Zero Trust Container Architecture (ZTCA): A Framework for Applying Zero Trust Principals to Docker Containers

Darragh Leahy and Christina Thorpe
Technological University Dublin, Blanchardstown, Dublin, Ireland
ZeroTCA@gmail.com
christina.thorpe@tudublin.ie

Abstract: Containerisation is quickly becoming an accepted industry standard for development environments and Gartner, in a recent market forecast, estimated that by 2022 more than 75% of organisations will be using containers in production deployments. With this explosion in growth comes an added focus on security and best practices for using containers. The use of containers, in particular Docker containers, has altered some of the more traditional deployment paradigms by giving control of deployments to the development teams. This has massively benefited the DevOps release cycle, but at the expense of many mature security and review processes that are integrated into traditional deployments. Like all systems, containers need frameworks to guide best practices for deployments and to ensure mistakes are not made that increase the risk level or attack surface of an application or service using containers, or the containers themselves. Indeed, according to a recent presentation during DevSecCon24 by Justin Cormack, Security Lead at Docker Inc., Cormack believes most security issues related to Docker are due to misconfiguration rather than direct exploit. While work has been previously conducted with regards to container security and separately applying Zero Trust Networking Architecture to containers, in this work we will investigate the security state of a default deployment of the Docker container engine on Linux and analyse how the principals of Zero Trust Architecture can be extended beyond the domain of networking, distilled into a ”Zero Trust Containers Architecture” and applied to secure Docker deployments. In order to determine this, research was conducted into the current state of Docker security and Zero Trust Architecture. Practical and theoretical attacks were reviewed against a default Docker deployment to identify common themes and areas of issue. Results were used to advise a generalised trust-based framework which was then used to analyse a Docker deployment and validate mitigation of a selection of the identified attacks, proving out the concept of the proposed “Zero Trust Container Architecture” framework.

Keywords: Zero-Trust Architecture, Containerisation, Virtualisation, Security, DevSecOps

1. Introduction

Regardless of the forecast advances made by Docker Inc. in securing the product, the current security state of Docker is still a concern. Many security issues do exist currently, this is driven by a combination of technical factors such as vulnerabilities in the Docker Engine and Linux OS, misconfigurations of the Docker system or limitations of its design, however equally important are the concepts of ’system espoused’ versus the ’system in use’ problem. That is, the difference between what the product developer aims to achieve and consequently what they design for, as opposed to how the system is actually used in the real world. Work in (Martin 2018) demonstrated that a fundamental difference exists in what Docker considers the recommended use case versus what is actually being done in the real world. These differences, combined with the technical issues, as well as the rapid adoption of Docker have combined to create a potential storm of security concern. Zero Trust Networking Architecture (ZTA/ZTNA) is a design paradigm for networks which identified the need to move the defensive perimeter down from the edge of the network. In a traditional edge firewall and Intrusion Prevention/Intrusion Detection system, the edge of the network was considered the point of security, with anything inside broadly trusted. ZTA advised to redefine this boundary away from the perimeter and down to the user and resource in use. ZTA re-frames the question of network security into one of trust, that is, the inherent insecurity of assumed trust that exists once inside the network perimeter. ZTA advises to assume no explicit trust within a system, that internal users accessing internal services without explicit authentication and authorisation cannot be trusted any more than external actors. ZTA’s ethos can be summed up with the line ’never trust, always verify’. ZTA offers opportunities to be re-purposed for use in securing Docker containers and thereby mitigating some of the known configuration issues as well as minimise the surface area of exploits and vulnerabilities.

This research will review and identify the issues with a default deployment of Docker through a combination of practical experimentation to answer the core research question: ***Using a novel generalised trust-based framework, based on the proven principals of ZTA, can a Docker deployment be secured?*** Some ancillary questions have also been defined: (1) What issues currently affect a default deployment of Docker? (2) What

additional common misconfigurations affect a Docker deployment? (3) Can ZTA, or an abstraction thereof be applied to help secure a Docker deployment?

Using the strategic thinking and principles underpinning ZTNA, this paper aims to distil a trust-based framework that considers multiple aspects of a Docker container deployment. With the aim of evolving ZTA beyond just the scope of networking. To the best of our knowledge, the re-purposing of ZTA into a generalised trust framework and applying the results to containers has not been done before. The success of this trust-based framework will be validated through practical example and process as well as theoretical example. This is to be achieved by applying the investigated attacks and issues against various Docker use cases reviewed and secured using the new trust-based framework and reasoning out the improved security surface. This paper will focus on the most straightforward deployment of Docker that can be configured, which is also likely to be one of the most common real-world deployment scenarios. To this, we will propose and apply a trust-based framework adapted from ZTA.

