TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Mostafa Hadi
IVCM165607

MAKING THE SHIFT FROM DEVOPS TO DEVSECOPS AT DISTRIBUSION TECHNOLOGIES GMBH

Master's thesis

Supervisor: Hayretdin Bahşi
PhD

Tallinn 2019

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Mostafa Hadi

01.04.2019

Abstract

The purpose of this thesis is to conduct a practical study on integrating all necessary security measurements, which in the DevOps era can easily be left out of the picture, as a sticky part into both application and infrastructure development in Distribusion Technologies GmbH. This goal would not fulfil without educating involve people with better security practices. Like any other start-up company, threat landscape against Distribusion is evolving with both new vulnerabilities discover daily and lack of security knowledge among both developer and DevOps teams that are responsible for the fast pace of developments in the company's products. These two main factors increase the possibility of neglecting security measurements, which might put the whole business into risk.

This thesis evolves by introducing proper metrics for having a reliable monitoring system that can represent the author's contribution in each stage of improvement, by recording both before and after states. In the third chapter chosen research methodology for conducting this thesis explained. In the fourth chapter, securing infrastructure have been discussed in two stages, re-architecting platform toward zero trust networking architecture (ZTN) alongside by hardening all infrastructure assets and applying, the concept of Infrastructure as Code (IaC) to the new infrastructure.

In the fifth chapter, author integrated missed security measurements to the development cycle by taking advantage of different types of approaches for having better development and deployment procedures, such as shifting security checks to the early stages of development, static code analysis against codes on each deployment and hardening containers that company's applications are running on them.

This thesis is written in English and is seventy-one pages long, including six chapters, fifteen figures and eight tables.

Annotatsioon

Üleminek DevOpsilt DevSecOpsile Distribusion Technologies GmbH-s

Selle hüpoteesi eesmärk on koostada praktilist õpet kasutades kõiki vajalikke kaitsemeetmeid ,mis DevOpsi programmis on lihtsasti välja jäetav, oluline osa mälemas applikatsioonis ja infrastruktuuti arengus on Distribusion Technologies GmBH. Vajadus ütleb, et eesmärgiks on kaasata haritud inimeste osa võtmist kaitsepraktikas. Nagu iga teine start-up ettevõte kasutab maastiku vastavalt Distributionile kaasates mõlema haavatavust, igapäevaselt avastatakse puudusi teadmistest turvalisusest nii arendajad kui DevOpsi tiim on vastutavad ettevõtte toodete kiire arenemise eest. Need kaks faktorit suurendavad turvameetmete eiramist, mis võib terve ettevõtte seada turvaohtu.

Hüpotees areneb tutvustades erinevaid meetmeid, et saada usaldusvaarne monitoorimissüsteem, mis esindaks autori panust igasse arenguetappi, salvestamine nii enne kui ka pärast olekut. Kolmandas peatükis selgitati selle uuringu läbiviimise metoodikat. Neljandas peatükis, millega tagatakse infrastruktuuri arutamine kahes etapis, infrastruktuuri ümberehitamine null-usaldusvõrgustiku arhitektuuriks (ZTN) together with the tightening of all infrastructure resources and the implementation of infrastructure as a new infrastructure code (IaC). Viiendas peatükis lõi autor vastamata jäänud turvamõõtmised arendus tsüklisse, kasutades erinevaid lähenemisviise, et saada paremaid arendus- ja juurutamise protseduur, näiteks muuta nihke vasakule, staatilise koodi analüüsi iga kasutuselevõtu- ja karastus konteinerite koodide vastu ettevõtte rakendused neid kasutavad.

Hüpotees on kirjutatud inglise keeles, mis koosneb seitsekümmend üks leheküljest pikk, sh kuus peatükki, viisteist joonised ja kaheksa tabelid.

List of abbreviations and terms

2FA	Two-Factor Authentication
ALB	Application Load Balancer
API	Application Program Interface
AWS	Amazon Web Service
CI	Continuous Integration
CIA	Confidentiality/Integrity/Availability
CI/CD	Continuous Integration and Continuous Delivery
DevOps	Development and Operation
DevSecOps	Development, Security and Operation
DOS	Denial Of Service
DTB	Double Trust Boundary
EC2	Elastic Compute Cloud
EFS	Elastic File System
GDPR	General Data Protection Regulation
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IaC	Infrastructure as Code
IT	Information Technology
QA	Quality Assurance
NACL	Network Access List
NIST	National Institute of Standards and Technology
NLB	Network Load Balancer
OWASP	Open Web Application Security Project
S3	Simple Storage Service
SCM	Source Control Management
SDLC	Software Development Life Cycle
SG	Security Group
SSH	Secure Shell
SSL	Secure Socket Layer
STB	Single Trust Boundary

RDS	Relational Database Service
VPC	Virtual Private Cloud
VPN	Virtual Private Network
ZTB	Zero Trust Boundary
ZTN	Zero Trust Networking

Table of contents

List of figures

List of tables

1 Introduction

On the fast pace of software development and deployment in DevOps era, security often gets left out of the picture because usually, they threat as people who are slower than the process, Distribusion Technologies GmbH was not an exception. To back an agile process with security author plan to integrate all necessary security tools, measurements and policies into DNA of Distribusion Technologies GmbH. This marriage between security and business development should not turn to a bottleneck gate and slowing down the whole cycle. Integrating security with development and operation called as DevSecOps. DevSecOps adaptation is a must to do for all small organisation if they want to keep their business in the market. Because of Distribusion Technologies GmbH business model, moving from DevOps to DevSecOps can serve an excellent service to a broad spectrum of companies and their customers. Thanks to the Distribusion's API, almost all bus and train operators can connect with travel retailers on a worldwide scale and sell their tickets to the end customers. Lack of security experts, security measurements and security knowledge in the Distribusion Technologies GmbH, had been exposed plenty of security flaws to the company and they had potential to put the whole business into risk; Most of these security flaws have been addressed in the course of this study.

This chapter provides an overview of this thesis. The first section gives a brief introduction to the definitions used in this thesis and examines the problem and objectives; Followed by research questions, the importance of this topic to work on and limitations of the study.

1.1 Problem

Two main security problems have been identified in the Distribusion Technologies GmbH which consist of application security and platform security. Also, each of these two categories separated into some subcategories to address better.

Here is a list of security flaws which has been spotted in application development and deployment.

- There was a lack of automated security checks on codes that get written by developers who have insufficient knowledge in the cybersecurity area.
- Exposing all security credentials in plain text on all application jobs
- Lack of least privilege policy for users access.
- Applications were utilising vulnerable libraries, deploying codes on un-authorised docker images.
- Absence of blue/green deployment in production application deployments.
- Absence of proper security checks against code and their respective third-party libraries

Regarding the platform security here is a list of identified security problem in infrastructure.

- Applications and confidential data were hosted on an infrastructure that has been architected without security mindset.
- There was a risk of direct human touch on mission-critical infrastructure by manually maintaining it.
- Fragile infrastructure against unexpected traffics.
- The undocumented number of unnecessary services and API endpoints had public access from the internet.
- There was a lack of proper level of isolation between environments, servers and services.

By the end of this thesis, the author takes all necessary actions to address all the security flaws mentioned above.

1.2 Motivation

The author firmly believes that taking security to the utmost seriousness in all IT start-up companies that already adapted to DevOps methodology is a must to do if they want to stay in their business market, Distribusion Technologies GmbH is not an exception. Adaptation to DevOps means embracing rapid deployments for introducing new features on a dynamic environment which has the potential, to leave security measurement neglected. This potential risk can get even higher in start-up companies since there is no dedicated security expert in their IT department. By being aware of the fact that all those classic and manual approaches toward security have reached to limit of their effectiveness and they cannot get utilised with the fast-moving nature of DevOps culture, and these security assessments and assurance should use in automated manners. Have plenty of insecure services running on a fragile and unprotected infrastructure with complex networking in the background has been the main concerning issues that turn to a motivation factor for Implementing an automated security approach for putting security frame around all DevOps related tasks in Distribusion Technologies GmbH were the main driving factor for conducting this study. As already mentioned, achieving this goal, successful DevSecOps adaptation in Distribusion, not only can protect Distribusion assets from different threat factors but also it can protect all its plus forty partners' API and their customers, which have been integrated with Distribusion's API. [1]

1.3 Research Questions

The author firmly believes that all of those organisations which embraced fully functional DevOps should incorporate security into their DevOps methodology; however, this can be so time-consuming and hard to achieve fully but the outcome of this marriage worth it. Through this practical study, the aim is to integrate security into DevOps methodology on the following research questions:

1. Why have accurate monitoring and alerting systems are the essential step for moving from DevOps to DevSecOps?
2. Why there is a high chance that security left out of the picture when start-up companies launch a new product?

