will automatically do so upon receiving the SYN/ACK. This occurs because nmap handled the initial SYN packet itself, so the OS was not aware of any attempt at establishing a connection nor was it expecting any response. Upon receiving any SYN/ACK response it was not expecting, the OS will respond with RST.

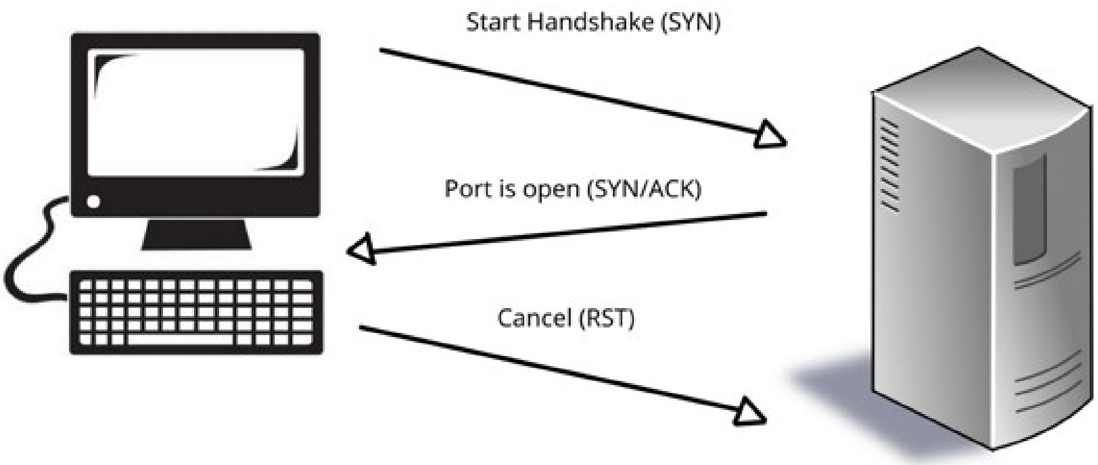


Figure 2.5: TCP SYN scan when the target port is open

### 2.6.3 Attack Vectors

Attackers can achieve their goals through attacking specific vectors. Some of the vectors they can use are discussed below.

### Network Attacks

DDoS attacks, which overwhelm endpoints with massive amounts of traffic, are an example of this form. The standard DDoS attack involves using infected "zombie" machines to send large numbers of packets to a web server, making the target website unresponsive. Such was the rise in prominence of DDoS attacks over the late 2000s and 2010s, that many companies now offer services for mitigating such attacks to websites and other web services.

There are also attacks which involve passive monitoring of network traffic. However, with the rise of HTTPS and DNS over HTTPS, the potential damage caused by passive packet capture is much reduced. That said, it still remains a useful tool for an attacker.

### Targeting Individuals

An attacker can gain access to a system or sensitive data by targeting an individual user or employee. They then can attempt to manipulate their victim into giving up passwords or into allowing the attacker to bypass security protocols at a company.

One way to achieve this is through social engineering. This category includes phishing emails, in which the attacker poses as a trusted entity such as the victim's bank or a company's internal HR department and sends emails requesting sensitive data. This can be passwords and bank details, or even sensitive corporate information such as company secrets.

Social engineering can also be done in person by an attacker. Many pen testing companies gain access to a target company's systems through posing as an IT contractor and simply walking in the front door and asking for access to company machines or server rooms. This type of attack exploits the victim's social expectations, A man or woman in a high visibility jacket holding a clip board is often trusted to be a legitimate worker. Once an attacker has access to a company's internal systems, they can install monitoring devices or attempt to gain maximum privileges and extract sensitive data.

### Targeting System Vulnerabilities

Attackers can exploit oversights in the implementation of a system, such as software bugs, in order to carry out their attack.

Many software vulnerabilities can be grouped into similar classes. As certain bugs are very common, an attacker can easily attempt to break the system by iterating over the most common ones. Examples of common system vulnerabilities are buffer overflows, a lack of input sanitation[2] and stack or heap overflows.

[2]This class of vulnerabilities enables many common attacks, such as SQL injection, format string attacks and some Cross Site Scripting attacks

Once an attacker has found a software flaw to exploit, they can use it to execute privileged actions which permissions would usually prevent them from doing. Typical goals are to escalate their privilege in a system via Return Oriented Programming, or to get the system to run a desired payload on other user's machines such as in a Cross Site Scripting attack.

#### Side Channel Attacks

More sophisticated attackers are able to exploit the design of computer systems themselves. This class of attack is known as a side channel attack. Examples of side channels which can be exploited are variations in power usage when computing or decrypting data, measuring the time taken to run specific operations or exploiting the way in which CPU caching is implemented to leak sensitive data.

The Spectre (CVE-2017-5753) vulnerability in Intel, ARM and AMD CPUs and disclosed in 2018 is an example of a vulnerability that enables side channel attacks. Spectre refers to a class of side channel vulnerabilities which exploit branch prediction and out of order execution in modern CPU designs to discover the contents of cached memory via timing attacks(16).

The related Meltdown (CVE-2017-5754) vulnerability, affecting Intel and some ARM CPUs is another example. Meltdown enables unprivileged code to read memory locations it lacks permissions for by exploiting a race condition during memory access. It is possible to access this memory thanks to exploiting the way speculative execution works to load sensitive data into the CPU cache where it can then be accessed(17).

As side channel attacks target the underlying computer implementation, a software developer working on a critical application can do very little to prevent such attacks.

### 2.6.4 Malware

Malware is the general term used to describe any form of malicious software. This includes categories such as computer viruses and worms. However, today's malware landscape is particularly dominated by a certain type of malware, Trojans.

#### Trojan Horse

A Trojan Horse, often shortened to Trojan, is a category of malware which masquerades as a piece of legitimate software. This can be an email attachment, PDF or game executable file among many other deceptions. The Trojan strategy is often the mechanism by which malware creators get their code to run on their victim's machines, making it an extremely common way of propagating malware in the modern world.

### Ransomware

Ransomware is a sub-type of malware which has grown in prominence in the past few years due to several high profile incidents. Ransomware is now a global security threat.

Ransomware scams follow a general pattern, though there are some variations in behaviour. A user downloads and runs a Trojan disguised as a legitimate application. Upon executing the Trojan, the Ransomware begins to encrypt the user's files using a generated key. Once this is done, the user will be informed of the encryption and asked to pay in cryptocurrency to have their files decrypted. It is not guarenteed that the user will receive the key or be able to decrypt their files even if they pay the ransom.

Some of the high profile ransomware incidents were CryptoLocker in 2013, Petya and the derivitive NotPetya in 2016 and 2017 and WannaCry in 2017.

- CryptoLocker spread via a Trojan disguised as an email attachment. Upon encrypting the victim's files, the ransomware demanded payment before a certain deadline, at which point it claimed decryption would be impossible. CryptoLocker saw such success that it has spawned several clones, including unrelated malware which have adopted the CryptoLocker name.
- Petya differs from conventional ransomware in that it overwrites the Master Boot Record of the victim's hard drive. This means that on the next boot up of the infected machine, Petya's payload would be run instead of the Windows bootloader. It would then encrypt a key part of Window's NTFS file system (the Master File Table) before demanding the ransom payment. Though the file system has been compromised, individual files are not encrypted, thus forensic and security experts are able to retrieve individual files from the hard drive at great effort.
- NotPetya surfaced in 2017 as a derivative of Petya, which was modified to utilise the leaked, NSA developed EternalBlue exploit(18). NotPetya is notable as it was impossible to recover files at all due to design changes made to the original Petya. Is is alleged that NotPetya was created to disrupt the Ukrainian economy, possibly by Russian agents, rather than as a true ransomware attack.
- WannaCry gained infamy when it struck the United Kingdom's National Health Service, leading to a disruption in the normal operation of some hospitals(19). It too utilised the EternalBlue exploit to infect Windows computers(1). The attack was halted when a security researcher discovered a "kill switch" which prevented further spread of the ransomware.

In addition to financial scams, there are also examples of ransomware created as a practical joke. One such example is Rensenware(20), which encrypts the victim's files like any other ransomware. Where it differs from the standard ransomware attack is that it does not ask for money to decrypt the files, but rather prompts the victim to obtain over 200,000,000 points in the Japanese indie game Touhou to recover their data. The creator later apologised and released a fix which would decrypt the encrypted files by cheating in the game and setting the player's score to the required value.

### 2.6.5 Botnets

A botnet is a collection of infected internet connected devices, which can be controlled by the botnet's master to perform illicit tasks on their behalf.

Many botnets used the Internet Relay Chat protocol to send and receive command and control messages, due to its relative simplicity and low bandwidth use. There has been a move to peer to peer controlled botnets in recent years due to the ease by which IRC communications can be blocked.

Botnets are commonly used to launch DDoS attacks against a target. This is done by having the infected machines send large numbers of malicious packets at the endpoint. This can overwhelm the server at the target, by tying up processing time handling all of the requests, or by using up memory by forcing the server to create buffers for each incoming packet stream. The severity of the attack depends on the size of the botnet. The amount of data being pushed to a victim during a DDoS attack can exceed 100s of Gb/s.

Other actions that botnets are used to carry out are propagating spam emails and mining cryptocurrency for the owner's gain at the victim's expense.

#### Zombie Machines

An infected machine in a botnet is often referred to as a zombie, or a bot. How a machine is infected and added to the botnet depends on the botnet in question. A victim can download and execute a Trojan which infects their machine, or an automated probe can attempt common username and password combinations in order to gain access to a device and install their payload.

Zombie machines can have backdoors installed onto them which enable the botnet controller to access the infected device when they please, stealing any useful or sensitive data.

#### IoT botnets

One type of botnet design that has grown enormously lately are Internet of Things Botnets. The 2016 Mirai botnet remains the most famous of these, although many others exist such as Hajime, ECHOBOT and PureMasuta.

### 2.6.6 Example: Mirai

Mirai rose to prominence when it was used to carry out several high profile DDoS attacks. This included a 620 Gb/s attack against Krebs on Security, the personal site of notable security personality and journalist, Brian Krebs. More significantly, Mirai was used to launch a DDoS attack against the DNS provider Dyn. The attack left many popular websites such as Reddit, Github and Netflix unreachable in late October 2016.

Mirai works by scanning the internet address space for vulnerable IoT devices, targeting the Telnet TCP ports 23 and 2323. It is hard coded to ignore certain address ranges, including those used by the US Department of Defense. Once it has identified a potential target, Mirai launches a dictionary attack on the login credentials, using a dictionary containing the default username and password combinations for many IoT devices. The worm relies on its victims not having updated these default credentials, a massive security flaw. At this point, the bot payload has been downloaded and installed to the infected device, evidence steathily cleaned up, and the bot is now listening for commands from the botnet master. The infected device will also simultaneously search for other vulnerable IoT devices to add to the botnet(21).

Interestingly, Mirai attempts to block other botnets or worms from utilising an infected device. It does this by blocking SSH and Telnet traffic from the target, and by attempting to identify other malware on in the device's memory and remove it from the system. Targeted examples include the qbot and .anime botnets.

Mirai's command and control code is written in the Go language, while the worm that infects devices is written in C.

## 2.7 Threats to Cloud Computing

All systems have vulnerabilities, cloud based deployments are no different. A cloud deployment retains the same vulnerabilities that any internet connected service has. Weak user credentials, vulnerability to DDoS attacks and vulnerable open ports are just some examples of these.

Litchfield and Shahzad discuss some of the potential security risks associated with cloud computing(22). They pay special attention to potential side channel attacks in which an attacker with access to a virtual machine cloud instance can gain access to private keys belonging to other VMs which are running on the same hardware in the cloud datacentre. This type of attack is described in more detail in the Side Channel subsection above.

Christodorescu et al. note that vulnerabilities in cloud computing do not only relate to the virtualised environment(23). They note that the VM images themselves can be compromised at or before launch and that one cannot assume that the system is clean.

### 2.7.1 Recent Example: CVE-2019-5736

The exploit CVE-2019-5736 was disclosed in February 2019. It exploits a flaw with runc to escape any container launched using runc, which included any default Docker setup. Runc is a Linux runtime which is used to launch programs, including containers(24).