2. Background

2.1 Containers

Figure 1 (a): Docker Container versus VM

Containers, as defined by Docker Inc., provide "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". While not all containerisation systems share the same ethos, the fundamental structures, as seen in Figure 1, are broadly similar. Where Virtual Machines (VMs) run on top of a hypervisor 'multiplexing' the hardware, containers run on top of the Host Operating System (Host OS) sharing and 'multiplexing' the kernel. Containerisation offers an alternate solution to full VMs, with the ability to address specific demands for simplicity and speed arising from modern development concepts like microservices. Containers present the perfect solution by offering rapid and idempotent deployment of container images from private or shared repositories, extremely fast boot times and portability. Figure 2 shows an example of just how fast the start time of a Docker container is relative to alternate solutions.

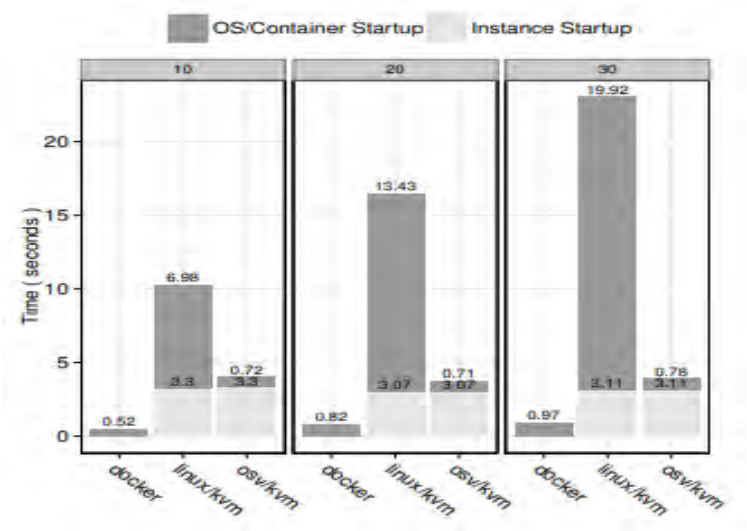

Figure 2: Instance and Operating System/Container Start-up for 10, 20 and 30 instances (Xavier et al., 2016)

2.2 Docker Containers

Docker as a container solution emerged in 2013. Docker views containers as a mechanism for packing specific applications inside a (relatively) idempotent image that can be shared and deployed but eschews the typical features of an OS like system services, daemons, etc. That makes Docker and similar projects much closer to a software distribution mechanism than a machine management tool." (Graber 2016). Another important element of Docker that sets it apart from many other containerisation engines is the sheer ubiquity of adoption, with Docker claiming that container images have been downloaded from the Docker Hub over 130 billion times and that the Docker engine itself is responsible for sparking the 'Containerization Movement'. Docker is one of the most recognisable names in containerisation. This, combined with its developer targeted approach in supporting rapid application development and deployment, has made it the focus of this paper.

2.3 Lifecycle of a Docker Container

Docker container images are built from a reference spec called a 'Dockerfile' (See Figure 3). A Dockerfile is commonly a yaml file (yaml is used by lots of different applications for both configuration and for defining data in a human-readable structured data format) that describes the steps required to assemble an image, allowing developers to select a base OS image, install dependencies and inject a copy of their software. Each of the steps in this file is converted into a 'layer' by the Docker engine with each layer combining to form the final container image. This image then becomes idempotent, such that while individual running instances can be changed, the image itself cannot be, allowing for 'known good' configurations, leading to extreme portability. These images are then pushed to an image repository, either private or shared, e.g., Docker Hub (also supports private use cases). Other developers, Ops teams, or indeed any permitted end user can then pull these images to their Docker deployments and have a container running the image in a highly repeatable, straightforward fashion. Rather than upgrading the container OS or application, when a new container version is released, the new container image can simply be download and the old image discarded.