3. Why is automating security checks an undeniable fact for adapting with DevSecOps?
4. How can isolation in-depth serve security into both application and platform?
5. Why is it recommended to conduct code security checks in the early stages of the software development lifecycle?

1.4 Related works

According to my research so far no study has been conducted on making the shift from DevOps to DevSecOps in a start-up company as a real-world case. However, there are some research studies which cover some parts of this topic, none of them had been done to this extent an on a real-world case; Previous studies [2] [3] [4] [12] [27] [28] mostly had been conducted in lab environments.

In both pieces of researches, "Exploiting DevOps practices for dependable and secure continuous delivery pipelines" [2] and "Vulnerabilities in Continuous Delivery Pipelines?" [27], authors conducted some of National Institute of Standards and Technology (NIST) and Open Web Application Security Project (OWASP) security checks against codes and their third-party dependencies and base on matrix table evaluate possible impacts of potential vulnerabilities on confidentiality, integrity and availability (CIA), on the final application products. Finally, they have mentioned the importance of security awareness among IT team since they have spotted a gap there and integrating security into the Continous Deployment (CD) pipeline. In the "Containerization of telco cloud applications" [3], most essential ways for securing containers have been mentioned, and also some of the security outcomes of moving applications to containers have been highlighted such as increasing level of isolation among those applications which run inside containers also ease of scalability for having better availability. Another relevant study, titled as DevOps' Shift-Left in Practice: An Industrial Case of Application [12], mainly emphasize on importance of moving security checks to the left side of application development and deployment pipeline in order to have better code by conducting frequent security checks and gathering feedbacks and reporting immediate result of each deployment.

Evaluation of Infrastructure as a Code for Enterprise Automation [28], is another study which mainly emphasises on the importance of four main deployment tools, Ansible, Chef, Puppet and SaltStack in the DevOps era to have a reliable way of maintaining and managing infrastructure.

As it has been mentioned already, this study has covered a broad and necessary spectrum of criteria which are required to have an improvement in their security aspects to have a better adaptation to DevSecOps in an organisation.

1.5 Contribution

Improving security in the Distribusion Technologies GmbH is the backbone of the author's contributions. This security improvement has happened into two central areas application development/deployment, and platform security. For having a secure development/deployment application pipeline and platform, the author utilised plenty of security tools while provisioning and maintaining new assets and deploying new code. Moreover, the outcome and lesson learned of this practical study which will be available on the Authors Git repository can be a useful resource for other start-up companies that already embraced DevOps but for any reason they still do not have any security expert in their own IT team.

Improving security in a start-up company, as a real-world case study, is the main novelty of this study; As none of the previous studies have covered a complete shift from DevOps to the DevSecOps by improving security on both platform and application development and deployment.

1.6 Limitations

There was some number of main limitations that could not be controlled by the author while conducting this practical study. The author has been put his best to minimise the negative impacts that might cause by these limitations to the final result of this study. Listed below are the barriers and reasons why they have been encountered in this study.

1.6.1 The time frame and lack of human resource

However this study has been conducted by the only DevOps engineer person, author, in a timeframe of twelve months in Distribusion Technologies GmbH, but because of vast number of elements that needed to touch, improve or even replace this time frame was short for conducting this study and there are some elements left that can develop as a future work.

1.6.2 Dealing with out of hand elements

For applying security into application's pipeline, there were some limitations such as the limitation, on upgrading all the vulnerable libraries into their secure version; this was one of the main blocking factors and took plenty of author's time. Upgrading vulnerable libraries to a secure version could impair the availability of application as one of the primary CIA triads and this is was not something that the author could do it without having developers and business people involvement. Having developers involved in this process was needed for verifying all application's functionality after upgrading each third-party library.

1.7 Overview of the Thesis

This chapter provides a short introduction to the thesis topic, background to the problem to be studied, relevance, objective, limitations. Chapter 2 covers the Methodology that has been used for conducting this study.

Chapter 3 represent the current status of the monitoring and metrics and how those metrics and monitoring will get more productive and reliable by cooperation between developers and Author. Having this phase as the first phase is vital for recording the current state and future state after applying each stage for repressing the outcome of each phase with clear graphs. Establishing an accurate alerting system base on the metrics that monitoring systems collect from different assets is the next step that will cover in this chapter.

Chapter 4 covers the process of shifting security to the left in the application development and deployment cycle. In this chapter, we inject two types of security checks against both application codes and container base images to make sure that vulnerable codes or images do not push to production.

Chapter 5 covers three main areas, re-architecting infrastructure base on Zero Trust Network (ZTN), moving infrastructure to code and introducing elasticity to the platform. In the re-architecting infrastructure base on Zero Trust Network (ZTN), existent attack surfaces on mission-critical assets are going to minimise as much as possible. Afterwards author has proceeded with reducing human touch on critical parts of the infrastructure to immune them of human failures; this will be done by adapting our infrastructure to code, and finally, we going to utilise some solutions for using elasticity features that public clouds like Amazon provide for their clients.

Chapter 6 declares the study results, lesson learned and possible future works that can build upon this work.

2 Methodology

As this study based on improving security in an IT start-up company and all the finding have been applied to this study case, the author believes that action research is an appropriate way of conducting this study. Each improvement step of this study has been conducted through the following three steps: [30]

1. Discovering those areas that security missed out of it by establishing a reliable monitoring and alerting system.
2. Taking measurable actions to either mitigate or eliminate the negative impact of spotted security issues from both applications and platform.
3. Evaluate the outcome of each improvement action by comparing before and after states of each phase.

Having plenty of applications, services and confidential data hosted on AWS public cloud, it make sense for the author to divide all corresponding assets into smaller parts to have a better understanding of elements that their security should improve; Distribusion's assets have been divided into three different but related components such as monitoring and alerting, application security and platform security.

Except for monitoring and alerting phase which has been implemented between March 2018 till May 2018; Security of other parts of these study, including application security and platform security, has been under improvement process from May 2018 till now May 2019.

2.1 Monitoring and alerting

Establishing a reliable monitoring solution for scraping different types of metrics from all in use assets, not only is the main cornerstone of this study for having a reliable validation method for recording different state of each phase but also plays a crucial role on giving clear transparency on functionality of all applied security measurements that author is planning to apply on Distribusion's assets. In this phase, we will install a

metric scraper on each assets depending on what kind of security and functional metrics we are looking for those specific assets. Running an accurate alerting system and rules base on the metrics that monitoring system collects from different assets is also another important thing that we need to utilise for establishing a successful DevSecOps in Distribusion Technologies GmbH.

2.2 Securing applications

Regarding the application security, some security measurements need to come into play on storing, writing and deploying codes to have an application that does not put the security of the organisation into risk. Following list had been taken as a step by step plan by the author to introduce better security into application and codes that get written by developers and deploy by deployment tools into production.

- Hosting written codes on a secure and reliable source code manager, such as GitHub and Git which can control the version of the codes that get pushed to the corresponding repository and allows to quickly roll back to the previous version of codes in case unwanted code get pushed to the repository.

- Running static code analyser against all newly written codes and third-party libraries that they use; This analyses should also run automatically on each application deployment. As soon as a vulnerability gets detected during application deployment, the deployment job should immediately be terminated and do not make it to production. Result of this type of failed deployment should announce to both developer and product manager, immediate feedback.

- Repetitive feedback to the respective developers and product manager on failed deployment can be a great source of truth for understanding why an application deployment has been failed. Integrating slack with Jenkins in the job deployment pipeline can fulfil this goal. Having detailed information on output result of running static code analyser against new application code on each new deployment will increase visibility on the background of the deployment pipelines to a decent level.

- There is no need to put Database and other confidential assets' credentials in plain text inside application deployment jobs or import them to the applications

as environment variables; This can increase the risk factor since it increases the attack surface in a large scale. To protect the secrets of getting exposed, they store and retrieve just from a centralised secret management system, called as HashiCorp vault. After utilising this centralised secret management tool, deployment jobs access to the vault server and retrieve the required secrets that are needed by the application.