The exploit works as follows. A malicious user can replace the runc binary on the host from any container. This can be achieved by getting runc to launch an instance of itself within the controlled container. In this environment, the attacker can modify environment variables so that this instance of runc will use local copies of software libraries that it depends on. This is where the attacker can insert their malicious code.

The spawned runc instance will still be owned by the root user. This enables overwriting the original runc with this local copy while maintaining the elevated permissions. Now when the host runs runc again, the attacker's code is run instead. This malicious code is being executed as root in the host's context. At this point, the attacker has root access to the host.

In a cloud environment, this vulnerability would enable a malicious user with access to a container to gain access to other elements of the cloud environment, posing a massive risk to the entire system. AWS has reported that the vulnerability had been patched in the same blog post where they announced it to the public.

## 2.8 Intrusion Detection

Intrusion detection mechanisms provide a system administrator with a number of advantages.

- Alarms can be raised when malicious activity is detected. System administrators can then mive to backup key systems or take them offline to prevent data loss or vandalism.
- The ability to distract an attacker from legitimate targets using honeypots, and to otherwise waste the attacker's time.
- The ability to track an attacker's movements within the system and to log their actions. This can provide critical data on what parts of the system have been compromised, and what security measures have succeeded or failed

### 2.8.1 Honeypots

A honeypot is one form of countermeasure to an attack on a system. It is the name given to a system which is deliberately designed to be targeted by a malicious actor.

Honeypots vary in design depending on the type of attack the developer wants to mitigate, and the goals of the person deploying them. If the goal is to trap an intruder so that they are slowed down and their actions monitored, a complicated design which more closely emulates a real system and encourages a lot of interaction from the attacker is used. If the goal is simply to bait login attempts so that they can be logged, a simple design can suffice.

Honeypots have an additional role as a research tool. They can be used by security researchers to analyse ongoing attacks, log typical attacker behaviours within a compromised system, log an attacker's keystrokes and to detect when a new security threat arises.

#### Advantages

Baiting an attacker into targeting a honeypot offers many advantages. Firstly, the honeypot can distract an attacker from a more vulnerable system that we want to protect. This can enable the vulnerable system to be taken down or backed up before it is compromised. Secondly, the honeypot can be designed to slow an attacker down. This can be achieved by having a very large data volume for an attacker exploit. Thirdly, an attackers methods can be logged and used as future reference when designing a secure system.

#### Challenges

A major challenge to the design of honeypots is ensuring that the honeypot seems to be a legitimate system from the perspective of an attacker. This challenge is further compounded by the fact that what appears legitimate to a human attacker manually launching an attack is not necessarily the same as what a piece of malware would consider legitimate. Human attackers may place more value on human readable filenames and evidence that the system is legitimately being used, such as new files with actual data being created. Automated attackers may not care about filenames or evidence of legitimate use to the same extent as a human, but may be able to more rapidly compare the target environment to known honeypot environments to determine that the target is indeed a honeypot.

Zou and Cunningham considered the design of a botnet which could automatically detect if an infected system was a honeypot or not by attempting to send malicious packets from an infected target to an endpoint the botnet owner controls(25). If this is not possible, or if attack commands are not being followed, the botnet can stealthily clean itself up from a system.

The authors further investigate this idea by proposing a two stage infection process, whereby the target is first infected by a "spearhead". The attack will only continue, and the real payload will only be deployed, if the spearhead can confirm that the target is not a honeypot. This method exploits the fact that many individuals or companies deploying honeypots would rather not have their resources used as part of a real world attack.

### 2.8.2 Honeynets

Honeynets build upon the concept of a honeypot to provide more detailed monitoring of an ongoing attack against a system, while also providing a more enticing target for an attacker. A honeynet will usually consist of a network of high interaction honeypots, deployed in a realistic network configuration. Typically, there are multiple layers of data collection in this design. One can deploy honeypots simulating a more complex network environment, with fake SQL databases, fake webservers and fake worker nodes. This means that a wider variety of attacks can be baited into attacking the honeynet.

Spitzner in The Honeynet Project: Trapping the Hackers describes a basic honeynet deployment, dubbed "Gen I". This architecture describes a simple, contained environment. This design mostly suits research purposes only, baiting attackers to target the honeynet to study the anatomy of their attacks. It consists of a network containing almost entirely honeypots, along with an Intrusion Detection System such as Snort, and some form of logging server. For this reason it is not very suitable for securing a production network.

The architecture is described in Figure 2.6[3]. The honeypots are highlighted in yellow.

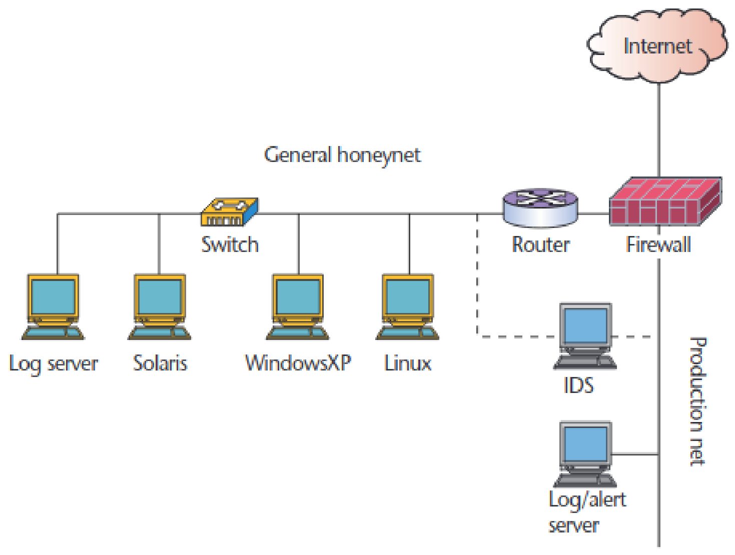


Figure 2.6: Gen I Honeynet Design

The author goes on to describe a more sophisticated honeynet design. Dubbed a "Gen II"

[3]Sourced from The Honeynet Project: Trapping the Hackers

architecture, this design allows malicious traffic in a network to be isolated in an internal honeynet. Another key feature of this design is the addition of a layer 2 bridge, named the Honeynet sensor. This bridge allows incoming packets from an attacker to flow freely, but limits outgoing traffic to hinder an attacker's ability to further propagate attacks. The Honeynet sensor also enables packet capture and can log an attacker's key strokes.

The architecture is described in Figure 2.7[4]. The yellow machines are the honeypots, while production machines are coloured grey. The Honeynet sensor is the bridge which seamlessly filters malicious traffic such that it can slow outbound attacks.

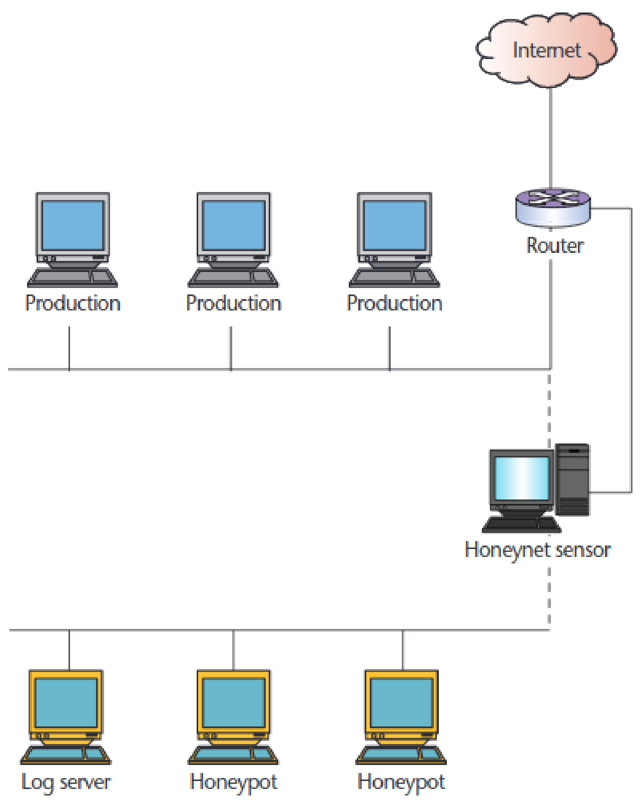


Figure 2.7: Gen II Honeynet Design

### 2.8.3 Signature Based Intrusion Detection

Signature based intrusion detection relies on comparing network traffic and activity within the system to a database of known malicious or suspicious activity. The concept is similar to the approach of the same name used in commercial anti virus software.

A downside of this strategy is that new attacks can only be detected if they are derivatives of attacks that are already known. An adaptive attacker can also disguise their activities and remain undetected by a purely signature based approach.

[4]Sourced from The Honeynet Project: Trapping the Hackers

### 2.8.4 Anomaly Based Intrusion Detection

In contrast to signature based methods, anomaly based intrusion detection does not have to have a record of a specific attack beforehand in order to detect it. Instead, it relies on heuristics to determine whether the current state of network traffic differs from the norm in a suspicious way.

Anomaly based approaches may involve a period of training in which the system learns what typical traffic in the system looks like, potentially using some form of machine learning model.

## 2.9 Related Work

The concept of securing cloud deployments is less explored in academia than its traditional deployment counterpart. Much of the research that does exist is aimed at preventing intrusion from external sources to the deployment, or on methods that cloud service providers can use to detect suspicious activity on their infrastructure. Less work has been done on the cloud service user securing their deployment once an intruder has already gained access.

Graham, Winckles and Sanchez-Velazquez(26) investigated detecting botnets within cloud infrastructure using flow protocols. They propose using Netflow, a protocol used to monitor and analyse network traffic, to detect the command and control communications of a botnet within a cloud environment. This research is intended for use by the cloud service providers, rather than the end users.

Christodorescu et al(23). describe a novel method of securing virtual machine instances running in a cloud environment. The process involves first determining what guest operating system is in use by analysing the Interrupt Descriptor Table. Once this has been established, they can verify that key OS features have not been tampered with by verifying the integrity of key OS data structures. This design aimed at determining if a virtual machine has been infected by malware, it offers little protection against unauthorised access to the cloud environment.

Containerised honeypots are a less explored topic in research. Much of the existing literature focuses on the application of containerised or cloud based honeypots in the research field, with the goal of studying attacks rather than securing systems.

One such example is the work done by Kyriakou and Sklavos(27). They deployed multiple Docker containers using honeypot images of Cowrie, Glastopf and Dionaea to the cloud. Each honeypot was simulating a different service. They then monitored these honeypots in real time to gather useful data on malicious activity.

Containerised versions of commonly used honeypots have been created, and can be deployed into a cloud environment. This includes Docker-Cowrie, a containerised version of the popular Cowrie honeypot.

# 3 Problem Formulation

The objectives of this research are motivated by the modern threat landscape as discussed in the previous chapter. The simultaneous push for cloud based solutions while online criminality increases is a massive cause for concern. Additionally, many deployments are moving to containerised approaches, which have some advantages, but are arguably less secure than a corresponding virtual machine. More work needs to be done to shore up the defences of such systems to ensure the protection of cloud deployments in the future.

## 3.1 Challenges

The security of cloud deployments faces challenges on multiple fronts. Tackling at least part of these challenges is the goal of this research. Some of the most relevant identified problems from the background research are enumerated below.

(1) It is not possible to rely on default settings alone to maintain a secure cloud deployment. Some form of active defence infrastructure is required to halt or slow down ongoing attacks.

(2) While cloud service providers help secure deployments from the outside, and provide developers the tools to securely link their deployment to the internet, a base cloud deployment contains no tools that allow for intrusion detection.

This means that a malicious actor who gains access to SSH keys uploaded to Github, stored on a lost USB drive or a computer than is currently logged into a cloud environment, can infiltrate a deployment undetected. They can then easily exfiltrate sensitive data, or compromise the whole system and install some form of malware, without much risk of detection.

(3) A lack of intrusion detection mechanisms means that a system can be compromised, and used for unauthorised purposes such as cryptocurrency mining or as part of a botnet. A user inexperienced with IT systems may not notice any difference other than a slowdown in normal tasks.

By the time anomalous behaviour is noticed by someone, it is too late to react. Some form of simple to understand alert system could help to mitigate this problem, and enable even those who are somewhat less experienced with the topic area to notice an issue and react accordingly.