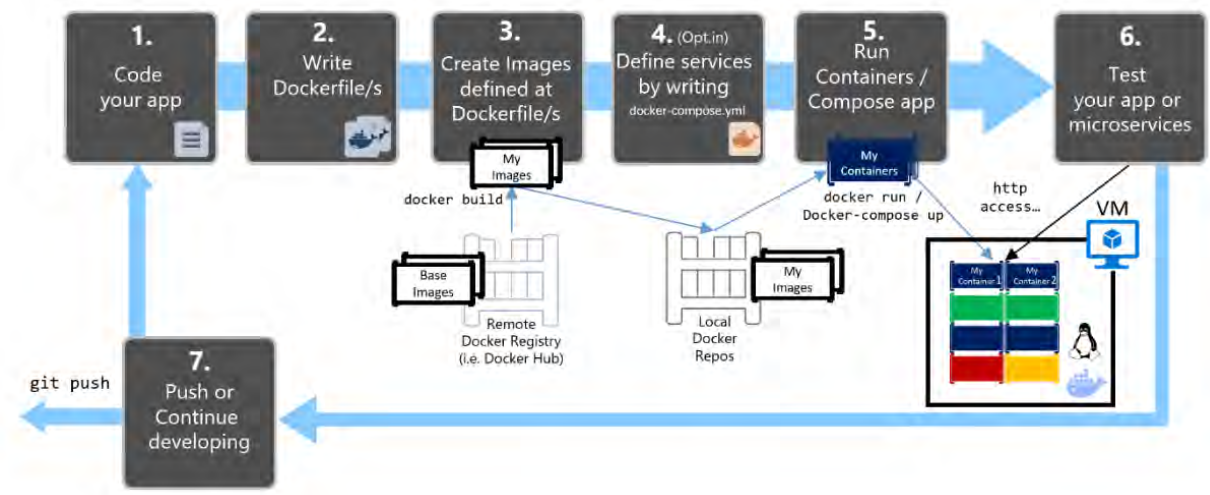

Figure 3: Step-by-step workflow for developing Docker containerized apps [1]

2.4 Zero Trust Network Architecture

Traditional network security generally focused on a perimeter-based model, implementing the majority of security systems at the network edge. This edge was typically secured with firewalls and included Intrusion Prevention/Intrusion Detection systems (Kerman 2020). Once inside the network edge however trust was broadly assumed, resulting in poor, if any, security systems policing and protecting the internal network environment. Zero Trust Network Architecture (ZTA, or ZTNA), exists as a set of security principals for securing networks. Originally proposed in (Kindervag, 2010), ZTA assumes that implicit trust itself is a vulnerability in the network environment and seeks to eliminate the risks posed by malicious actors, internal or external, abusing this implicit trust. This is achieved by removing the concept of a trusted internal network and untrusted external network, rather instead assuming all networks are untrusted and ensuring that devices and users within a network operated from a least permission model. This was to be further augmented by the visibility and auditing

[1] https://docs.microsoft.com/en-us/dotnet/architecture/microservices/docker-application-development-process/docker-app-development-workflow)

of all activities taking place within a network. Kindervag proposed three key concepts for ZTA (Kindervag, 2010): (1) Ensure that all resources are accessed securely regardless of location (2) Adopt a least privilege strategy and strictly enforce access control (3) Inspect and log all traffic. ZTA is often boiled down to the following mantra: Never trust, always verify.

3. Related Work

Much work has been undertaken in the areas of container security, including Docker, and separately Zero Trust Architecture. Both topics being well established and studied, however there have been relatively few papers discussing Zero Trust Architecture for containers. Indeed, those papers that do exist in the crossover between container security and ZTA are focused primarily on implementing the traditional Zero Trust Networking approach to containers rather than abstracting the rules of ZTA into a more comprehensive framework to consider and govern the entire surface area of a container deployment, or more specifically as it relates to this paper, the surface area of a Docker container deployment.

(Surantha 2020) provides arguments for the use of ZTA from a technical standpoint in securing a container deployment, without evolving the concept of ZTA beyond the network. (de Weeve 2020) provides a more in-depth look at technical details, strategy and considerations when using ZTA with containers and identifies ZTA as more than purely a technical concept, but a security approach. (Yasrab 2018) provides a review of some of the most common issues affecting Docker containers and suggests several technical mitigations to address specific issues without presenting a framework to be used in addressing and mitigating issues going forward. (Martin 2018) conducted an extensive study of container security with practical demonstration from the standpoint of what Docker recommends vs what in seen in the wild and being done by major Cloud Service Providers. While this is an excellent paper that provides the basis of a mechanism for assessing Docker from a security perspective, it is driven from a very technical focus and does not provide a complete framework for analysing Docker deployments. (Bui 2015) contends that with proper security measures and lockdown, such as a reduction of the Linux CAPs assigned to a container and implementation of SELinux or AppArmour, as well as the removal of root use and privileged mode from containers, that containers could be considered relatively secure. Bui however does not present any framework or methodology that can be used to analyse a Docker deployment. (Rose 2019) provides a more comprehensive abstraction of ZTA, broadly arguing that ZTA can be used as a tool for strategic thinking, focusing much more on the concepts of trust and how this can be established rather than specific technical implementations.

4. Problem

In a 2020 Cyberthreat Defence Report published by CyberEdge, containers were listed as the number 1 weakest link in the security landscape. Containers are subject to bugs and vulnerabilities and therefore open to abuse. (Sultan 2019) states that containers are less secure than traditional VMs, and notes that this relative insecurity, or perceived insecurity, of containers remains one of the largest barriers to the ongoing adoption of containers. What makes Docker in particular an interesting target in the world of containers is threefold. First, we argue that the default state of a Docker deployment does not apply security best practices. Second, common real-world misconfigurations of Docker deployments increase the vulnerability and attack surface of a given deployment. Finally, Dockers large uptake by the developer community to assist in rapid development and deployment has resulted in the bypass of mature deployment processes. These factors have all combined to escalate the threat posed by Docker when deployed without the use of a methodology, or framework, for considering its security. In an article published on BlackHat, a leading security event series, Anthony Bettini, founder of Flawcheck, dedicates an entire section to vulnerabilities not necessarily needing a Common Vulnerabilities and Exposures (CVE) or code-based exploitation but rather those which attackers can take advantage of due to ”implementation weaknesses (ultimately, design imperfections)” (Bettini 2015). This is further borne out by several other articles describing some of these real-world exploits or proving out the issue directly (Simonovich 2019), (Higashi 2018).