- Spotting which applications are internet face and how many endpoints they are exposing to the outside world and also understanding all the potential values that application can officially accept from their end-users and what kind of malicious values can potentially get injected into these applications as valid inputs.

- Developers should get educated on writing clean and secure codes; Such as using git-secret on their machine to prevent them from pushing credentials into their codes and respective repository. Moving security checks to the early stage of the development pipeline can serve a significant impact on having secure products with the least amount of cost.

2.3 Securing platform

Alongside to securing development and deployment pipeline, the author is planning to put the same or even more amount of effort on securing the whole platform that all Distribusion's assets and application are hosted on. To have a clear understanding of those areas that we want to improve their security in this chapter, I have divided these areas into two main sections, moving toward Zero Trust Network security model and moving infrastructure to code.

- Re-architecting infrastructure according to the best security practices such as applying Zero Trust Networking architecture (ZTN) concept to its networking part.[21][29] By applying ZTN concept to the infrastructure, there will be precise and clear isolation among all involving assets which known as segmentation; This can decrease potential attack surface to the compromised asset in case of an attack.

    - After making sure that there is a right level of isolation between different servers author proceed this phase with hardening each single of servers;

this phase can be called as host security. In this phase, all applications, services and mainly running operating system on the server upgraded to their latest stable and secure version and all unnecessary running services on them should stop, and those services which need to stay running now have a better level of isolation around themselves. Also, in this stage one vulnerability scanner should run periodically against all mission-critical assets and provide a report about the potential threats that each server might have.

- In this phase, current ingress and egress traffic gets split into two types of traffics, internal and external on both sides. By doing so, we can be assured that confidential data or any un-necessary traffic do not get routed to the internet. Tools and technologies such as internal and external load balancers, internal and external domain names, re-writing traffic routing roles will be involved in this phase of risk management. By successfully applying this phase not only mission-critical assets will get protected but also since it decreases the number of internet face assets to the least amount of possible, in case of a functional or security incident forensic investigation for spotting the root cause can be much faster since our platform operates on internal and external manners and this means least amount of elements need to get investigated.

- Regarding the remotely accessible assets, there are some mitigation steps that the author is planning to apply to the corresponding asset. These steps are as following:

    - No SSH to the instances are allowed and if in urgent cases SSH is needed it should be evident in the log who did, what and when. Logging these three types of data in the log can be an excellent source for forensic investigation in case security incidents happen. As a general rule, remote access services should not use their default ports, and SSH service is not an exception.

    - External access from the internet is better to get eliminated on mission-critical assets as much as possible. The author is

planning to introduce a VPN server for the company and move all these types of assets behind it. Also for security reason, this is important that company VPN servers do not use any self-signed certification.

- Possible human errors that might cause because of direct human touch on mission-critical assets is a good driving factor for cutting all possible human contact on production assets and make all those manual processes as automated as possible. Maintaining assets in an automated manner by taking advantage of infrastructure as code concept, not only provide us with the same results on all environments, development, staging and production but also opposite to human code does not make a mistake and do precisely the thing as it supposes to do. Since all Distribusion's infrastructure and applications are hosted on a public cloud, Amazon Web Service (AWS), implementing a fully automated scaling solution for all running servers base on the real-time traffic load not only has a financial advantage for the whole business but also give us assurance on being protected against DOS attacks. Also, having a transparent and traceable source of verification, such as Ansible playbook and Terraform templates, for checking what version of service is running where and what type of resource operating and for what can serve a great value for identifying which running service need to patch or upgrade. [5]

For the sake of improving readability, each of phase as mentioned earlier will cover in a different chapter, and it will follow the following format:

- Overview of what the author is trying to accomplish
- What was wrong and how bad was it
- How did the author or the team try to resolve the issue
- What was the outcome
- How the outcome will get verified

3 Monitoring and alerting

Collecting performance and security metrics from application and infrastructure assets is a must to do when an organisation is trying to integrate security into its DevOps culture. This will help to have a reliable record of asset's behaviour and by using these collected metrics, set up an alerting mechanism base on the expected behaviour of critical assets. Moreover, having reliable records of metrics can also be a good source of truth for investigating the root cause of any type of incidents that might happen at any time. [6] Plus, establishing a good visualization of collected metrics not only can give a clear idea to the whole IT team on how the entire business is rolling but also it can be a useful recording tool on this practical academic research for representing before and after states of those areas that their securities are going to improve.

3.1 Application security monitoring

As functional incidents are way more visible than security incidents, in case anything goes down because of it functionality problem, there is a high chance that everyone notices it immediately, but most of the security fires can easily stay unnoticed. The reason that security problems in the code and the running services are quite, caused by lack of security team in the start-up companies such Distribusion Technologies GmbH, and the almost all of testing processes such as Quality Assurance(QA) is around functionality test of new features on the product. Always there is a possibility to have some vulnerabilities lurking inside the code without coming to notice, and they can remain hidden for a while in the code until an actual attack happen.

3.1.1 Tracking requests

One of the basic thing that can be done about all invisible security fires is to increase transparency on how and where packets coming from and going to. Implementing a unique sticky Identification code (ID) on all incoming requests can provide good traceability on network traffics; This can be a good source of truth to refer in case a security incident accrue. In Distribusion Technologies GmbH, the same concept of

identifier has been implemented for all incoming request packets on the main gateway of each environment, HAProxy servers. As it can be seen in figure 11 HAProxy servers, sit on the edge of each environment, and they load balance the incoming traffic between each server of the respective environments. As HAProxy servers are the only gateway to each environment, not only we can monitor the traffic of each environment by them, but also we can manipulate the traffic on them according to our will. As an example, by using a module of HAProxy called as LUA, HAProxy-LUA, we have added a unique sticky identification code (ID) called x_request_id to all incoming request. This identification field is our exclusive way to track down all requests through our system and correlate with all other logs that we are getting in our system. After implementing X_request_id into traffic logs, we can track down all incoming requests through Distribusion network. Implementing x_request_id into all packets has accelerated the troubleshooting process in case of an incident.

For providing better readability of logs, all application and HAProxy logs have been redirected into a log store service called as, ElasticSearch and then all stored logs in ElasticSearch get visualised with Kibana. Implementation of x_request_id which has been generated by HAProxy-LUA right on the entry edge of the network can be seen in figure 1. As it can be seen in the bottom table of the figure, by April 2019, all the incoming requests have got their own unique X_request_id.

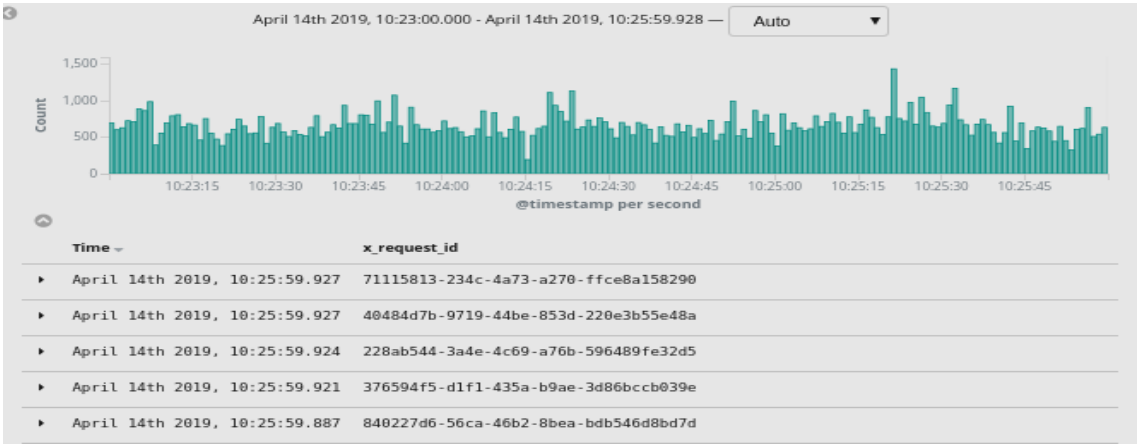

Figure 1: Implementing x_request_id as tracking ID for all incoming requests.

Some other identity tracking options have been implemented in front of Distribusion's API by moving it behind AWS CloudFront solution, such as geographical location, operating system, internet browser and user device of users. These type of fields can

provide better transparency on where exactly our requests are initiated from. Figure 2 represents the top three regions that Distribusion's traffic is coming from.