(4) The proliferation of platform as a service and infrastructure as a service architectures means that a compromised container and virtual machine can lead to an attacker accessing confidential information from other user's VMs and containers running on the same host. A side channel attack is one method to achieve this aim.

## 3.2 Proposal

The main thrust of this research is to introduce some form of active defence to a cloud environment, to mitigate the challenges identified above. The solution should be able to bait attacks away from key infrastructure and data, while also alerting the system administrators that something suspicious is taking place.

While reviewing the literature, it was noted that most research had left out the concept of a malicious actor bypassing external security protocols by gaining direct access to login credentials or a company's hardware. As such, such a scenario became a focal point for investigation.

The initial concept of this research was to propose a new, containerised adaptive honeypot design, based on the open source Cowrie honeypot. This would enable better interactivity than a standard honeypot deployment, and therefore slow down an attack more while gathering higher quality data. The adaptive nature of the honeypot would also make it harder to detect by a malicious actor. The solution would provide an attack defence mechanism inside the deployment, which would be able to detect both external threats that had somehow made it inside, and internal threats of the nature discussed above.

Further review revealed that there was some existing work done in this area. As such, it was decided to focus on a different, but related topic. This was securing cloud based Kubernetes deployments via some form of active defence. Despite the growth of Kubernetes cloud deployments, there has not been as much research done into actively securing such systems. This applies doubly so for internal security of a deployment.

The objectives of the solution are as follows.

- The design should both detect an ongoing attack, and distract the attacker from sensitive user data by baiting them to a seemingly more lucrative target.
- The system should alert administrators when a breach has occurred, preferably in real time via an alarm system.

- It should log when a security incident occurs, so that administrators can review the logs and adjust their policies based on the outcomes.
- As many parts of the implementation as possible should be kept low key in order to avoid detection by the attacker.
- The solution should be easily deployable in a cloud environment and cost effective. An overly cumbersome implementation could lead to confusion during deployment. A very costly one could lead a company to decide that it would rather gamble with the risk of a data breach, than pay a significant sum of money.

# 4 Design

The design of a solution to this problem is one of the most critical steps in developing a working solution. A mistake here may only surface during the final evaluation of the proposal. As such, a lot of effort is devoted to creating a functional design. Additionally, the design stage is where possible security flaws and ethical issues may arise and should be solved.

## 4.1 Goals

In order to help meet the objectives identified in Chapter 3, they were refined into specific design goals.

- The system should be containerised
- The system should detect any ongoing attacks from insiders
- The system should slow an attacker to buy time, and distract them from system containers to protect data
- The system should be simple enough that anyone capable of deploying applications in the cloud is able to deploy it

## 4.2 Proposed Design

This research proposes a solution that builds upon the existing practices used in traditional enterprise deployments and extrapolates it to a container based cloud deployment utilising Kubernetes. Theoretically, the proposed design can also be used in bare cloud deployments that do not utilise Kubernetes as well.

The solution consists of two containerised components. The first is a containerised version of the Cowrie Honeypot. Cowrie was chosen because it is open source, already containerised, has high interactivity and blocks attempted downloads. In theory, any decently interactive honeypot can be chosen for this role. In a production environment, the chosen honeypot should depend on what systems the user or company is trying to protect.

The second portion of the design is the port scanner listener. This is a Ruby script which listens for packets that are associated with port scanning. The reason for this is that a port scan is a common tool used by attackers to build a map of potential targets within a system, and there is usually no legitimate reason for a port scan to occur within a network without the system administrator's knowledge. The example implementation is designed to detect the common TCP stealth SYN scan, though it can be extended to detect more varieties of scan too. The details of a TCP SYN scan are discussed in more detail in Chapter 2.

Figure 4.1 shows an abstracted view of how the system works. The containerised environment is already protected from external malicious traffic by the cloud service provider and the configurable firewall in the deployment. Inside the deployment, the internal intrusion detection system provides a way to identify suspicious activity. The honeypot can be actively connected, just like a production container, so that the attackers can be baited to it. The port scanner listener only passively interacts with the network. TCP packets can be sent to it, but it will not maintain a TCP session.

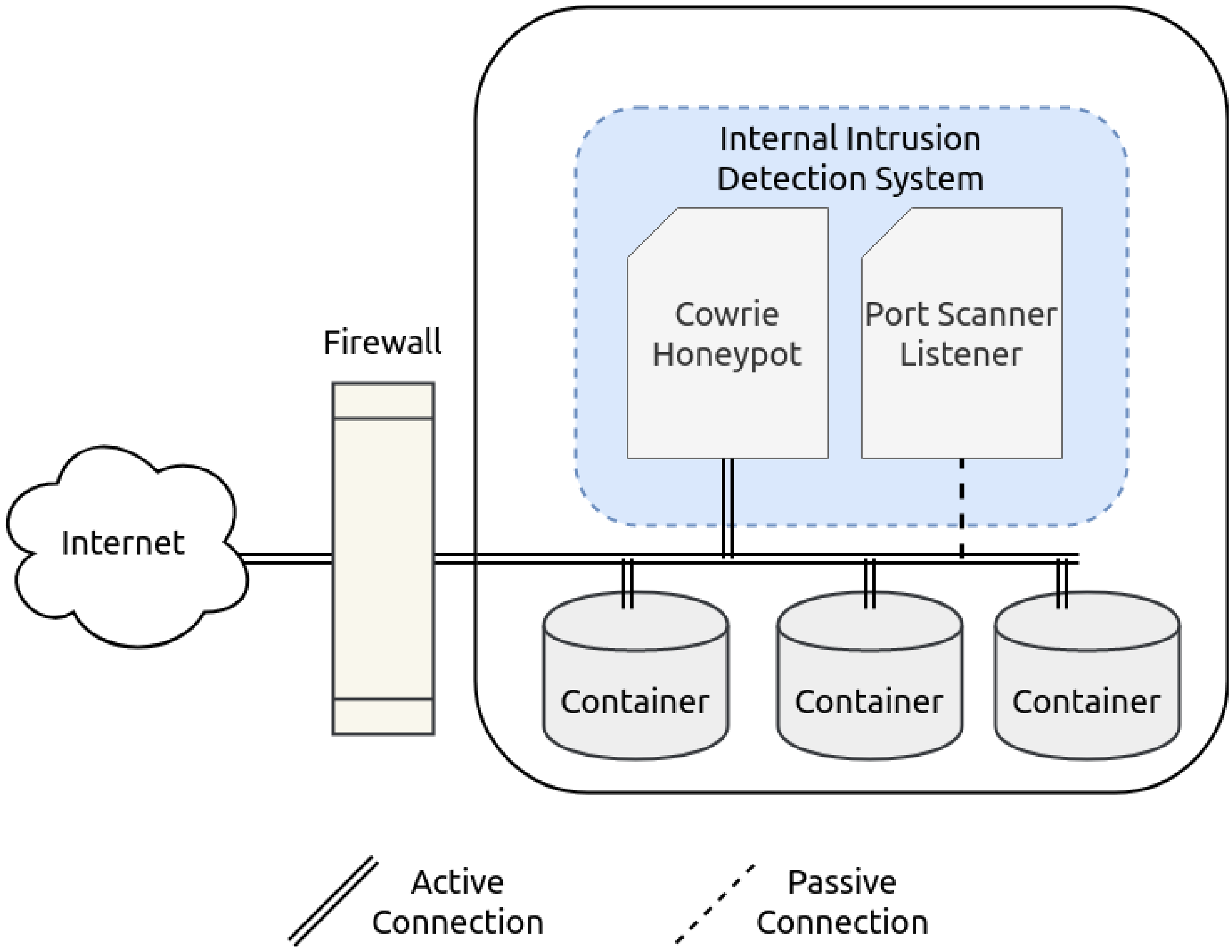


Figure 4.1: Generic architecture of the proposed design

Figure 4.2 demonstrates how the proposed system is laid out in a Kubernetes deployment in the cloud. Each worker node that contains a pod will have at least one instance of the internal intrusion detection system, as each pod will have one. Within the pod, the system will work as described in abstract design section above.

Initially, the design intended for only one single instance of the internal intrusion detection system to be deployed per worker node. In the end, it was decided to go with the one per pod design. This requires more resources, however it ensures that if a container in a pod is compromised and the attacker is only interested in other containers in that pod, that this attack will be detected. Additionally, the design is simpler and therefore easier for an end user to deploy. With further time this claim could be investigated further, and the precise trade offs involved calculated.

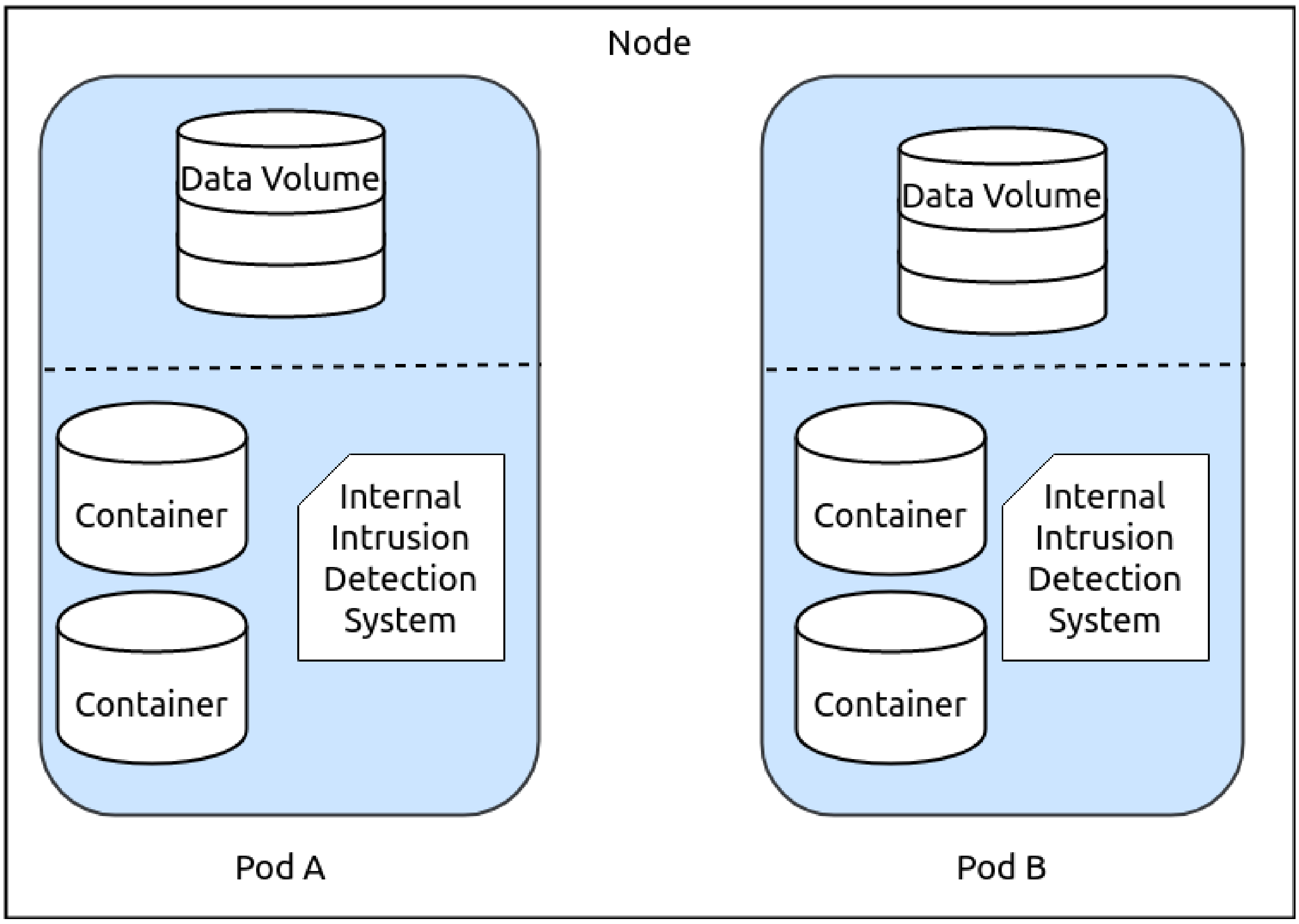


Figure 4.2: Layout of solution in a Kubernetes node

## 4.3 Example Use Case

Figure 4.3 walks through an example of a possible attack against the system, and how this proposal deals with it. A malicious user has gained access to one of the containers in the deployment. In order to identify further targets, including containers holding critical user data, the attacker launches a port scan of the internals of the system. They use a typical TCP stealth SYN scan for this purpose. The port scanner listener detects suspicious TCP packets with RST flags set. It now logs the event and notifies the administrators of what is going on.