A similar investigation was conducted in 2019, that detected over 20,000 Docker instances, and a similar number of Kubernetes instances, via Shodan (Quist, 2019). Quist notes that while these 40,000 combined instances are not inherently vulnerable, they are at least demonstrating security misconfiguration in their deployments. The research did however discover a number of vulnerable databases and deployments leaking personal information. TrendMicro, ran a study to determine some real-world exploit scenarios against containers (Oliveira 2019). They discovered that malicious actors were using automated scanning to find exposed Docker APIs and

deploy cryptomining container images against them, as well as containers to run scans for other exposed hosts. An exposed Docker API is not the only issue, if we assume that a container has been access or exploited (via Docker API or vulnerable container), there are a number of subsequent possible attack vectors. For instance, (Hertz 2016) describes a number of attacks for escaping privileged containers or exploiting unprivileged containers. (Avrahami 2019) details how to abuse a specific CVE, CVE-2019-5736 to break out of Docker, (Sharma 2020) provides a walk-through for abusing one of the Linux Capabilities granted by default to Docker images to break out of a container. (Wist 2020) found that "As many as 82% of certified images [on DockerHub] contain at least either one high or critical vulnerability".

5. Methodology

5.1 Test Environment, Version Assumptions and Setup

We used CentOS as the Host OS for Docker. CentOS 7 was installed using the Net Installer CD, using mirror.centos.org/centos/7/os/x86_64 as the install source to ensure the latest release is obtained. As of 29 October 2020, the latest release version of Docker is 19.0.3. Docker was installed using the first set of steps presented on the Docker for CentOS install page 13. The VM was then powered down and a snapshot taking for quick reversion of state as needed.

5.2 Empirical Approach

There are a number of well-known issues with Docker containers which will be investigated in Section 6, Docker Security Analysis. These issues are composed of both weaknesses in the default security posture of a Docker deployment, as well as issues introduced through misconfigurations. We will investigate, either by practical demonstration or by reference, examples of these issues.

5.3 Theoretical Approach

Not all of the issues with Docker can be easily demonstrated for two reasons. Firstly, they either have not been discovered yet, or secondly, they have been patched out of the latest release of Docker, which this paper will use to establish a reasonable baseline of a Docker deployment. However, by studying past vulnerabilities and how they have exploited Docker, we can infer reasonable steps that may be taken to lockdown elements favoured for exploit and help mitigate the risk of future exploits.

6. Docker Security Analysis

This section will present some of the more common issues affecting Docker default deployments, however this does not claim to be a complete list of issues. Results will be analysed later to draft the proposed framework 'Zero Trust Container Architecture'. It should be noted, that rather than focus on specific attacks as some other papers have done, this paper will attempt to separate the components of a Docker deployment into 'trust zones', i.e., to consider the elements of a Docker deployment with respect to the trust, implicit or otherwise, that can be used to exploit the system, rather than to focus first on the attack and then the affected element. Decisions on what to investigate here were informed by the literature review.

6.1 Docker Service & RemoteAPI:

Misconfiguration of the Docker RemoteAPI/Service allowing public, unauthenticated access provide easy access for attackers to compromise the Docker deployment.

6.2 Users and the Docker Group:

Users are often the weakest element in any system. Docker is no exception and further aggravates this by treating all users as root allowing malicious, or negligent users to easily compromise a system.

6.3 Container Networking:

Docker's flat networking approach by default opens up containers to easy attack from other malicious containers. Several basic ICMP and TCP/IP attacks can be launched, that can be very difficult to detect inside a Docker network leading to silent and invasive compromise.

6.4 Container Resources:

Containers share the Host OS's available resources. By default, no resource usage limitations are placed on containers. This can be easily abused by malicious containers to perform Denial of Service, or potentially theoretical entropy attacks against legitimate containers.

6.5 Linux Capabilities:

Capabilities are a permissions subsystem in use by Linux and used with Docker to delegate permissions to containers. These can be abused to escape a container and compromise the Docker Service or the host directly.

6.6 Leaky Secrets:

Secrets, that is usernames and passwords, or API token, to access other application and third-party services, are often passed into containers as environmental variables. Should a container, or deployment be compromised, these secrets may lead to additional resources being compromised.

6.7 'Privileged' Mode:

Privileged mode grants a container all permissions on the Host OS. Should a privileged container be compromised, the Host OS, be default, should be considered compromised.