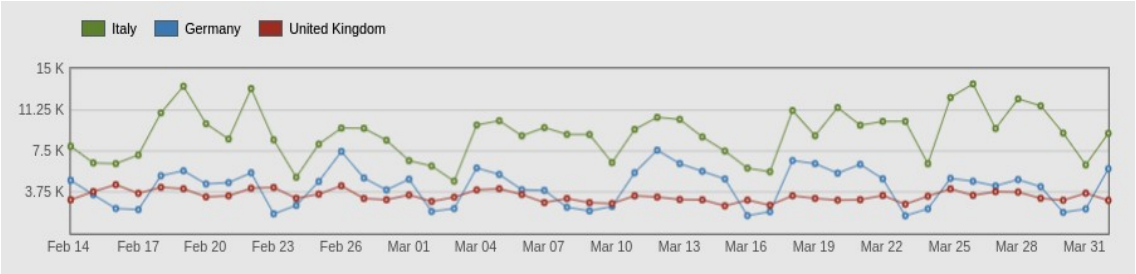

Figure 2: Identifying Requests location.

Also, having a clear recode of most popular in use devices, web browsers help us to improve application products to provide better support and data availability for our end-users.

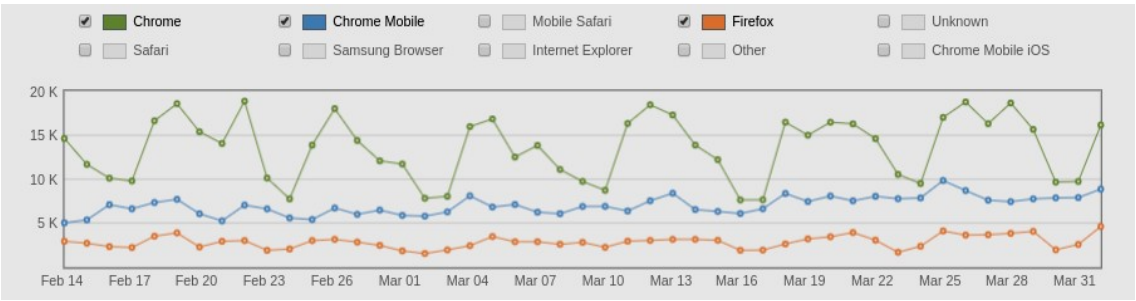

Figure 3: Internet browsers that initiated requests.

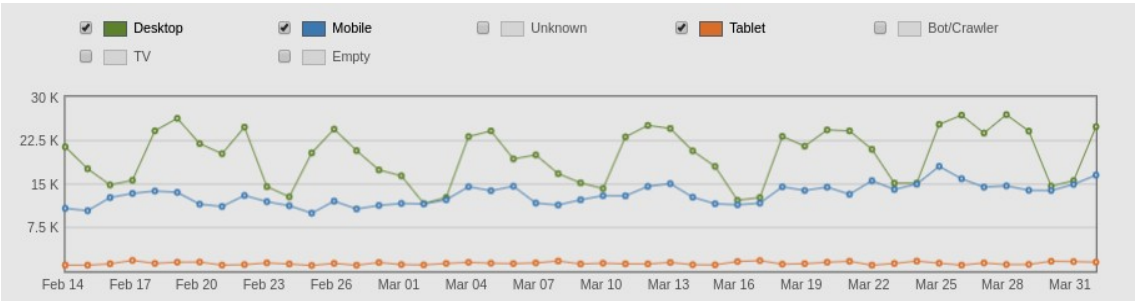

Figure 4: Electronic devices that initiated requests.

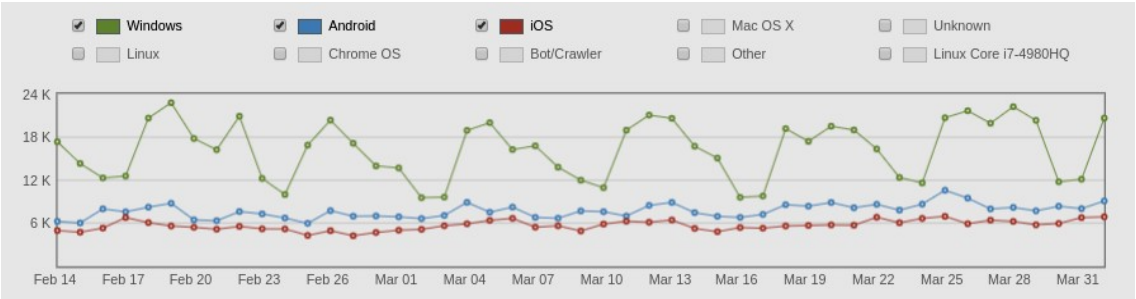

Figure 5: Operating systems that initiated requests.

As it can be seen in the above figures, figure 2, figure 2, figure 4 and figure 5, moving Distribusion's API behind AWS CloudFront not only provided us with better availability for all API users but also provide better visibility on all incoming requests. Having a good record of all these pieces of data can give us a better understanding of correlating abnormal traffic patterns through different logs that we have.

Distribusion does not have a limitation on utilising these type of requests tracking as part of its monitoring since it has been claimed in its GDPR compliance and personal identity of end-user does not get recorded. Knowing from which regions our request mainly initiate from help us to adjust the edge resources for providing better availability of data for clients.

3.1.2 Monitoring libraries and codes

Continuous monitoring on the codes that get pushed by developers to the code repository and third-party libraries that they are using in their codes is another essential and undeniable area for monitoring in DevSecOps era.

The author has integrated two code security tools for scanning security of codes that push to the repository; These tools called as Brakeman and Warnings Next Generation Plugin. [7] Also all application's codes have been moved to the GitHub since it performs automated security check for finding potential security vulnerabilities in the new codes that pushed to the repository. GitHub automated security check offers excellent visibility and frequent feedback on the potential vulnerabilities that might turn to a security breach. [8] [9] As it can be seen in figure 9, thanks to the regular security scans against codes, immediate feedback on the code deployments and remediation actions on improving security, no vulnerable libraries have been pushed to the source code repository since December 2018.

3.2 Resource monitoring and alerting

Monitoring assets, such as network bandwidth, database parallel connections, CPU and memory that have limited resource is a right way for predicting potential resource limitation that might happen in the future base on the current behaviour pattern. Having a proper monitoring and automated alerting base on predefined thresholds for each resource can guaranty full availability on the company's assets. The author has

integrated some monitoring tools such as Prometheus, Node Exporter, Cadvisor, AWS cloud watch, Grafana, VictorOps, Slack together to scrap right metrics from all corresponding resources and visualise the result. In Distribusion technologies GmbH, available resource monitor in three different levels, network, server and container. Monitoring in these different levels, bring better visibility for the DevOps team to narrow its investigation in case something goes on fire. For example, the following figure represents the current traffic load on the Distribusion Load Balancer with a defined threshold which triggers an alert in case of observing a spike in traffic load. The same type of monitoring and alerting pattern has been implemented for each of the resources mentioned above.

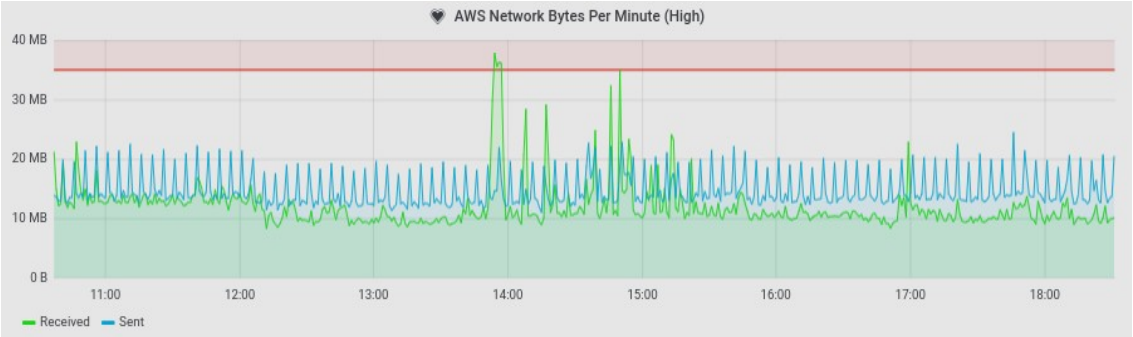

Figure 6: Ingress and Egress traffic load on Distribusion's Load Balancer.