The attacker continues to scan and discovers the honeypot. Thinking it to be a valuable system, they attempt to access it. This event is also logged, and now the attacker is wasting time in a fake service instance. The system administrators, alerted to the danger, step in and block all SSH connections to the environment. The attack has been halted.

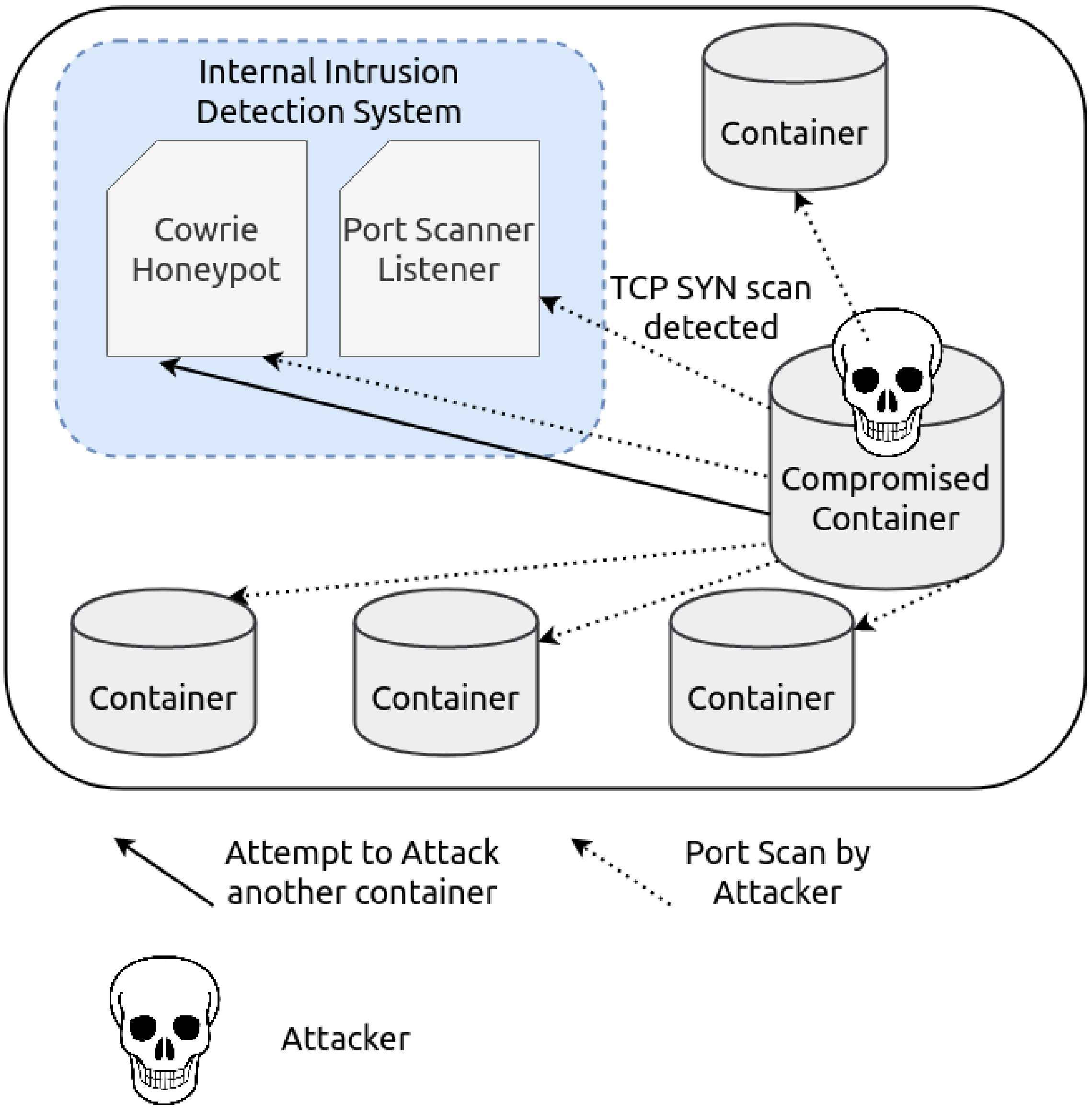


Figure 4.3: Diagram of an example attack in the system

## 4.4 Deployment

It was decided to iteratively deploy a solution. First it would be tested locally. Then a containerised version would be tested while running in a Docker based containerised environment. Finally, the solution would be deployed to a Kubernetes based cloud environment for further performance analysis.

The benefits of this approach are that it is easier to debug any issues that arise locally than in a containerised environment. This in turn is easier to debug than a cloud environment. Furthermore, local experiments enable easy use of tools such as Wireshark, which can monitor packets being sent in a network. This should enable a more detailed look at what is happening when evaluating the performance of the solution. A downside of this plan is that a lot of time is eaten up transferring the design from one deployment scenario to another.

The cloud provider selected for deployment was Amazon Web Services (AWS). Initially, the goal was to verify the integrity of the solution on a wide number of cloud environments, including Google Cloud and Microsoft Azure. Unfortunately, time constraints necessitated a revision of this deployment goal to the current, reduced version. Future work could investigate the performance of this proposal in other deployment scenarios.

## 4.5 Challenges

If an attacker is aware of the honeypot, they can avoid interacting with it. If combined with the knowledge that the port scanner listener exists, the whole system may be bypassed. This is a challenge which is very difficult to solve, but one possibility is to utilise some form of anomaly based detection that would monitor every packet in the system. The downside here is the creation of logs containing every packet interaction in the system. If an attacker becomes aware of this system as well, they may find ways to disguise their packets as benign through some form of steganography. Unfortunetly, there was not enough time to explore this idea.

## 4.6 Ethics

When designing the implementation, it is important to ensure that there are no ethical violations in its design. Some potential ethical issues raised during this research were further attack propagation and the ethics of logging activity.

### 4.6.1 Attack Propagation

If an attacker takes over part of the system, they can use that part to launch further attacks on other victims. Because the proposed solution will only be reachable inside the deployment, there are no concerns about entrapment. However, the concerns over the attacks lauched from the compromised elements remains. This could include spam, DDoS attacks or propagating malware. The way to ethically manage this issue is to ensure that the system design is as secure as possible and to minimise the attack surface outside of the honeypot, which should block attacks launched from itself.

The implications of an attacker using these resources to carry out further attacks are discussed further in the security section.

### 4.6.2 Logging Activity

Logging all suspicious activity in the system raises concerns over possible privacy violations. Ensuring that the logs are securely stored solves half of this problem, but the simple existence of these logs could be viewed as an infringement on user's rights.

However, it is reasonable to assume that an attacker that is trespassing in a system has no reasonable excuse that would to not have their actions recorded. This fact can be reinforced by having the default sign on message notify all users that their actions within the system may be monitored.

## 4.7 Security

While considering the design of this system, it is important to ensure that potential security holes are kept in mind during all stages of development.

### 4.7.1 Risks

Running honeypots in a cloud environment immediately raises a number of security concerns. What happens if an attacker successfully takes over the honeypot, and is then able to use this position to infect or attack other virtual machines or containers or services within the network or system? Another possible concern is that the attacker takes over the honeypot, and then uses it to attack other, external targets. For example, a honeypot being taken over and added to a botnet which is then used to launch a DDoS attack against an external web server.

Another possibility is that an attacker could gain access to virtual machines belonging to other customers of the cloud service provider running on the same hardware. This would occur through some flaw in either the VMs themselves which allows breakout into the system as a whole, or more subtly through some type of side channel attack. The latter would require knowledge of what was also running in the same hardware to be truly effective. Either of these events would be a complete disaster both for me and for the company in question.

This type of risk is not mitigatable by me alone. I am dependant on the cloud providers to prevent this type of vulnerability from being exploitable. Given that a breach of this nature would cause significant reputational damage and likely loss of business, they do take this type of threat very seriously. I am reducing the attack surface by reducing the amount of public facing containers, and open and public facing ports within to the bare minimum, while still maintaining a realistic deployment.

### 4.7.2 Potential Outcomes

Due to the manner in which I am implementing this system, it should never have the honeypots be public facing. This is because the design I am investigating is focused on attackers who have already gained access to a system or network, through obtaining SSH keys that were foolishly stored, or physical access to a machine that is already logged into the system. The goal is to investigate intrusion detection and techniques for mitigating harm when an attacker is within the system. This means that the risk of an attacker gaining access to my cloud deployments is significantly reduced.

However, it is possible that through an error on my part, or on the part of one of the cloud service providers (AWS or Google Cloud), an attacker could gain access to either a container instance, or the worst case my whole cloud environment.If this were to happen, there are a number of possible outcomes, depending on what gets compromised.

The first outcome is simple. An attacker gains control of a container and uses it for their own purpose, such as crypto mining or as part of a botnet. The consequence of the former is simply increased charges on my account. The consequence of the latter is malicious traffic directed against a target in a DDoS attack. The second outcome is that an attacker somehow gains access to my whole cloud environment. This would be even more costly for me financially, and potentially more harmful than the previous case if they are able to utilise all of the containers to do more malicious activities.

The third outcome is that my whole account with the cloud service provider is compromised. This is effectively the same as the previous case but with more serious ramifications. On a personal level, my billing details could be obtained. On an external level, more containers can now be used for DDOS attacks. Additionally, malicious or illegal web content could be hosted which would be harmful for the general public, and very difficult to trace back to the perpetrator if done correctly.

The final outcome is the previously discussed container escape scenario. The result of this would be the potential loss of or leaking of the private data of other services on the same host. The cost of this could be enormous depending on what other resources are sharing the same environment.

To mitigate these issues, I am taking the following steps.Firstly, I am regularly stopping my deployments. When they are restarted, it is from a known good baseline. This is possible because the actual contents of the containers is irrelevant to this research. The benefit this provides is that if something is compromised by an attacker, it is wiped clean and restored from a backup. I am also limiting what is publicly accessible, and the duration of the deployments being active as mentioned earlier.

### 4.7.3 Privacy

Fortunately, I am not handling anyone's personal data, and these AWS and Google Cloud accounts are separate from my personal ones. Therefore,there are no other privacy concerns with my dissertation, outside of those that arise due to the security risks discussed above.

# 5 Implementation

This chapter describes the process by which the solution was implemented. This covers the process of building the system, testing it, containerising it and the challenges faced in deploying a usable solution to the cloud.

Work on the implementation of this research was delayed by the shift in focus of this research described in Chapter 3.

## 5.1 Cowrie Honeypot

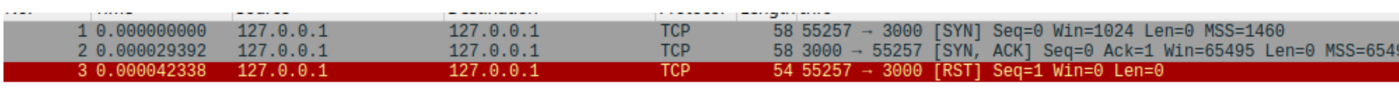
Cowrie is a medium interaction honeypot, which is accessible via TelNet and SSH connections. Cowrie was selected as the honeypot for this proof of concept as it fits in with most of the requirements of this solution. It is able to log attacker's attempts to access it and to log their interactions with its fake shell environment.

A containerised version of Cowrie is available on Github. This is the base implementation used in this research. A local copy of the image can be obtained and run using the command "sudo docker run -p 2222:2222/tcp cowrie/cowrie", which will pull down the image from Docker hub and then run the container with port 2222 exposed to TCP connections.

## 5.2 Port Scanner Listener

The port scanner listener would form an additional layer of security by adding a passive method for detecting port scanning attempts. It was decided to focus on the typical TCP SYN stealth scan as a proof of concept. The idea is easily extended to any of the common scanning techniques.