6.8 Docker vulnerabilities and Container Houdini:

As with any system, Docker suffers from vulnerabilities, both its own and those affecting related systems. These vulnerabilities have been identified as potential causes of exploit, and indeed have been used in real-world exploits as well.

6.9 Container Images:

Container images are point in time, static files that contain executable code to bootstrap an OS image, often containing libraries to support development, or entire applications. These images are uploaded to either private or public repositories for use by internal or external users. These images, by accident or indeed malicious design, may contain vulnerabilities or malware. The older the image, the more likely that several serious vulnerabilities exist.

7. Proposed Solution: Zero Trust Container Architecture (ZTCA)

In general, many of the issues identified can be described as the security reality of a default deployment of the Docker engine, or due to misconfigurations of Docker. That is, the implicit trust that users of Docker are delegating to the default install state of the Docker engine. This is why ZTA, an architectural framework that speaks specifically about trust in systems and how to avoid security issues arising from this trust, is suitable to mitigate these issues. Similar to (Rose 2019), we will create a framework by abstracting the principals of traditional ZTA, with a focus on challenging this implicit trust delegated to default installs of Docker. To be clear, there is nothing wrong with trust, or delegating trust within a system, however, this trust must be deliberate, explicit, and scoped appropriately. A key rule in ZTA is that every request, no matter where it is made from, should be secured, or verified. In traditional ZTA terminology this speaks to requests not only from 'north/south' sources being secured (from outside into a network) but also 'east/west' requests (within the same network) being secured. ZTA moves the boundary of trust from the network to the users and the endpoints (Rose 2019). If we refocus, or abstract, the boundary of ZTA to the Docker Service itself, this brings a completely different lens into what constitutes a request, as well as the meaning of securing said requests. For example, a user interacting with the Docker service becomes an untrusted communication. The Docker service or host OS communicating with a container (for instance passing secrets to a container) becomes an untrusted communication. The Docker service communicating with a container registry becomes an untrusted communication. Indeed, a similar argument for abstraction, albeit with a different focus is, made by (Rose 2019) "to take this one step further, the word "resource" can be substituted for "data" so that ZT and ZTA are about resource access and not just data access." By re-purposing this idea of a 'resource' in ZTA as the individual elements interacting with the Docker service, as well as considering the service itself, and combining this with a lens of trust, it is possible to create a set of 'maxims' for scoping, and analysing, the trust boundaries of a Docker deployment (see Figure 4).

This list of maxims allows for the design of a category/consideration matrix where elements of a Docker deployment, or 'Categories', are attributed a non-finite list of trust items, or 'Considerations'. 'Considerations' are suggested items to be considered when the related 'Category' is used with the framework, as such they are not mandatory, as they are not necessarily relevant to every scenario. Users of the framework may also add additional considerations to reflect their particular needs (e.g., alignment with another framework, standard or internal business rules).

Trust not the Host: The host running the Docker engine itself is a prime target for exploit. Docker does not exist in a vacuum and indeed the host it is running on has its own set of vulnerabilities (in particular kernel vulnerabilities as these increase the risk posed by compromised containers as they share the kernel with the Host) and security misconfigurations that allow it to be exploited and compromised independently of Docker. Securing the host is a crucial part of ensuring a Docker deployment remains secure.

Trust not the User: By the terms of a default install of Docker, users granted access to the Docker group are effectively granted root access to the Host. This mandates much more consideration when granting users access to avoid unintentionally elevating permissions. Separately, users within a Docker container map to users on the host. By default, the lack of user namespacing means that running as root inside a container is effectively running as root on the Host (limited only by Container assigned CAPs on the root host).

Trust not the Service: The Docker service runs with root permissions which can lead to a number of the compromises discussed previously in Section 6. Further, misconfigurations in presenting the Docker service publicly can lead to serious compromise with little to no effort on the part of the attacker.

Trust not the Image: Docker images have been identified as a major concern with respect to trust. Both in terms of inherent vulnerabilities with images, or with outright malicious images. Further, a failure of process to frequent update images to the latest version could expose what is otherwise a secure system.

Trust not the Container: By default, a Docker container itself presents a number of issues when it comes to implicit trust. We are trusting the lack of namespacing which allows container root to be restricted only by the CAPs applied to a container. We are trusting the Container not to abuse or overuse system resources. We are trusting a Container with secrets passed that may expose additional services to compromise. We are potentially trusting the container with privileged execution, grating it all power on the system. We are trusting it to share the kernel with the Host. All of these items must be properly reviewed and considered.

Trust not the Network: By default, the flat network topology inside a single Host node allows for all Containers to communicate with each other, an implicit trust. Basic principles of ZTA dictate the need for securing east/west traffic, as well as north/south. As such this default network topology stands in violation of some of the fundamental rules of ZTA. Further, by default, there is no separation of concerns as with more traditional networks, which are often layered into distinct zones (LAN, DMZ, etc.) to limit communications and exposure, between services to only that which is necessary.