3.3 Monitoring payment system

Experiencing numbers of fraud booking with stolen credit cards made sense to put some effort into securing our payment application and afterwards put it under constant monitoring. Here are some security rules that have been defined for Distribusion's payment system and if any of these rules get violated customers get redirected to the respective bank portal for conducting 3D secure authentication of the customer; This will provide guaranty that real owner of the credit card do shopping. [10]

- Limit usability of the credit card to the same country card has been issued.
- Limit maximum amount of transaction to a reasonable amount.
- Limit number of declined credit card transaction per user.
- Limit the number of transactions that can be done by a single account.

Figure 7 represents the effectiveness of these type of rules between October 2018 till March 2019, on the axe x; And number of blocked payments, dark blue line, on the axe y in this period.

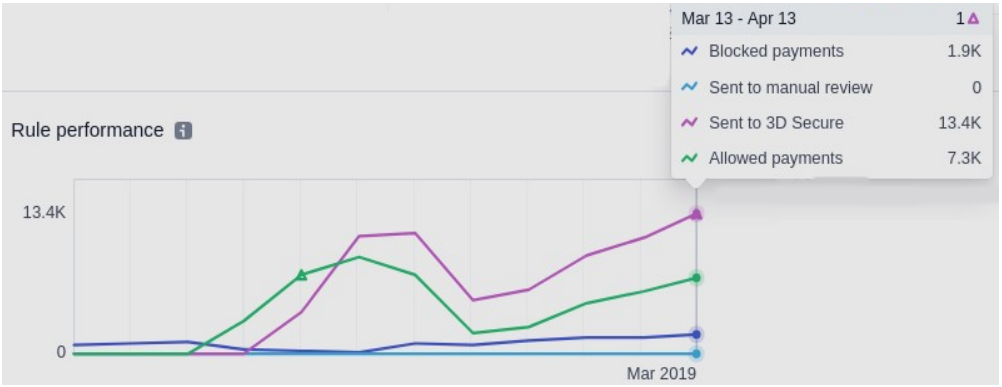

Figure 7: Performance of security rules for payment system.

3.4 Alerting

A meticulous monitoring system should also back up with good alerting rules. Keeping track of each software stack and infrastructure assets can be a useful input for setting up the right threshold on differentiating between normal and abnormal states for firing alert whenever abnormal state observes.

In Distribusion's Technologies GmbH, the author has implemented an alerting system by utilising tools such as Prometheus, Alert Manager, AWS cloud watch, Grafana, Slack and VictorOps. In Prometheus and Grafana more than fifty different types of alerting rules for monitoring Distribusion's assets has been defined. Alert base on severity has been separated into two distinct parts, non-production alerts and production alerts; Non-production alerts will ping on duty person through a message in slack and once production alerts observe VictorOps application will directly call on his cellophane. Production incidents go through two escalation policies, as first step alerting system tries to immediately ping on-duty engineer and if he does not acknowledge the alert in less than ten minutes next engineer get ping till he or first engineer acknowledges the incident alert; this has reduced reaction time on critical alerts. As it can be seen in figure 8, maximum reaction time on production alerts is less than eight minutes in the Distribusion Technologies GmbH in April 2019. Unfortunately, the author does not have any record of the reaction time, before introducing the new alerting system and escalation policies on the alerting system but alerts could remain

unnoticed for hours or even days since there was no any precise rotation and escalation policies in place. As a side note regarding the figure 8, x axe represents each day of the month(April), and Y axe contains integer digits as a representative for minutes.

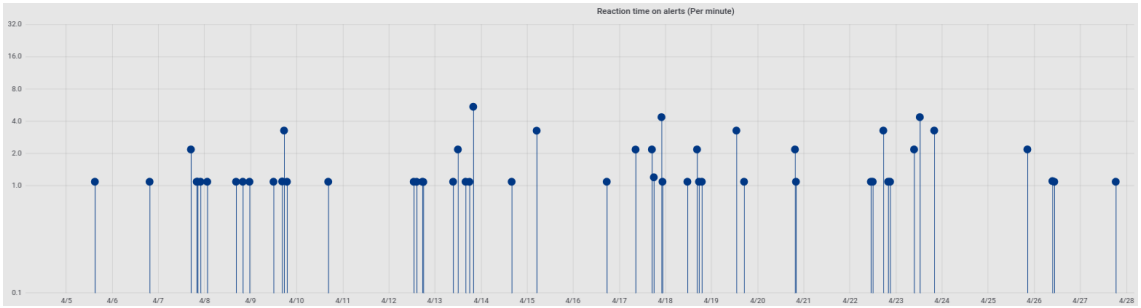

Figure 8: Reaction time to production alerts in Distribusion in April 2019.

4 Securing applications

According to a recent study that has been conducted by Department of Homeland Security (DHS), ninety percentages of reported security incidents against critical infrastructure result from exploits against defects in the design or code of the software. [11] This mean it having a secure infrastructure does not guarantee that company is immune against security risks; A secure system can only result by having a secure infrastructure alongside with secure applications which have been examined against security vulnerabilities.

One the most practical way for being sure about system reliability, and integrity is to include security as a built-in part into both development and deployment phase. Integrating security into application demands two main things, close cooperation with developers to shift security to left as much as possible and automating security checks against all codes that are going to deploy.

As it goes by itself, to make something secure we need first to know it inside out. For this matter, the author is going to elaborate software development and deployment pipeline in Distribusion's Technologies GmbH. In Distribusion Technologies GmbH, as soon developers write their code they can push it to the source code repository under the respective branch that they are developing code for. By having their code in the code repository, they can proceed with the deployment pipeline. Deployment tool deploys all applications inside Docker containers in all environments ranging from playground, and production. All of the docker images that the deployment tool is using for deploy codes on are hosted in a public registry. Plus, communication between application containers orchestrated using Swarm and each of these application containers have a different type of subservience base on their functionality.

In this chapter, the author is going to discuss two main phases that he has taken to mitigate application security vulnerabilities in Distribusion Technologies GmbH. These steps are involved from, shift security to the left, image and container security.

4.1 Securing applications

In the DevSecOps methodology security is everyone's responsibility, and when it comes to the application security, these responsibilities mainly fall on DevOps and the developer's shoulders. Because of lack of security experts in Distribusion Technologies GmbH, it is not feasible to educate current (and future) developers fast enough, but it can be a good and highly relevant approach to provide them with an already made security policies to help them with writing secure codes. Moreover, have some security scanner tools running in all stages of software development life cycle (SDLC) and provide them with quick feedback can serve great value for increasing code security in the company. Last but not least, we are going to secure the way previous integration tool has been utilised and inject some security measurement into its functionality. [12]

4.1.1 Integrate security as developers code

One of the most important steps for integrating security into continuous integration / continuous delivery(CI/CD) process, is to integrate as early as possible in the life cycle. There are some essential advantages into this, such as creating a short feedback loop between the discovery of a security flaw in the developer's code and bring it back to the responsible developer to fix. This quick feedback, provide a faster and cheaper for developers to fix something in the code that they have just coded rather than fixing a security issue in an old code they developed a couple of months ago. Integrating security checks before application deploy on production can save a lot of costs that business should pay when a security flaw detected in the production application. This concept has been proven by a study conducted by NIST, that it is much cheaper and more effective when we shift the security on the early stage of the development pipeline. As it has been shown in table 1, the cost of mitigating security issues increase as we allow issue move to the end side of the software development life cycle (SDLC). [13] [14] Following diagram has been taken from "The Economic Impacts of Inadequate Infrastructure for Software Testing" study, which has been conducted by the National Institute of Standards and Technology in 2010.

Requirements Gathering and Analysis/ Architectural Design	Coding/Unit Test	Integration and Component/RAISE System Test	Early Customer Feedback/Beta Test Programs	Post-product Release
1X	5X	10X	15X	30X

Table 1: Relative cost to repair defects when found at a different stage of software development Life cycle. [13]

By knowing above mentioned reasons currently In Distribusion Technologies GmbH, developers have been educated to follow the following security policies when they start coding. But before going through them, it worth to mention that the author has moved all already written codes form in-house source control management repository to the GitHub. GitHub by default performs regular security scan with its rich Common Vulnerabilities and Exposures (CVE) security database, against all code repositories. Here are the security measurements that developers should consider while writing code in the Distribusion Technologies GmbH. [15]

- All code repositories are private.
- All incoming objects have to pass the integrity check.
- All communications with code repository are encrypted.
- None of the developers is using privilege account to develop.
- Block sensitive data being pushed to the project repository by git-secrets.
- All master branches are protected against deletion and direct commits.
- Repositories hooked to dependency vulnerability tester called as Snyk.
- All commits signed by developers and this mechanism verify periodically.
- Repositories hooked to a code quality reviewer called as CodeClimate.
- Repositories have a proper .gitignore file for preventing information leak.