Figure 5.1 shows a capture of a local test of the SYN scan against the port scanner listener using nmap with port 3000 open. The initial SYN commencing the handshake is sent to the target port 3000 from a port controlled by nmap, in this case 127.0.0.1:55257. As the port is open, a SYN/ACK response is sent. Nmap then responds with the RST packet. This interaction was successfully detected by the port scanner listener. Further details are discussed in Chapter 6.



```
1 0.000000000   127.0.0.1     127.0.0.1     TCP      58 55257 → 3000 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2 0.000029392   127.0.0.1     127.0.0.1     TCP      58 3000 → 55257 [SYN, ACK] Seq=0 Ack=1 Win=65495 Len=0 MSS=654
3 0.000042338   127.0.0.1     127.0.0.1     TCP      54 55257 → 3000 [RST] Seq=1 Win=0 Len=0
```

Figure 5.1: A wireshark capture showing a test TCP SYN scan run against the port scanner listener

### 5.2.1 Packetfu

The Packetfu Ruby gem was used to give mid level access to network interfaces and packets. Packetfu is a library that supports the creation, manipulation and capture of packets. It also supports the parsing of network packets, including individual IP, UDP and TCP packets. It also enables configuring the headers of these packets, and the ability to send and recieve packets manually.

Packetfu was utilised to monitor every TCP packet received at the listener container for suspicious activity. The active network interface was selected to capture packets from, as this is where a port scanner's TCP packets would be handled by the operating system. The capture class enables the gathering of all raw packets received at a specific network interface. This class was used to create a stream of packets, which can then be easily monitored for behaviour similar to that of a port scanner.

Packetfu depends on the libpcap-dev package. Without this package, the gem cannot be installed to the host system. Use of this package requires root permissions. For this reason, the port scanner listener should not have any other functions which could be exploited to gain access to it.

### 5.2.2 Listener Implementation

It was decided to implement the port scanner listener in the form of a Ruby script. Ruby was chosen because of its ease of use, portability, decent networking performance and the author's familiarity with the language.

Below is a high level overview of the Listener class, used to implement the main functionality in the script. The network interface is configurable in the script body, which enables testing against any of the active interfaces, including localhost and whatever interface is being used as the primary network interface.

```
include PacketFu

class Listener
    def initialize(interface, portsArr)
```

```
        @iface = interface
        @rstCount = 0
        @ports = openPorts(portsArr)
        puts @iface
    end
    #open limited number of ports to have a valid TCP handshake
    def openPorts(portsArr)

    #close ports when we are done with them
    def closePorts

    #listen for port scanning attempts
    def listen

    #log incidents
    def log(packet, type)

    #send alarm notification
    def notify(packet)
end
```

The reason we most open some ports is to ensure that we detect RST TCP packets. If every port is closed, then the port scanner will not send an RST because the three way handshake wont proceed to the third step. It is ensured that the open ports do not accept any data other than the TCP handshake packets. This is a necessary risk in order to detect the TCP stealth SYN scan.

The key component of the listener is the logic that detects RST bits set in the TCP header of a packet. A simplified version of this is shown below. First it is determined if the packet at the network interface is a TCP packet. If it is, we check for RST in the TCP header. If it has been identified, then we can proceed to log the event and notify adminsitrators of the incident. The notification is done via email in the current design.

```
    capture.stream.each do |pack|
            packet = Packet.parse(pack)

            next unless packet.is_tcp?
            if packet.tcp_flags.rst == 1
                @rstCount += 1
                log(packet, "RST")
                notify(packet)
            end
        end
```

### 5.2.3 Creating the Listener Container

Once the port scanner listener was verified to work locally, a Docker container image needed to be created to test its functionality in a containerised environment.

As the script is written in the Ruby language, it was decided to use the official Docker Ruby image from Docker Hub as a baseline. This image contains all of the necessary libraries to run a Ruby application, including the Ruby interpreter itself, gem for handling external libraries and. All of this is packaged on a Debian environment.

In order to automate the build process, a Dockerfile was written which sucessfully creates a container image with all dependancies and the port scanner listener. A Dockerfile is a script which specifies the commands necessary to build a functional container image. Running the docker build command with the Dockerfile as a parameter will result in Docker creating the image including any necessary dependencies. If the image is based on another image as in this case, Docker retains a hash indicating the parent image of this image.

The main portion of the Dockerfile is shown here.

```
FROM ruby

RUN apt-get update && apt-get upgrade -y
RUN apt-get install -y libpcap-dev
RUN gem install packetfu

COPY listener.rb .

CMD ["ruby", "listener.rb"]
```

This first downloads the official Ruby image, before updating the system's packages. Then libpcap-dev and packetfu are downloaded and installed. Next, the script is installed into the container. Finally, the default command issued when the container is launched is specified. This is to have the Ruby interpreter run the port scanner listener script.

Running the Docker run command will create a container instance and then immediately launch the container with the port scanner listener already active and listening for connections. This greatly simplifies the deployment process. The container can be stopped by issuing the command docker stop.

## 5.3 Local Deployment

The port scanner listener was first tested locally. This was done to verify that everything worked before proceeding any further.

An issue arose when it was not possible to detect the packets unless scanning was done against the loopback address, localhost, which is the internal operating system network interface. This would occur if the port scanner listener was listening on the network interface used by the laptop running the test, even when the correct local IP address was specified. Using wireshark revealed that packets were sent on the local network when specifying a different local address, but not when specifying the address that the listener was active on.

It was revealed after studying the problem that the operating system was likely to blame. As the packets being sent by nmap were destined for the same machine they were being sent over localhost instead. This meant that the packets never left the test machine and therefore never reached the local network. Switching to reading from localhost enabled these seemingly missing packets to be picked up.

It was decided to not test Cowrie locally, as running it on the machine it is being tested on does not give many useful insights.

## 5.4 Containerised Deployment

To test that the system worked in a containerised environment required using the Cowrie image sourced earlier, and the port scanner listener image created above. Docker was chosen to be the container engine for this test deployment, as it had already been used to create the container image.

The test run in the containerised environment encountered one issue, packets were not being flagged as suspicious. Reviewing the script revealed that an error had been made when setting up the network interface. The script was not specifying the default Docker interface, which is what is used for inter container communications. Changing the script to use this interface fixed the problem.

## 5.5 Cloud Deployment

The final stage of the implementation involves deploying the solution into a cloud environment.

### 5.5.1 Understanding Kubernetes in the Cloud

A significant portion of time was spent understanding how Kubernetes based cloud deployments work. Study revealed that it is common for certain features of the deployment, such as the control plane, to be distributed across multiple instances.

It was decided to use AWS's Amazon Elastic Kubernetes Service (EKS) to host the implementation. EKS was chosen because it is directly offered by AWS and as such was already fully integrated with AWS.

Accessing individual containers being hosted by EKS via SSH and listening for others was more complicated than first expected. Previous experience with testing ordinary AWS instances had not revealed this problem so it came as a big surprise. This resulted in difficulty deploying the final implementation on Kubernetes in the cloud.

### 5.5.2 Email Alerts

Unfortunately, the project period came to an end before an alert system that worked in the cloud environment could be fully implemented.

# 6 Evaluation

Evaluation of the proposed solution was done via carrying out several experiments, and analysing the scenarios.

## 6.1 Challenges

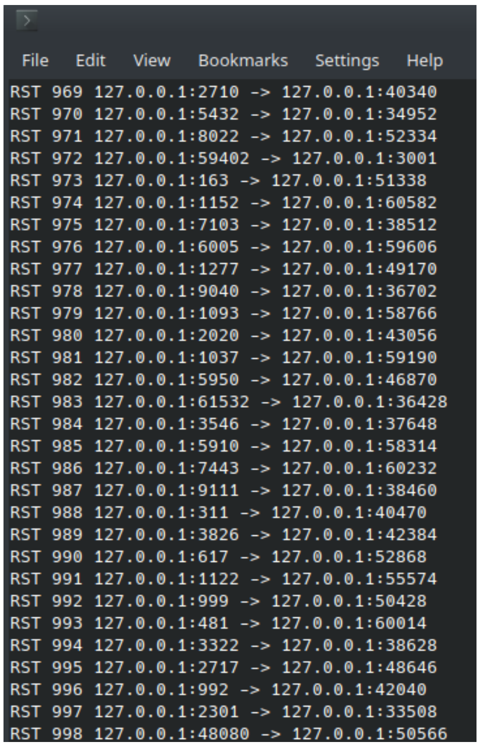
Testing this solution in a rigorous manner provides a number of problems. These problems limit the experiments that can feasibly be carried out.

- Obtaining precise metrics on how much more secure the solution is than the baseline is very difficult, because the concept is hard to quantify
- Leaving the system publically accessible would be risky, and would not simulate the primary type of intrusion that this research has set out to mitigate, namely an attacker gaining access to the deployment using obtained SSH credentials or a company machine that is still logged in
- Therefore, the experiments would have to deal with a simulated attack
- Asking a third party to conduct an attack is somewhat infeasible, and would not give a anymore of a true evaluation of the system as they would be aware that some security measures are being tested. They would therefore be more likely to behave more cautiously than they would against a real target.

To get around these challenges, it was decided to compare the suggested solution implementation to a baseline deployment without the security features under test from an attacker's point of view. The two deployments were compared in two main experiments. The first was to see how the addition of the security features alters an attacker's perspective. The second was to test what happens when an attacker attempts to port scan the deployment to find open ports. Before these experiments were conducted, the port scanner listener was evaluated locally to test if it could detect a port scanning attempt.

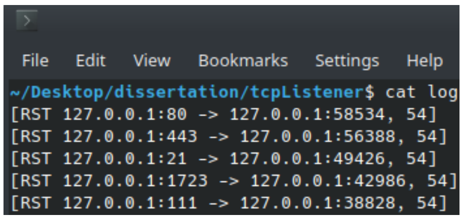
## 6.2 Testing the Port Scanner Listener

To conduct this test, the port scanner listener script was run locally in one process, while nmap was run in another process. The parameters passed to nmap were "-sS", to enable a TCP SYN scan, "-p 2997-3003", to scan the ports in the range 2997 to 3003, and "localhost" to indicate that the target was running locally on the test machine. Figure 5.1 shows a Wireshark capture of part of the port scan.



```
RST 969 127.0.0.1:2710 -> 127.0.0.1:40340
RST 970 127.0.0.1:5432 -> 127.0.0.1:34952
RST 971 127.0.0.1:8022 -> 127.0.0.1:52334
RST 972 127.0.0.1:59402 -> 127.0.0.1:3001
RST 973 127.0.0.1:163 -> 127.0.0.1:51338
RST 974 127.0.0.1:1152 -> 127.0.0.1:60582
RST 975 127.0.0.1:7103 -> 127.0.0.1:38512
RST 976 127.0.0.1:6005 -> 127.0.0.1:59606
RST 977 127.0.0.1:1277 -> 127.0.0.1:49170
RST 978 127.0.0.1:9040 -> 127.0.0.1:36702
RST 979 127.0.0.1:1093 -> 127.0.0.1:58766
RST 980 127.0.0.1:2020 -> 127.0.0.1:43056
RST 981 127.0.0.1:1037 -> 127.0.0.1:59190
RST 982 127.0.0.1:5950 -> 127.0.0.1:46870
RST 983 127.0.0.1:61532 -> 127.0.0.1:36428
RST 984 127.0.0.1:3546 -> 127.0.0.1:37648
RST 985 127.0.0.1:5910 -> 127.0.0.1:58314
RST 986 127.0.0.1:7443 -> 127.0.0.1:60232
RST 987 127.0.0.1:9111 -> 127.0.0.1:38460
RST 988 127.0.0.1:311 -> 127.0.0.1:40470
RST 989 127.0.0.1:3826 -> 127.0.0.1:42384
RST 990 127.0.0.1:617 -> 127.0.0.1:52868
RST 991 127.0.0.1:1122 -> 127.0.0.1:55574
RST 992 127.0.0.1:999 -> 127.0.0.1:50428
RST 993 127.0.0.1:481 -> 127.0.0.1:60014
RST 994 127.0.0.1:3322 -> 127.0.0.1:38628
RST 995 127.0.0.1:2717 -> 127.0.0.1:48646
RST 996 127.0.0.1:992 -> 127.0.0.1:42040
RST 997 127.0.0.1:2301 -> 127.0.0.1:33508
RST 998 127.0.0.1:48080 -> 127.0.0.1:50566
```

Figure 6.1: Live output from the port scanner listener during a port scan



```
~/Desktop/dissertation/tcpListener$ cat log
[RST 127.0.0.1:80 -> 127.0.0.1:58534, 54]
[RST 127.0.0.1:443 -> 127.0.0.1:56388, 54]
[RST 127.0.0.1:21 -> 127.0.0.1:49426, 54]
[RST 127.0.0.1:1723 -> 127.0.0.1:42986, 54]
[RST 127.0.0.1:111 -> 127.0.0.1:38828, 54]
```

Figure 6.2: The log of the local port scan

### Result

The listener successfully detected the scanning attempt. The print out results are shown in Figure 6.1, while the logs stored are shown in Figure 6.2. It should be noted that the RSTs from closed ports are also printed here.