Trust not the Silence: This describes the absence of logs, alerts and generally visibility, termed in this work simply as the 'Silence'. By default, Docker and the OS both provide some level of logging, however by default this is broadly opaque and decentralised. This does not meet the more comprehensive, auditable logging advised by ZTA. Docker logs are broadly concerned with the execution and state of a container, more specifically the 'STDOUT' and 'STDERR' logs, or with collecting metrics, no logging takes place with regards to network packets, traffic flow or inter-container communication. Without appropriate logging it is extremely difficult to understand what is actually taking place, to evaluate the situation and to ensure that trust has been delegated properly and is not abused.

Figure 4: ZTCA Maxims

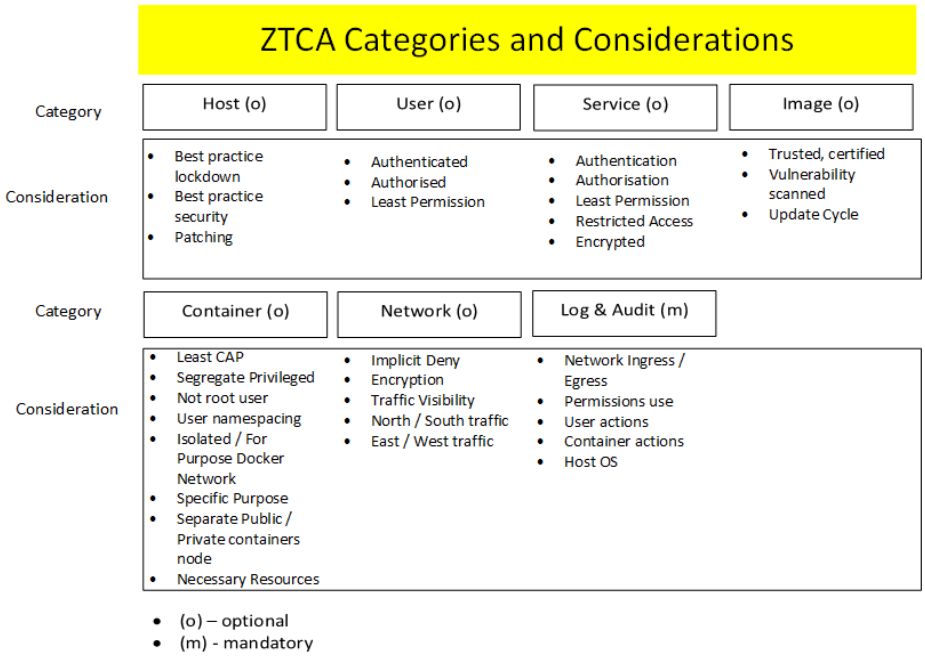

Figure 5: Categories and Considerations Matrix

The 'Categories and Considerations' matrix (Figure 5) serves as the input into the framework. Individual, or multiple, optional categories from the 'Categories and Considerations' matrix are passed to the framework for consideration. In order to simplify the use of the framework it is not advised to pass any more than 2 optional categories at a time. 'Log & Audit' category, which is mandatory, is to always be included.