Effectiveness of integrating security while coding can be seen in the following security vulnerability report, figure 9, that frequently perform by GitHub aginst Distribusion's code repositories. As can be seen in figure nine, since December 5, no new security vulnerabilities have been reported.

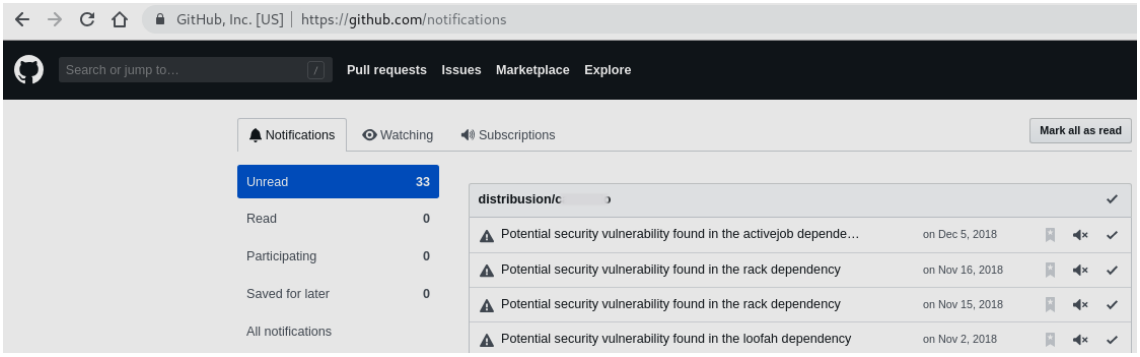

Figure 9: Latest report from GitHub Security Vulnerabilities.

4.1.2 Autonomous security into CI/CD pipeline

However, by integrating security while coding most of the vulnerabilities catch either in the client machine or in the source control management(SCM); but always some security vulnerabilities slip from these two phases and find their way into the deployment phase. For this matter, It does make sense to have an automated vulnerability security check in the deployment pipeline with frequent feedback for making us assure of code security before deploying it to the production. In Distribusion Technologies this has been implemented by integrating Brakeman, A static analysis security vulnerability scanner for Ruby on Rails applications, into Jenkins which use as the continuous integration tool. Appendix number one and Appendix number two representing the effectiveness of integrating security vulnerability checks into CI/CD pipeline after development and DevOps teams acted adequately on them. [16]

4.1.3 Develop a culture of visibility

Contrary to what most developers think, the responsibility of the development team does not finish after their codes deploy in the production environment. By adaptation with the DevSecOps, developers remain responsible for the functionality of their code in the production environment. Nobody like developers who have written the code, know how their code can compromise, so their collaboration on monitoring code behaviour in production serves excellent value to having better application security. Alongside with DevOps team, the development team also should regularly monitor their code behaviour and analysis what kind of requests are coming to their application endpoints and how applications respond to the incoming requests. In Distribusion's Technologies GmbH, metrics from live running applications collect by Prometheus and visualise by Grafana to have traceable metrics of application's behaviours.

4.1.4 Secure deployment jobs

Other than software development there are some concerning security problems when it comes to the software deployments, such as lack of a centralised vault for storing secrets rather than putting them in plain text format on deployment jobs. The author with the help from the development team, have implemented HashiCorp vault as a centralised secret management tool for all applications and cleaned up all Jenkins jobs of plaintext secrets. By introducing vault into integration tool, Jenkins read required environment variables from the vault server and inject it to the application during deployment; Jenkins has the least level of access privilege, read-only, to the vault server. Table 2 represents this improvement in protecting the credentials.

Before Using Vault	After using Vault as the secret manger

`# Application (system)`
`export RAILS_THREADS=8`
`export RAILS_ENV="production"`
`export RACK_ENV=dbus_rack`
`export GEMSERVER_USER=p[redacted]m`
`export GEMSERVER_PASSWORD=L[redacted]`
 	
`cat >$WORKSPACE/config/database.yml <<EOL`
`# Managed via Jenkins`
 	
`ProductionDB:`
`  adapter: mysql2`
`  encoding: utf8`
`  host: [redacted]`
`  port: 3306`
`  database: [redacted]`
`  username: [redacted]`
`  password: [redacted]`
`  pool: 16`
`EOL`
	`# Application (system)`
	`export RAILS_THREADS=8`
	`export RAILS_ENV="production"`
	`export RACK_ENV=$RAILS_ENV`
	`export GEMSERVER_USER=${GEMSERVER_USERNAME}`
	`export GEMSERVER_PASSWORD=${GEMSERVER_PASSWORD}`
	
	`cat >$WORKSPACE/config/database.yml <<EOL`
	`# Managed via Jenkins`
	
	`ProductionDB:`
	`  adapter: mysql2`
	`  encoding: utf8`
	`  host: ${MYSQL_CORE_HOST}`
	`  port: ${MYSQL_PORT}`
	`  database: ${MYSQL_CORE_DATABASE}`
	`  username: ${MYSQL_CORE_USERNAME}`
	`  password: ${MYSQL_CORE_PASSWORD}`
	`  pool: 16`
	`EOL`

Table 2: Comparison between the use of variables in deployment jobs before and after utilising HashiCorp vault for managing secrets.

Since this security vulnerability mitigation directly related to reducing the attack surface on deployment pipeline, we discuss this topic in chapter four too.

4.2 Image and container security

As all Distribusion's applications deploy on Docker containers improving security on both container and their base images is a necessary step to take in the process of adapting with DevSecOps.

Containers providing isolation on the process level, they face the same challenges that a regular process might face, but since containers act like a wrapper around the main application code that is running inside them, the author is planning to decrease all the threat factors that might affect both functionality and security of this containers. The reason that the application's functionality is as important as other security factors is that if the application does not function as expected availability of the data which is an angle of CIA will tamper. Having the right amount of CPU and Memory available for the application that is running inside the container can guaranty service availability for the application that is running inside containers.

Regarding the integrity of the container images, the author has established an automated method for verifying the integrity of base images that containers are running on. This process has been implemented by using Anchore in the deployment pipeline; Anchore performs some customizable security checks against container images to validate container image.

4.2.1 Spotting existing security flaws

Containers also like any other process might face different types of threats and challenges. In Distribusion Technologies GmbH, these security risks against the container and their images can potentially happen in the following seven different ways.

- Breaking out of the container; this can happen because of lack of right isolation on namespaces which potentially can lead to being vulnerable against cross-container attacks. By having this type of vulnerability in our container application, an attacker could leverage his access to other containers; For example, if one of those containers that host carrier's application compromised by an attacker, an attacker could leverage his access to those containers which were hosting financial applications. [23]

- Running container as non-root users; Almost all application containers were running with root user privilege and without any restriction on allowed capabilities, which means misbehave application or potential hacker has root access to the host machine that container is running on.

- Resource overuse; This can leads to self DOS attack from inside the container. This might happen for different reasons such as bad codes, or a misbehave third-party libraries with leaking memory, CPU or even storage issue. This potential issue has been solved by assigning a predefined amount that each container can utilise; This has been hardcoded separately for each application inside their Docker deployment files.

- Tampering of container images; Unreasonable access privilege to the container image repository, registry servers, will increase the risk of an insider hacker push malicious code inside container images which are a source of truth for all application that runs inside containers. This security issue has been solved, by cutting developer's access to the Image repository and only integration tool, Jenkins, and DevOps team have the right access to communicate with the registry servers.

- Un-patched OS or applications; using an un-patched operating system for hosting container applications and un-patched third-party libraries that is in-use inside application code was another issue that has been faced in Distribusion Technologies GmbH. These two issues have been addressed in two ways, by periodically running an Ansible playbook against all servers and updating their security packages to the latest version also shifting security to the left and perform security against before each deployment.