## 6.3 Experiment: Attacker's Perspective

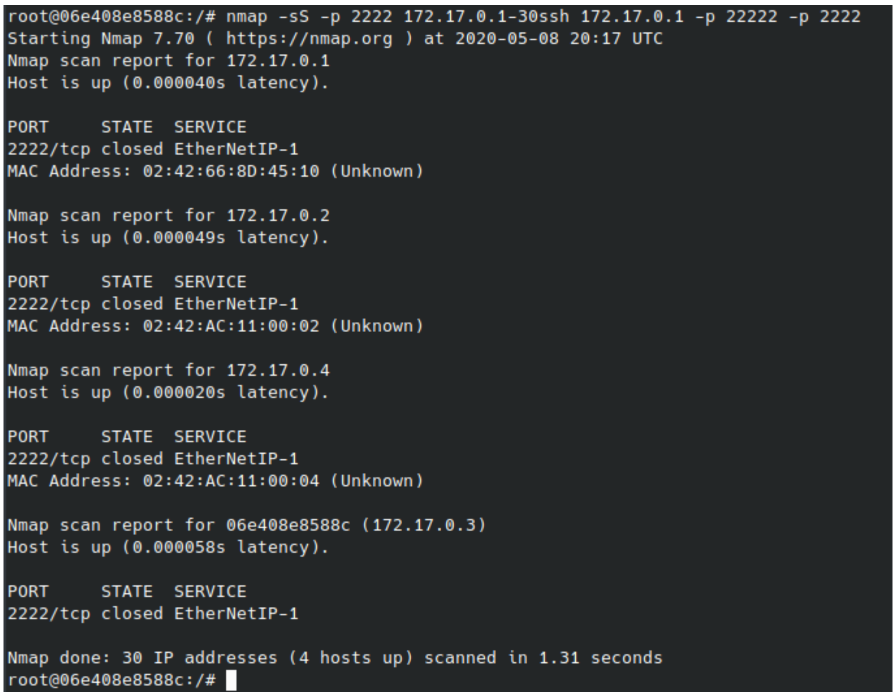
This experiment reviews what details an attacker is presented with if they gain access to the internals of the deployment.

## 6.4 Containerised Deployment

### Baseline Deployment

To test the attacker's perspective in the baseline deployment, one container was launched and used as the compromised container controlled by the attacker. It had nmap installed to enable discovery of other containers in the network. The other containers running were instances of the Ruby container image, used here as example application containers.

The attacker runs the command "nmap -sS -p 2222 172.17.0.1-30", to scan the local network for other vulnerable containers. The results of the attacker's infiltration are shown in Figure 6.3.



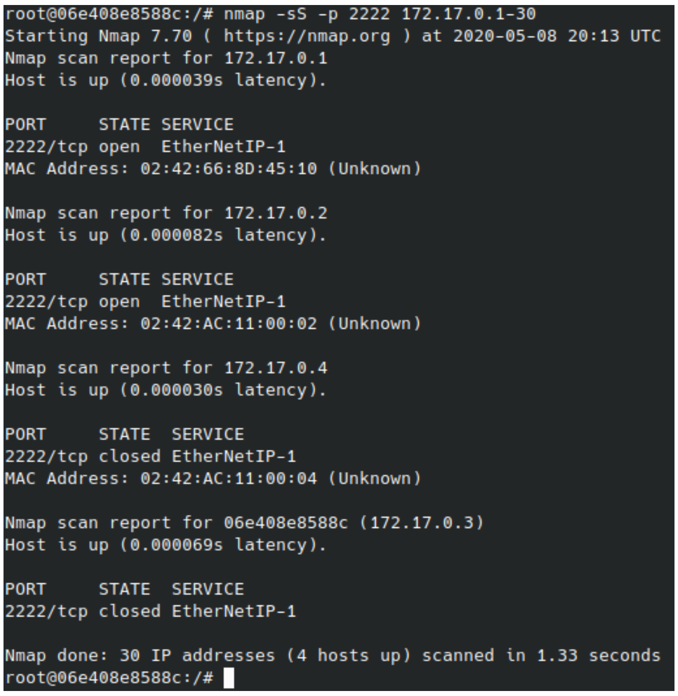
```
root@06e408e8588c:/# nmap -sS -p 2222 172.17.0.1-30ssh 172.17.0.1 -p 22222 -p 2222
Starting Nmap 7.70 ( https://nmap.org ) at 2020-05-08 20:17 UTC
Nmap scan report for 172.17.0.1
Host is up (0.000040s latency).

PORT     STATE  SERVICE
2222/tcp closed EtherNetIP-1
MAC Address: 02:42:66:8D:45:10 (Unknown)

Nmap scan report for 172.17.0.2
Host is up (0.000049s latency).

PORT     STATE  SERVICE
2222/tcp closed EtherNetIP-1
MAC Address: 02:42:AC:11:00:02 (Unknown)

Nmap scan report for 172.17.0.4
Host is up (0.000020s latency).

PORT     STATE  SERVICE
2222/tcp closed EtherNetIP-1
MAC Address: 02:42:AC:11:00:04 (Unknown)

Nmap scan report for 06e408e8588c (172.17.0.3)
Host is up (0.000058s latency).

PORT     STATE  SERVICE
2222/tcp closed EtherNetIP-1

Nmap done: 30 IP addresses (4 hosts up) scanned in 1.31 seconds
root@06e408e8588c:/# 
```

Figure 6.3: Attackers view upon scanning an unprotected deployment

### Solution Deployment

Again, the attacker runs the command "nmap -sS -p 2222 172.17.0.1-30". The output is shown in Figure 6.4.

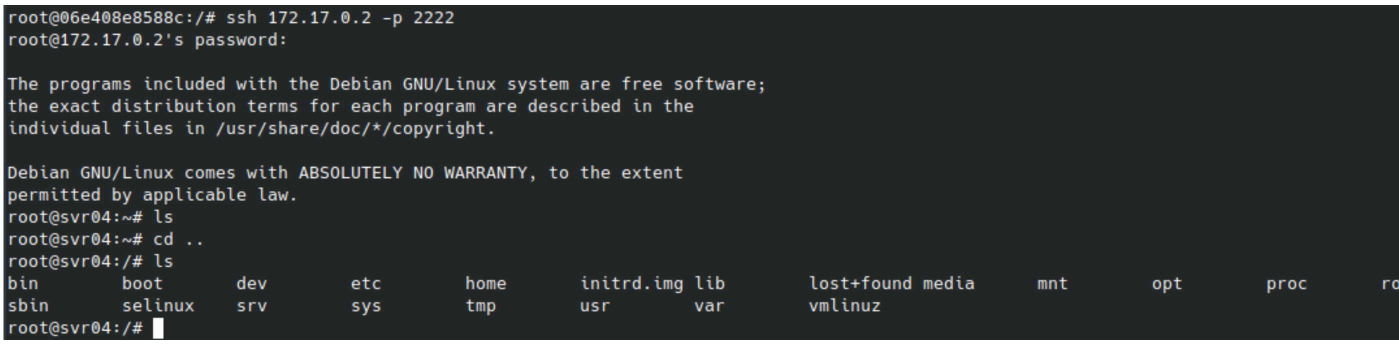
An open port is revealed and the attacker attempts to connect via SSH at that port. They



```
root@06e408e8588c:/# nmap -sS -p 2222 172.17.0.1-30
Starting Nmap 7.70 ( https://nmap.org ) at 2020-05-08 20:13 UTC
Nmap scan report for 172.17.0.1
Host is up (0.000039s latency).

PORT     STATE SERVICE
2222/tcp open  EtherNetIP-1
MAC Address: 02:42:66:8D:45:10 (Unknown)

Nmap scan report for 172.17.0.2
Host is up (0.000082s latency).

PORT     STATE SERVICE
2222/tcp open  EtherNetIP-1
MAC Address: 02:42:AC:11:00:02 (Unknown)

Nmap scan report for 172.17.0.4
Host is up (0.000030s latency).

PORT     STATE  SERVICE
2222/tcp closed EtherNetIP-1
MAC Address: 02:42:AC:11:00:04 (Unknown)

Nmap scan report for 06e408e8588c (172.17.0.3)
Host is up (0.000069s latency).

PORT     STATE  SERVICE
2222/tcp closed EtherNetIP-1

Nmap done: 30 IP addresses (4 hosts up) scanned in 1.33 seconds
root@06e408e8588c:/#
```

Figure 6.4: Attackers view upon scanning the protected deployment

succeed and attempt to login as root with the password "password". This also succeeds and they are presented with an interactive bash shell as the root user in a Debian environment. This is shown in Figure 6.5.



```
root@06e408e8588c:/# ssh 172.17.0.2 -p 2222
root@172.17.0.2's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@svr04:~# ls
root@svr04:~# cd ..
root@svr04:/# ls
bin        boot       dev        etc        home       initrd.img lib        lost+found media      mnt        opt        proc       ro
sbin       selinux    srv        sys        tmp        usr        var        vmlinuz
root@svr04:/#
```

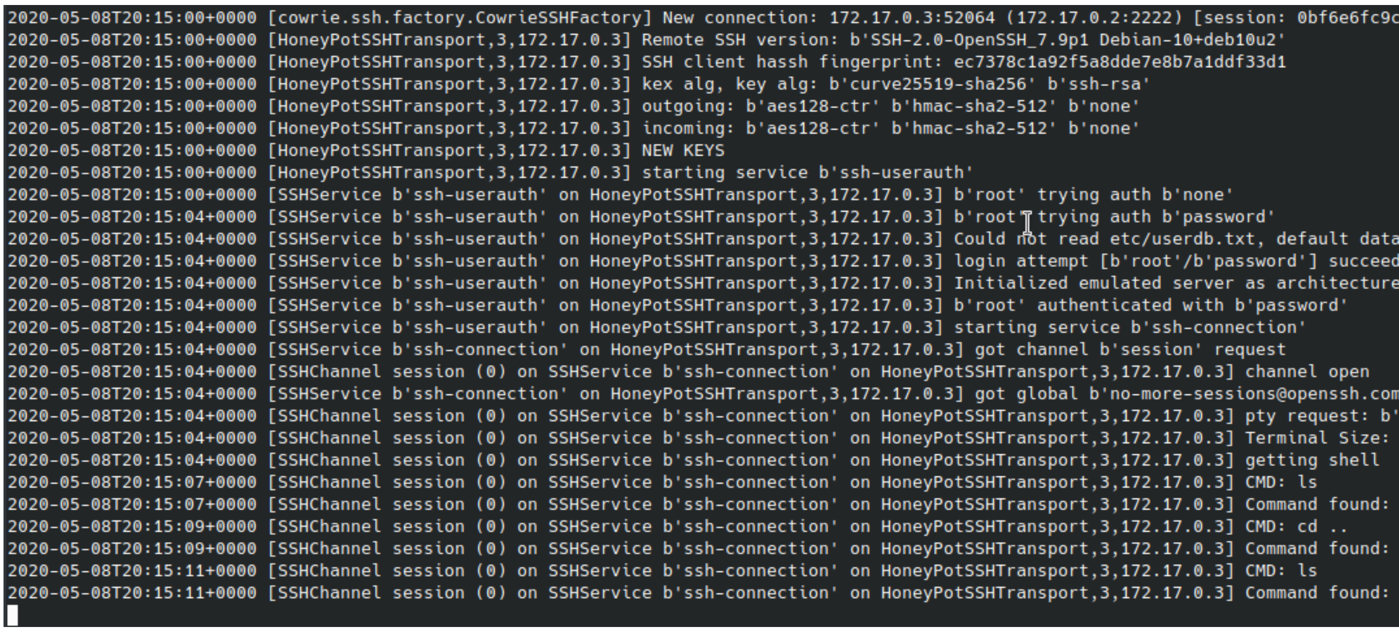
Figure 6.5: Attackers view upon attempting to access the honeypot

### Results

The baseline deployment offered only the containers that were in actual use to the attacker. The attacker can now attempt to gain access to any of them without any form of intrusion detection picking up what is going on.