7.1 ZTCA Framework

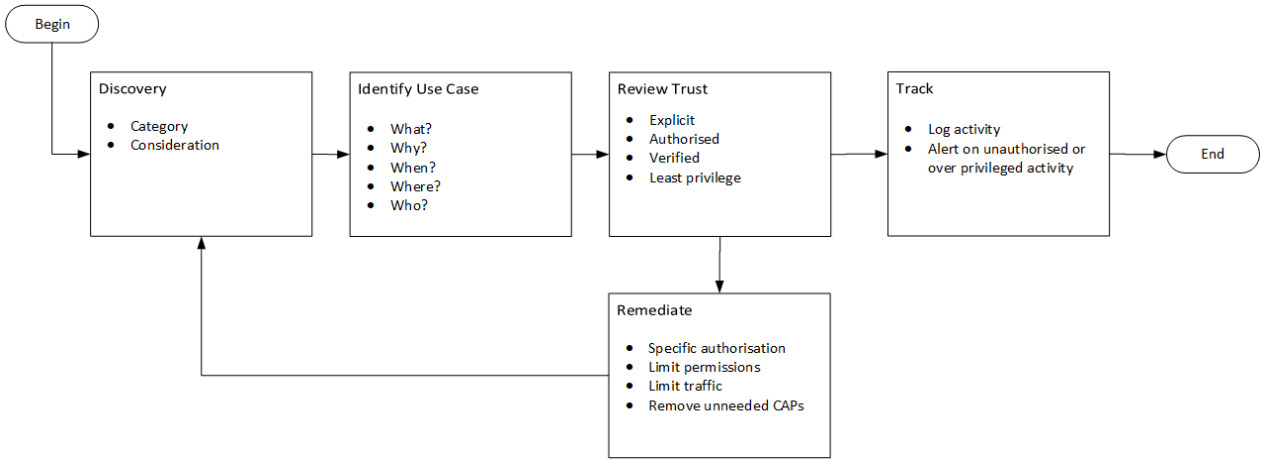

Figure 6a: ZTCA Framework

The framework (Figure 6a and 6b) will not attempt to be prescriptive, in that it will not specifically limit use cases (e.g. "never publicly expose the Docker Remote API") but will instead attempt to frame the thought process surrounding these use cases with respect to trust in such a manner as to ensure they are implemented in a secure fashion (e.g. "ensure the Docker RemoteAPI accepts only encrypted traffic, from only expected sources, from only authorised users"). As such, the framework does not output technical steps, as these may be specific to the users adopting the framework, for instance different firewall vendors mandate different technical steps. The framework is intended to be used in two ways. First, the framework can be used with a specific question, or use case in mind, for example 'grant a user access to the RemoteAPI', which the framework then can refine through a scoped lens of trust related to the categories passed, review the existing deployment, and propose remediation's to align the deployment with the refined use case. Secondly, the framework can be used to generate novel questions by passing category(s) without a specific question, or use case, in mind, and allowing users to consider their deployments through the lens presented by the framework and category(s) passed in order to generate novel questions. The framework operates by initially taking in category(s) which are used to

help scope and understand the question of trust being asked, then passing to subsequent functions which refine the exact use case in question, identify and define the appropriate trust required and contrast this with the existing trust in a deployment, before moving to a remediation step and finally the implementation of appropriate technical measurements to track the application of this trust. There are several decision gates included in the flow of the framework which can be used to iterate on sections which do not initially produce a refined output, or to review the process in light of remediation findings to ensure the framework has appropriately addressed all aspects of trust in question. The framework is designed to be used in conjunction with, not replace, other frameworks concerned with other elements of security (for instance frameworks for designing and securing a corporate network).

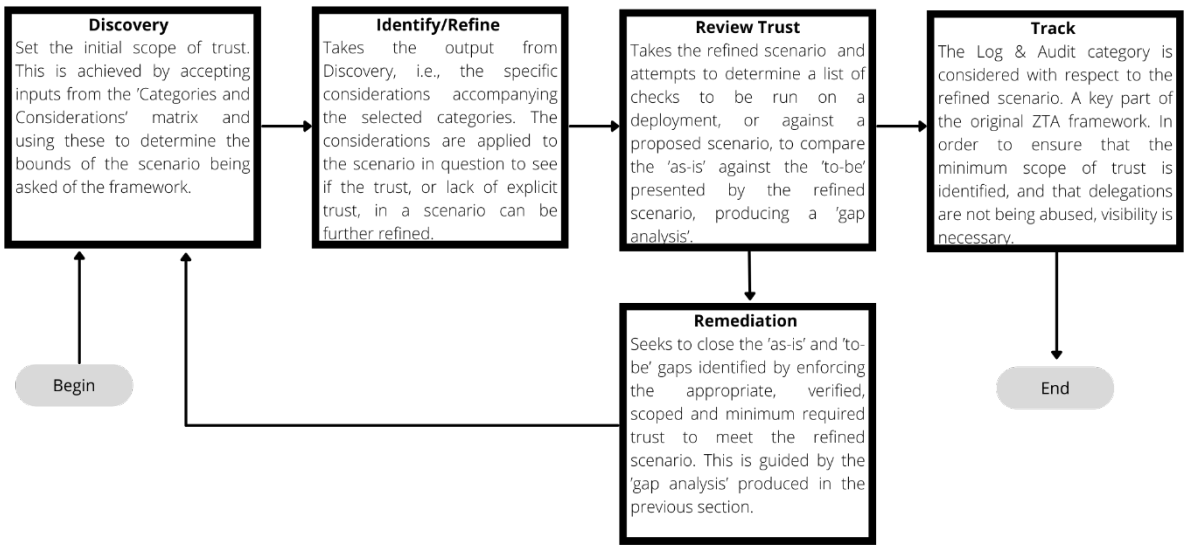

Figure 6b: ZTCA Framework Explained

8. Validation

Using the framework proposed, we analysed several scenarios derived from the issues identified in Section 6 and validated that the use of ZTCA leads to a successful mitigation, however it is not possible to address all attack scenarios demonstrated in Section 6, due to complexity and time required to implement and demonstrate mitigation of all in detail, however the following scenarios were investigated in detail:

- Test Scenario 1: Grant user access to Remote API (see Section 6.1)
- Test Scenario 2: Prevent Container Network Attack (see Section 6.3)
- Test Scenario 3: Review a Docker image (see Section 6.9)

Additionally, the following scenarios were walked through at a high level, that is the steps the framework suggests but without actually applying the mitigation:

- Test Scenario 4: Review an existing Container; a novel use case
- Test Scenario 5: Deploy a Privileged Container; as seen in Section 6.7 In beginning validation, it is worth noting that while there are various products on the market from a number of vendors to assist in securing aspects of Docker containers, or securing the network or Host OS, this paper will avoid providing implementation specifics for these products as this is outside the scope of this paper, but may reference concepts or utilise certain products as an example of how they can be applied to achieve compliance with suggestions of this framework.