- Transferring images to and from the registry in an insecure way; Lack of a proper encrypted way for transferring images to and from docker image registry was exposing an insecure link for man-in-the-middle attacks against container base images. This has been addressed just by moving communication ways from HTTP to HTTPS.

- Exposing unnecessary big attack surface; this potential security risk might happen by using big container base image which has plenty of packages and

applications installed by default. This potentials security issue has been discussed in detail following section.

4.2.2 Creating small and secure container images

Thanks to Docker creating container images have never been more straightforward; we can easily create any container image by specifying the base image, adding customise changes and build the container. Besides the simplicity of use, we need to bear in mind, that using default base images can lead to large images with a large attack surface and potential for having a lot of security vulnerabilities. Because of easy onboarding and rich repository in Ubuntu images, Distribusion's application containers were using Ubuntu their base image; This base image had more than one hundred megabytes overhead to our containers but Distribusion's applications are just a few megabytes in size, and they do not need all the preinstalled libraries inside Ubuntu images. Using this premade image had some disadvantages, such as waste of disk storage, need for more processing time and also a because of its complexity can be the right place for hiding security vulnerabilities and bugs. [25]

Using small base images is the easiest way to reduce the container size and increase transparency. Alpine as smallest Docker image with a rich package index became a good replacement for not only Ubuntu base image but also Nginx base image.

By replacing the Nginx container image with Alpine container image, we have reduced container image size by more than five times, from one hundred and ten megabytes to just twenty megabytes. This reduction in the size of images not only accelerated pull, push and application build time, but also reduced the attack surface on the container image by removing unnecessary libraries. As it has been mentioned, smaller containers have proven measurable advantages regarding their performance and security.

Table 3 represents the performance comparison between two types of container images regarding the required time for pull and push from and to the image registry and their build time.

Container Image	Size	Build + Push time	Pull
nginx:1.12.1	107 MB	540 Seconds	12.997 Seconds
nginx:alpine	20 MB	308 Seconds	2.052 Seconds

Table 3: Evaluating container images size and performance.

Five times drop in the base images size and six times drop in the time that is required for pulling images from the repository, can serve great value to better availability for applications in case of node failure. For example, if one production node goes down and a new replacement node come up and join to the cluster. Running applications on smaller images can decrease the images pulling time on the newly added node; This means cluster health can get back into the green state six times faster in comparison with time when we were using bigger images.

4.2.3 Test and verification

To verify the outcome of the above mitigation steps that had been applied to the Distribusion's application containers and docker engine, the author uses an official docker security check tool, called Docker Bench for Security. Docker Bench for Security is a tool for checking the most common best practices around deploying Docker containers in production. This tool performs a security scan through an automated Jenkins job that has been created by the author. Table 4 represents the previous and current security status of running docker daemon, docker images, and running application containers in Distribusion Technologies GmbH. The raw output of this test can be found in appendix 3 and appendix 4. [17]

Area of security check	Before	After
Host configuration		
There is a separate partition for containers	NO	YES
Auditing is configured for the Docker daemon	NO	YES
Auditing is configured for Docker files and directories	NO	YES
Docker daemon configuration		
There is a restriction between containers on the default bridge	NO	YES
Insecure registries are not used	YES	YES
User namespace support	NO	YES
Centralised and remote logging is configured	NO	YES
Containers are restricted from acquiring new privileges	NO	YES
Container Images and Build File		
Containers are not running as root	NO	YES
All containers use trusted base images	YES	YES
Unnecessary packages are not installed inside the container	YES	YES
Container Runtime		

AppArmor profile is enabled on running containers	NO	NO
SELinux security options are set	NO	YES
The host's network namespace is not shared among containers	NO	YES
None of the containers running with networking mode ‘host.’	NO	NO
Memory usage for the container is limited	NO	YES
The CPU usage for the container is limited	NO	NO
Container's root file system is mounted as read-only	NO	NO
Incoming container traffic is bound to a specific host interface	NO	NO
Ensure 'on-failure' container restart policy is set to '5'	NO	NO
Ensure PIDs C-Group limit is used	NO	YES
The container is restricted from acquiring additional privileges	NO	YES
Container health is checked at runtime	NO	YES
Ensure the Docker socket is not mounted inside any containers	NO	YES
Docker Swarm Configuration		
Data exchanged between containers are encrypted on different nodes on the overlay network	NO	YES
Swarm manager is run in auto-lock mode	NO	YES

Table 4: Status of previous and after the state of host and Docker container deployment.

5 Securing platform

In this chapter, the author wants to secure infrastructure in two separate but related phases. In the first phase, I am going to apply a better level of isolation between running containers, servers and environments and in the second phase, I will adapt the current infrastructure to code.

5.1 Moving toward Zero Trust Network security model

Today the only safe assumption is, that the attackers trying to get into the network or they are already in the network. One of the most efficient ways to tackle this issue is to limit the number of ways they can get into the network and contain the damage if they do. By doing so, if attackers find their way into the company network, they do not get access to the rest of the network assets. This can be done in the following phases, shrink the attack surface, contain the attack and lower security risk. In this chapter, we are going to discuss the ways that we can tackle both shrinking attack surface and lowering security.

Before proceeding with this chapter, it does make sense to have a common definition of the attack surface; any unsecured assets or privileged code, Distribusion API and its gateways to its clients API in this case, with an interface accessible by domain from out of the trusted zone, is called attack surface. According to this definition all our mission-critical data, servers, services and software products should be only accessible to the authorised users and applications with the least level of access privilege. This matter will not achieve without having a clear and comprehensive understanding of the current infrastructure and its existing problems. Distribusion’s network architecture was looking like following diagram, figure number ten, before re-architecting platform base on Zero Trust Network security model.

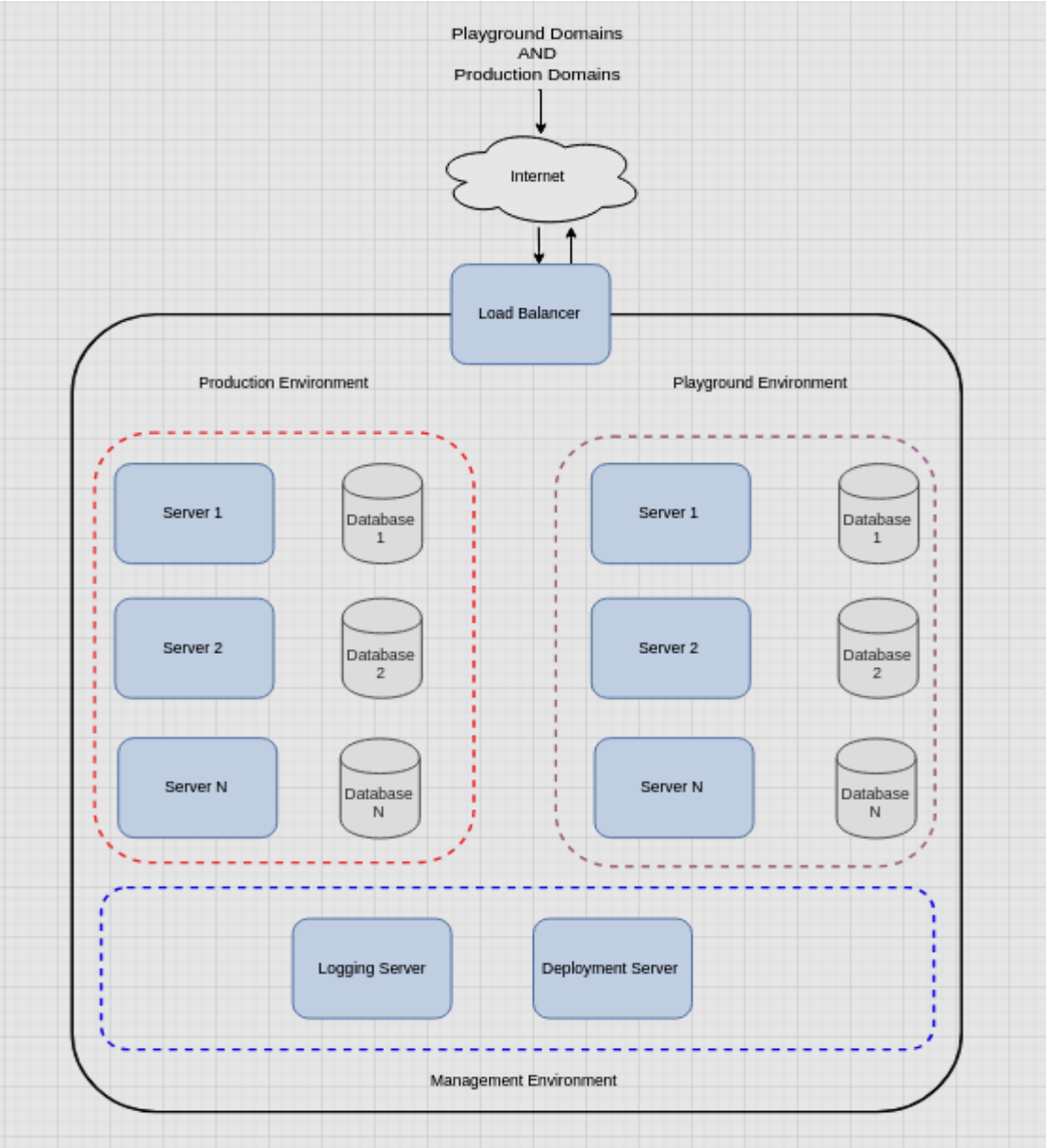

Figure 10: Distribusion's network architecture before adapting with Zero Trust Network