The solution deployment immediately offers the attacker an easy target, the container at IP address 172.17.0.2. The attacker attempts to gain access using some form of default credentials, in this case "root" as the user and "password" as the password. The attacker gains access and is presented with an interactive shell representing a supposedly real system.

The attacker now has access as root to this container, and will likely be inclined to explore it to find sensitive information.



```
2020-05-08T20:15:00+0000 [cowrie.ssh.factory.CowrieSSHFactory] New connection: 172.17.0.3:52064 (172.17.0.2:2222) [session: 0bf6e6fc9c
2020-05-08T20:15:00+0000 [HoneyPotSSHTransport,3,172.17.0.3] Remote SSH version: b'SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2'
2020-05-08T20:15:00+0000 [HoneyPotSSHTransport,3,172.17.0.3] SSH client hassh fingerprint: ec7378c1a92f5a8dde7e8b7a1ddf33d1
2020-05-08T20:15:00+0000 [HoneyPotSSHTransport,3,172.17.0.3] kex alg, key alg: b'curve25519-sha256' b'ssh-rsa'
2020-05-08T20:15:00+0000 [HoneyPotSSHTransport,3,172.17.0.3] outgoing: b'aes128-ctr' b'hmac-sha2-512' b'none'
2020-05-08T20:15:00+0000 [HoneyPotSSHTransport,3,172.17.0.3] incoming: b'aes128-ctr' b'hmac-sha2-512' b'none'
2020-05-08T20:15:00+0000 [HoneyPotSSHTransport,3,172.17.0.3] NEW KEYS
2020-05-08T20:15:00+0000 [HoneyPotSSHTransport,3,172.17.0.3] starting service b'ssh-userauth'
2020-05-08T20:15:00+0000 [SSHService b'ssh-userauth' on HoneyPotSSHTransport,3,172.17.0.3] b'root' trying auth b'none'
2020-05-08T20:15:04+0000 [SSHService b'ssh-userauth' on HoneyPotSSHTransport,3,172.17.0.3] b'root' trying auth b'password'
2020-05-08T20:15:04+0000 [SSHService b'ssh-userauth' on HoneyPotSSHTransport,3,172.17.0.3] Could not read etc/userdb.txt, default data
2020-05-08T20:15:04+0000 [SSHService b'ssh-userauth' on HoneyPotSSHTransport,3,172.17.0.3] login attempt [b'root'/b'password'] succeed
2020-05-08T20:15:04+0000 [SSHService b'ssh-userauth' on HoneyPotSSHTransport,3,172.17.0.3] Initialized emulated server as architecture
2020-05-08T20:15:04+0000 [SSHService b'ssh-userauth' on HoneyPotSSHTransport,3,172.17.0.3] b'root' authenticated with b'password'
2020-05-08T20:15:04+0000 [SSHService b'ssh-userauth' on HoneyPotSSHTransport,3,172.17.0.3] starting service b'ssh-connection'
2020-05-08T20:15:04+0000 [SSHService b'ssh-connection' on HoneyPotSSHTransport,3,172.17.0.3] got channel b'session' request
2020-05-08T20:15:04+0000 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,3,172.17.0.3] channel open
2020-05-08T20:15:04+0000 [SSHService b'ssh-connection' on HoneyPotSSHTransport,3,172.17.0.3] got global b'no-more-sessions@openssh.com
2020-05-08T20:15:04+0000 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,3,172.17.0.3] pty request: b'
2020-05-08T20:15:04+0000 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,3,172.17.0.3] Terminal Size:
2020-05-08T20:15:04+0000 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,3,172.17.0.3] getting shell
2020-05-08T20:15:07+0000 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,3,172.17.0.3] CMD: ls
2020-05-08T20:15:07+0000 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,3,172.17.0.3] Command found:
2020-05-08T20:15:09+0000 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,3,172.17.0.3] CMD: cd ..
2020-05-08T20:15:09+0000 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,3,172.17.0.3] Command found:
2020-05-08T20:15:11+0000 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,3,172.17.0.3] CMD: ls
2020-05-08T20:15:11+0000 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,3,172.17.0.3] Command found:
```

Figure 6.6: The honeypot's logs of the attackers interactions with it

Meanwhile, the honeypot container is logging their interactions with the fake shell. These logs can be used to build a picture of the attacker's interactions with the system. The logs obtained from this experiment are shown in Figure 6.6.

## 6.5 Experiment: Port Scanning

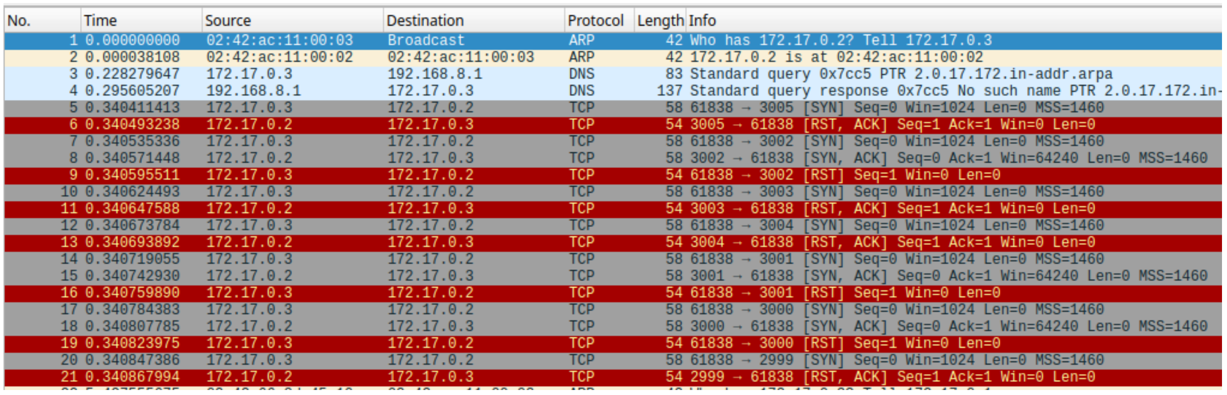
This experiment is intended to demonstrate how an attacker can utilise a port scanner to identify targets and map out the network, and also to demonstrate the difference between the baseline and the proposed implementation.

## 6.6 Containerised Deployment

The experiment was run against a containerised deployment of the solution, consisting of the two containers plus a third for the attacker to control. It was run against a similar baseline scenario with two Ruby container images running alongside an attacker on a third container. The attacker's container had nmap installed to conduct the experiment. A Wireshark capture of part of the scan is demonstrated in Figure 6.7.

### Baseline Deployment

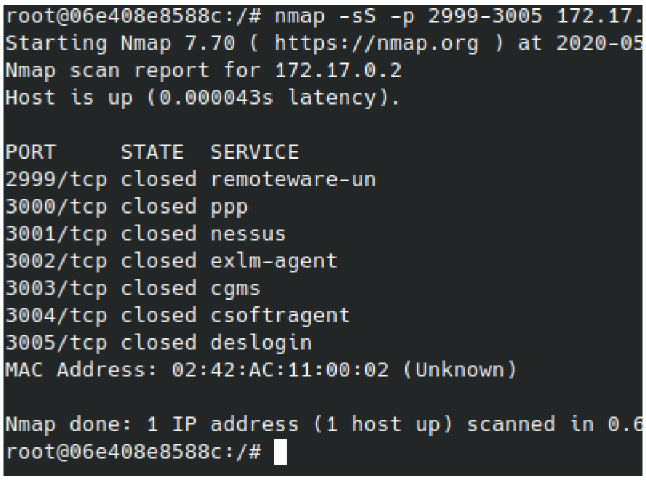
Nmap was run with the following command as root "nmap -sS -p 2999-3005 [Target IP Address]".



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 02:42:ac:11:00:03 | Broadcast | ARP | 42 | Who has 172.17.0.2? Tell 172.17.0.3 |
| 2 | 0.000038108 | 02:42:ac:11:00:02 | 02:42:ac:11:00:03 | ARP | 42 | 172.17.0.2 is at 02:42:ac:11:00:02 |
| 3 | 0.228279647 | 172.17.0.3 | 192.168.8.1 | DNS | 83 | Standard query 0x7cc5 PTR 2.0.17.172.in-addr.arpa |
| 4 | 0.295605207 | 192.168.8.1 | 172.17.0.3 | DNS | 137 | Standard query response 0x7cc5 No such name PTR 2.0.17.172.in- |
| 5 | 0.340411413 | 172.17.0.3 | 172.17.0.2 | TCP | 58 | 61838 → 3005 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 6 | 0.340493238 | 172.17.0.2 | 172.17.0.3 | TCP | 54 | 3005 → 61838 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 7 | 0.340535336 | 172.17.0.3 | 172.17.0.2 | TCP | 58 | 61838 → 3002 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 8 | 0.340571448 | 172.17.0.2 | 172.17.0.3 | TCP | 58 | 3002 → 61838 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 |
| 9 | 0.340595511 | 172.17.0.3 | 172.17.0.2 | TCP | 54 | 61838 → 3002 [RST] Seq=1 Win=0 Len=0 |
| 10 | 0.340624493 | 172.17.0.3 | 172.17.0.2 | TCP | 58 | 61838 → 3003 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 11 | 0.340647588 | 172.17.0.2 | 172.17.0.3 | TCP | 54 | 3003 → 61838 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 12 | 0.340673784 | 172.17.0.3 | 172.17.0.2 | TCP | 58 | 61838 → 3004 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 13 | 0.340693892 | 172.17.0.2 | 172.17.0.3 | TCP | 54 | 3004 → 61838 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 14 | 0.340719055 | 172.17.0.3 | 172.17.0.2 | TCP | 58 | 61838 → 3001 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 15 | 0.340742930 | 172.17.0.2 | 172.17.0.3 | TCP | 58 | 3001 → 61838 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 |
| 16 | 0.340759890 | 172.17.0.3 | 172.17.0.2 | TCP | 54 | 61838 → 3001 [RST] Seq=1 Win=0 Len=0 |
| 17 | 0.340784383 | 172.17.0.3 | 172.17.0.2 | TCP | 58 | 61838 → 3000 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 18 | 0.340807785 | 172.17.0.2 | 172.17.0.3 | TCP | 58 | 3000 → 61838 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 |
| 19 | 0.340823975 | 172.17.0.3 | 172.17.0.2 | TCP | 54 | 61838 → 3000 [RST] Seq=1 Win=0 Len=0 |
| 20 | 0.340847386 | 172.17.0.3 | 172.17.0.2 | TCP | 58 | 61838 → 2999 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 21 | 0.340867994 | 172.17.0.2 | 172.17.0.3 | TCP | 54 | 2999 → 61838 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |

Figure 6.7: Wireshark capture of the port scanning attempt in the containerised environment

The results in the baseline deployment are shown in Figure 6.8.



```
root@06e408e8588c:/# nmap -sS -p 2999-3005 172.17.
Starting Nmap 7.70 ( https://nmap.org ) at 2020-05
Nmap scan report for 172.17.0.2
Host is up (0.000043s latency).

PORT     STATE  SERVICE
2999/tcp closed remoteware-un
3000/tcp closed ppp
3001/tcp closed nessus
3002/tcp closed exlm-agent
3003/tcp closed cgms
3004/tcp closed csoftragent
3005/tcp closed deslogin
MAC Address: 02:42:AC:11:00:02 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.6
root@06e408e8588c:/#
```

Figure 6.8: Port Scanner's view during an attempted port scan in a baseline containerised environment

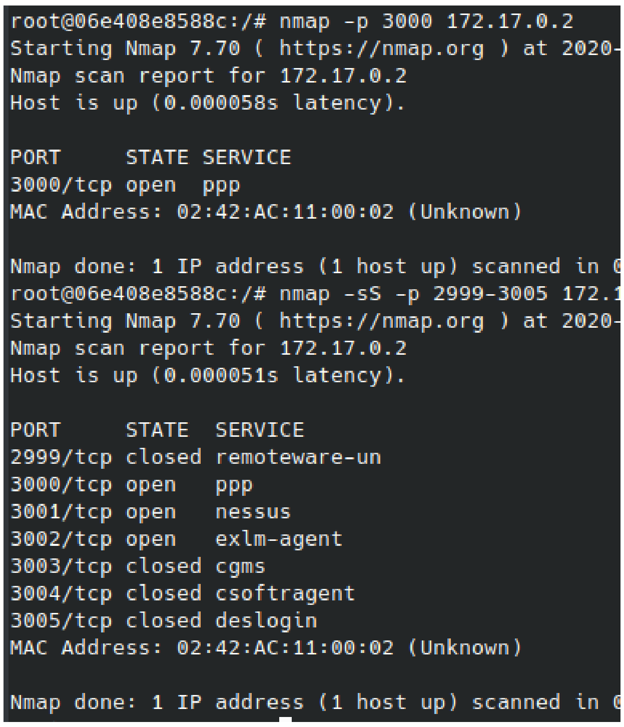
### Solution Deployment

The same command was run again against a solution deployment. The results are shown in Figure 6.9 and Figure 6.10.