This validation work is detailed in https://bit.ly/3EPJXN0, but due to space constraints, it cannot be presented in this paper.

9. Conclusion

This paper focussed on three research questions: (1) What issues currently affect a default deployment of Docker? (2) What additional common misconfigurations affect a Docker deployment? (3) Can ZTA, or an abstraction thereof be applied to help secure a Docker deployment? We have shown that Docker contains a

number of security concerns. From combination of both design issues and vulnerabilities, but also common misconfigurations that sacrifice security for convenience. Further, several security issues arise from the difference between a system espoused and a system in use. By distilling the principals of Zero Trust Architecture and leveraging these to create a ZTCA framework, this paper has proposed a method for analysing and determining the path to securing a Docker deployment. Validation has shown, when applying the framework to areas of concern as identified in Section 6, the proposed framework can be used to successfully analyse and recommend effective steps to ensure the security of a Docker deployment. The framework proposed, by virtue of the scope of possibilities, is complicated and assumes a level of proficiency with Docker as well as the ability to identify, analyse and implement additional appropriate security technologies. One of the main aims for future research should either be around a simplification of the security by design process, though this may rely on Docker Inc. developers themselves, or an abstraction layer for the Docker Engine which enforces the principals of security and least privilege by default.

References

Avrahami, Y. (2019). Breaking out of docker via runc – CVE-2019-5736. https://unit42.paloaltonetworks.com /breaking-docker-via-runc-explaining-cve-2019-5736/. Accessed: 2022-01-06.

Bettini, A. (2015). Vulnerability exploitation in docker container environments. https://www.blackhat.com/docs/eu-15/materials/eu-15-Bettini-Vulnerability-Exploitation-In-DockerContainer-Environments-wp.pdf.

Bui, T. (2015). Analysis of docker security. https://arxiv.org/pdf/1501.02967.pdf. Accessed: 2022-01-06.

de Weeve, C. and Andreou, M. (2020). Zero Trust Network Security Model in containerized environments. PhD thesis, University of Amsterdam.

Graber, S. (2016). LXD 2.0: Introduction to LXD [1/12]. https://stgraber.org/2016/03/11/lxd-2-0-introduction-to-lxd-112/. Accessed: 2022-01-06.

Hertz, J. (2016). Abusing privileged and unprivileged linux containers. Whitepaper, NCC Group, 48.

Higashi, M., Kumar, G., and Chiodi, M. (2018). Lessons from the cryptojacking attack at tesla. https://redlock.io/blog/cryptojacking-tesla. Accessed: 2022-01-06.

Kerman, A., Borchert, O., Rose, S., Division, Eileen, and Tan, A. (2020). Implementing a zero-trust architecture. Technical report, NIST

Kindervag, J. (2010). No more chewy centers: Introducing the zero-trust model of information security. Forrester Research.

Martin, A., Raponi, S., Combe, T., and Di Pietro, R. (2018). Docker ecosystem – vulnerability analysis. Nokia Bell Labs, 122

Oliveira, A. (2019). Infected containers target docker via exposed APIs. https://www.trendmicro.com/en_us/research/19/e/infected-cryptocurrency-mining-containers-target-docker-hosts-with-exposed-apis-use-shodan-to-find-additional-victims.html. Accessed: 2022-01-06

Quist, N. (2019). Misconfigured and exposed: Container services. https://unit42.paloaltonetworks.com/misconfigured-and-exposed-container-services/. Accessed: 2022-01-06.

Rose, S., Borchert, O., Mitchell, S., and Connelly, S. (2019). Zero trust architecture. Technical report, National Institute of Standards and Technology.

Simonovich, V. and Nakar, O. (2019). Hundreds of vulnerable docker hosts exploited by cryptocurrency miners. https://www.imperva.com/blog/hundreds-of-vulnerable-docker-hosts-exploited-by-cryptocurrency-miners/. Accessed: 2022-01-06.

Sultan, S., Ahmad, I., and Dimitriou, T. (2019). Container security: Issues, challenges, and the road ahead. IEEE Access, 7:52976–52996

Surantha, N. and Ivan, F. (2020). Secure kubernetes networking design based on zero trust model: A case study of financial service. Master's thesis, Bina Nusantara University, Jakarta, Indonesia.

Yasrab, R. (2018). Mitigating docker security issues. https://arxiv.org/ftp/arxiv/papers/1804/1804.05039.pdf. Accessed: 2022-01-06.

Wist, K., Helsem, M., and Gligoroski, D. (2020). Vulnerability analysis of 2500 docker hub images