### Result

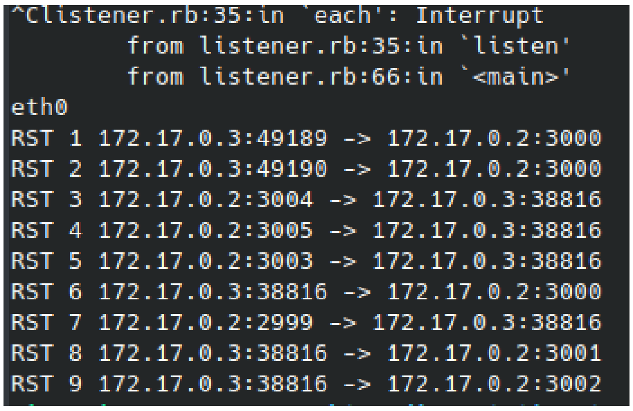
Both the baseline and solution deployments did not prevent the attacker from port scanning the network, however only the solution deployment was able to detect that the scanning was taking place. It successfully logged the incident. If the attacker proceeded to exfiltrate sensitive data, there would be little evidence of suspicious activity on the baseline, let alone an alert that something was wrong.

The ports listed as open in the solution deployment are the ones set to be open by the port scanner listener. Attempting to connect to these ports using netcat leads to no results as the ports will not respond to data sent to them outside of connection attempts.



```
root@06e408e8588c:/# nmap -p 3000 172.17.0.2
Starting Nmap 7.70 ( https://nmap.org ) at 2020-
Nmap scan report for 172.17.0.2
Host is up (0.000058s latency).

PORT     STATE SERVICE
3000/tcp open  ppp
MAC Address: 02:42:AC:11:00:02 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0
root@06e408e8588c:/# nmap -sS -p 2999-3005 172.1
Starting Nmap 7.70 ( https://nmap.org ) at 2020-
Nmap scan report for 172.17.0.2
Host is up (0.000051s latency).

PORT     STATE  SERVICE
2999/tcp closed remoteware-un
3000/tcp open   ppp
3001/tcp open   nessus
3002/tcp open   exlm-agent
3003/tcp closed cgms
3004/tcp closed csoftragent
3005/tcp closed deslogin
MAC Address: 02:42:AC:11:00:02 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0
```

Figure 6.9: Port Scanner's view during an attempted port scan in a solution based containerised environment



```
^Clistener.rb:35:in `each': Interrupt
        from listener.rb:35:in `listen'
        from listener.rb:66:in `<main>'
eth0
RST 1 172.17.0.3:49189 -> 172.17.0.2:3000
RST 2 172.17.0.3:49190 -> 172.17.0.2:3000
RST 3 172.17.0.2:3004 -> 172.17.0.3:38816
RST 4 172.17.0.2:3005 -> 172.17.0.3:38816
RST 5 172.17.0.2:3003 -> 172.17.0.3:38816
RST 6 172.17.0.3:38816 -> 172.17.0.2:3000
RST 7 172.17.0.2:2999 -> 172.17.0.3:38816
RST 8 172.17.0.3:38816 -> 172.17.0.2:3001
RST 9 172.17.0.3:38816 -> 172.17.0.2:3002
```

Figure 6.10: Port scanner listener output in the PS containerised experiment

# 7 Conclusion

This research achieved most of the goals it set out to achieve. Unfortunately, there was also a failure to fully complete all objectives necessary to have a fully functional implementation. Useful insights into secure cloud deployments were still gathered and are discussed below.

## 7.1 Implementation Conclusions

Not every aim specified in Chapter 1 was achieved during this research. The biggest issue was time constraints, partially caused by the Coronavirus pandemic. This resulted in the implementation and evaluation of the solution being more limited than was first intended. The goal of a fully functional intrusion detection mechanism for Kubernetes based cloud deployments was not entirely realised, as the incident alerts were not completely implemented.

Despite these setbacks, a simpler form of the solution was created and evaluated. The limited amount of testing showed that the solution had the potential to meet all of the goals it was designed to meet. The missing requirements could be added in a fairly short amount of time.

The importance of some form of intrusion detection was demonstrated in the Evaluation chapter. The baseline deployment had no way to know that it had been infiltrated, so sensitive data could be stolen or vandalised without anyone being aware.

The successfully implemented portions of the design also highlighted the benefits it offers not only compared to an insecure baseline deployment, but also compared to a system containing just one of the two elements of the design. This is due to the redundancy that they offer, the port scanner listener can detect attacks even if they do not take the honeypot's bait. The honeypot ties up an attacker and buys time for administrators to respond, which a port scanner listener by itself could not achieve.

## 7.2 Internet Security

The study of the current security threat landscape emphasised the continued importance of proper secure design patterns when building any internet connected system. The scale of the threat facing such systems in this modern, hyper-connected world was demonstrated.

## 7.3 Intrusion Detection Systems and Honeypots

This research has also successfully demonstrated that there are benefits to borrowing traditional techniques used to secure non-cloud deployments, and transferring them to a production cloud environment. In particular, the potential of honeypots and intrusion detection systems was highlighted.

More advanced adaptive honeypot designs may be key in securing future cloud deployments, as attacks and malware grow in sophistication.

## 7.4 Future Work

A big piece of future work would be to fully expand upon the work done in this research to create a fully formed Kubernetes based intrusion detection system, that meets every goal outlined in the Problem Formulation Chapter.

An alternative design which does not place an intrusion detection system in each individual pod but instead places it in the worker nodes may be worth investigating. Other possible architectures worth exploring are involving some form of honeynet design, having a intrusion detection control plane that manages one or more honeypots and port scanner listeners and the use of more adaptive honeypots which change based on the actions the attacker takes in order to maintain their interest.

Investigating open source intrusion detection systems such as Snort in a similar deployment scenario could also be of interest.

## 7.5 Conclusion

Although a large amount of work is required to transform this proof of concept implementation to a fully usuable, cloud deployable security system, this research has shown the value in pursuing this type of design and how it can improve the security of containerised deployments. It then extrapolates from a containerised deployment to cloud based containerised deployments. The final word on this research is that securing cloud based deployments has been shown to be of the utmost importance.

# Bibliography

[1] Jakub Křoustek Avast. Wannacry ransomware that infected telefonica and nhs hospitals is spreading aggressively, with over 50,000 attacks so far today. 2017, last accessed 8/5/2020. `https://blog.avast.com/ransomware-that-infected-telefonica-and-nhs-hospitals-is-spreading-aggressively-w`

[2] M. Zahid M. A. Qadeer, A. Iqbal and M. R. Siddiqui. Network traffic analysis and intrusion detection using packet sniffer. *2010 Second International Conference on Communication Software and Networks, Singapore*, 2010.

[3] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, page 412–421, 1974.

[4] Cve-2009-1244. *cve.mitre.org*, 2009, last accessed 8/5/20. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1244`.

[5] Cve-2018-12126. *cve.mitre.org*, 2008, last accessed 8/5/20. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-12126`.

[6] What is a container? *docker.com*, last accessed 8/5/20. `https://www.docker.com/resources/what-container`.

[7] A. Spyker A. Leung and T. Bozarth. Titus: Introducing containers to the netflix cloud. *ACM*, 2018.

[8] Asif. Khan. Key characteristics of a container orchestration platform to enable a modern application. *IEEE Cloud Computing*, 2017.

[9] What is kubernetes? *kubernetes.io*, last accessed 8/5/20. `https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/`.

[10] Kubernetes pods. *kubernetes.io*, last accessed 8/5/20. `https://kubernetes.io/docs/concepts/workloads/pods/pod/`.

[11] What is cloud computing? *aws.amazon.com*, last accessed 8/5/20. `https://aws.amazon.com/what-is-cloud-computing/`.

[12] Eugene H. Spafford. The internet worm program: An analysis. *Purdue University*, 1988.

[13] Stuxnet worm hits iran nuclear plant staff computers. *BBC*, 2010.
`https://www.bbc.co.uk/news/world-middle-east-11414483`.

[14] How a dorm room minecraft scam brought down the internet. *wired.com*, last accessed 8/5/2020. `https://www.wired.com/story/mirai-botnet-minecraft-scam-brought-down-the-internet/`.

[15] Tcp syn (stealth) scan (-ss). *nmap*, last accessed 8/5/2020.
`https://nmap.org/book/synscan.html`.

[16] Anders Fogh Daniel Genkin Daniel Gruss Werner Haas Mike Hamburg Moritz Lipp Stefan Mangard Thomas Prescher Michael Schwarz Yuval Yarom Paul Kocher, Jann Horn. Spectre attacks: Exploiting speculative execution. 2017.

[17] Daniel Gruss Thomas Prescher Werner Haas Anders Fogh Jann Horn Stefan Mangard Paul Kocher Daniel Genkin Yuval Yarom Mike Hamburg Moritz Lipp, Michael Schwarz. Meltdown: Reading kernel memory from user space. 2017.

[18] New petya / notpetya / expetr ransomware outbreak. 2017, last accessed 8/5/2020.
`https://www.kaspersky.com/blog/new-ransomware-epidemics/17314/`.

[19] Paul Rascagneres Martin Lee, Warren Mercer and Craig Williams. Player 3 has entered the game: Say hello to 'wannacry'. 2017, last accessed 8/5/2020.
`https://blog.talosintelligence.com/2017/05/wannacry.html`.

[20] Rensenware source code copy. *Github*, last accessed 8/5/2020.
`https://github.com/0x00000FF/rensenware-cut`.

[21] Michael Bailey Matthew Bernhar Elie Bursztein Jaime Cochran Zakir Durumeric J. Alex Halderman Luca Invernizzi Michalis Kallitsis Deepak Kumar Chaz Lever Zane Ma Joshua Mason Damian Menscher Chad Seaman Nick Sullivan Kurt Thomas Yi Zhou Manos Antonakakis, Tim April. Understanding the mirai botnet. 2017.

[22] Alan Litchfield and Abid Shahzad. Virtualization technology: Cross-vm cache side channel attacks make it vulnerable. 2016.

[23] Douglas Lee Schales Daniele Sgandurra Mihai Christodorescu, Reiner Sailer and Diego Zamboni. Cloud security is not (just) virtualization security: a short paper. *Proceedings of the 2009 ACM workshop on Cloud computing security*, 2009.

[24] Anatomy of cve-2019-5736: A runc container escape! *AWS*, last accessed 8/5/2020.
`https://aws.amazon.com/blogs/compute/anatomy-of-cve-2019-5736-a-runc-container-escape//`.

[25] Cliff C. Zou and Ryan Cunningham. Honeypot-aware advanced botnet construction and maintenance. *IEEE*, 2006.

[26] Mark Graham, A. Winckles, and Erika Sanchez. Botnet detection within cloud service provider networks using flow protocols. 2015.

[27] Andronikos Kyriakou and Nicolas Sklavos. Container-based honeypot deployment for the analysis of malicious activity. 2018.