As it can be seen in figure number ten, all three environments have been created in one virtual private cloud (Big Black rectangle), and each environment has one layer of isolation from other environments, by their subnet network (Colourful and point base rectangles). Also, there has been no isolation between servers and databases in each environment and each server not only have direct access to all other servers, but also it had direct access to databases. This type of network architecture called as single trust boundary (STB), which means just one layer of isolation is around each asset in this type of architectures and attacker can easily leverage its level of access to other assets.

5.1.1 Problem with the current state

As an unfortunate and inevitable fact in the software industry, regardless of the effort developers put into their codes always there are some unknown vulnerabilities in their software products. These vulnerabilities can come directly from the code that has been written in-house or from the third-party libraries that codes using in their background.

Also, high and frequent demand in start-up companies for new features, in a better, faster and cheaper way, can put both DevOps and developers under pressure to deliver the requested features before the deadline by any cost even by using unreliable third-party libraries. These types of un-trusted and unverified libraries might serve good to the business in the short-term but in the long-term can put the whole business into risk because of the security flaws that they expose to the company.

One of the most effective ways of tackling these insecure third-party libraries is to establish a solid upgrade policy on them. However, there is a gap between when software vulnerabilities disclosed and when they identified and patch. Moreover, Frequent upgrade and apply patches to the third-party libraries does not mean that we can be assured that we are immune against the risk that might cause because of zero-day vulnerabilities inside third-party libraries that we are using in applications. [18]

Sometimes the problem does not limit just to the codes, and in-use libraries, level of individual access to the mission-critical data can also be considered as a critical attack surface. To have accurate products that have been tested with the real-world type of data, In Distribusion we are using some of the production but not personal data in the pre-production stage. However this will increase the accuracy of our final products, but on the other hand, we are expanding our attack surface here which I have to address it after the attack surface mitigation phase.

Plus, having more internet face URLs and API endpoints is equivalent to an increase in the unwanted and malicious traffic load to our assets and network infrastructure. Unwanted traffic load can decrease the transparency on the type of traffic that float toward us also it puts some additional costs such as financial and the compute time on the company’s shoulder.

Moreover, all three environments, playground, demo and production had been created in one single virtual private cloud (VPC) and all playground servers can directly access to the production servers.

5.1.2 Improvement methodology

Before proceeding with the attack surface reduction, we need to break down the current infrastructure assets and applications base on their functionalities and rules. This separation will give us a good understanding on the location of our confidential data on servers, location of applications, what kind of data each application is dealing with and what type of access level should they have to each other. To have clear and accurate answers for questions mentioned earlier, I have drawn the following table, table 5, to have a precise record of the attack surface status before proceeding with the reduction phase.

General attack surface status	Status	
	Yes	No
Is there any replica for confidential data?		●
Is there any backup policy in place for confidential data?	✓	
Is there any user has root access to confidential data?		●
Is there any encryption at rest in place for confidential data?		●
Is there any encryption in transit in place for confidential data?		●
Is there any authorisation method to this data?		●
Is there any metrics that represent the status of confidential data?		●
Is there any authorisation needed for internet face applications?	✓	
Does any other person have root access to this data except admin?	✓	
Does any other person have direct access to this data except admin?	✓	
Does any user other than root, has access on docker daemon?	✓	
Do applications have access to this data without an authorisation?	✓	
Is there any 2FA in place for accessing confidential data?	✓	
Do vulnerable third-party libraries get spotted immediately?	✓	
Is there any precise isolation between applications?	✓	
Is there any fault-tolerant implementation for critical applications?	✓	
Is there any isolation between test and production environments?	✓	
Is there any fault-tolerant implementation for critical servers?	✓	
Is there any isolation in depth in place for mission-critical assets?	✓	

Is there any security awareness sessions for developers?	✓	
Is there any source code analyser in place for the written codes?	✓	
Is there any sandboxing for libraries to minimise the exploit impact?	✓	
Is there any use of vulnerability scanning tool in place?	✓	
Is there any robust incident response plan in place?	✓	

Table 5: Before the attack surface reduction for confidential assets.

For hiding, confidential data implementing different layers of isolation can be an effective approach. For example, using VPN as our main gateway for authenticating users, putting confidential and production data in a different VPC than development and demo VPCs and using a separate load balancer also has been implemented as a good approach for separating traffic flow to the confidential and un-confidential data. Separating production assets from other assets will help to have better visibility to all ingress and degrees traffic to the mission-critical environment. In this stage playground, demo and production assets, their respective applications, servers, and databases have been moved to different virtual private networks (VPC). At the end of this phase, we have separated our four environments into different VPCs. Previously, all of these environments were hosted in one single VPC without any restriction from Network Access List or Security group rule; which means playground servers can directly communicate with production servers.

Encryption at rest and encryption in transit is a proven good security practice for protecting the integrity of data. By using reliable encryption for storing data in the database storage, we can be assured that the integrity of the data remains preserved even when the attacker gains access to the database storage. Regarding reducing the attack surface on the data that are stored in the database, cutting direct human access to the main database is a must to do. Cutting individuals access to confidential assets with eliminated mistakes that might happen because of human error. Since in Distribusion, most of the employee needs to have access to this data and we cannot cut their access, using a slave replica database alongside to the main database can be a good way to go. This means, those users who need to have access to production data can have read-only access to this data which are a replica of the main data, but they are hosted on another database other than the primary database.

As we cannot eliminate the risks for all the criteria as mentioned earlier’s, but now since we have spotted them by breaking them down, we can control them better. For example,

as start-up Company with the limited in-house developers it is not feasible to write all the needed codes from scratch and stop using third-party libraries, but we can have an automated security check on those libraries in our source code management. This will help us to either patch the vulnerable library or replace it with a secure one. Have said that, educating developers about security flaws that previous or currently in use third-party libraries have and the necessity of knowing the fact that what kind of libraries they are committing can be an effective approach toward securing code. Also, implementing static and dynamic security code analyzer in the development and deployment pipeline can be a good approach for making sure that malicious code does not find their way to the production; this will cover more in detail in chapter five under shift security to the early stage of the development.

5.1.3 Moving from Single Trust Boundary to Zero Trust Boundary

As it has been mentioned in the current infrastructure all four environments (playground, demo, production, and management) has been created in one single virtual private network(VPC), and anyone with access to the playground assets can have direct access to the production assets. The most basic and simple model is the single trust boundary(STB) network security model. As its name implies, it refers to networks that have single layer of trust can be defined as a network that shares one single VPC for hosting all company assets and they get protected on the VPC level by Load Balancers (HAProxy and NLB), this is the current architecture of the network that I am planning to improve its security by implementing isolation on different layer for existing assets. [19]

Network segmentation is a very effective technique that has been applied to the previous infrastructure. Network segmentation ultimately relies on a very simple principle, if you cannot reach the server you cannot hack it. As it has been explained base on the figure number ten, Distribusion’s network architecture had been designed base on Single Trust Boundary (STB) network architecture, which means each asset can have direct access to other assets in the network and there is no restriction around it. On the other hand, in Zero Trust Boundary (ZTN) network architecture since each asset have multiple layers of isolation around each asset such as network subnet group, network access list(NACL), security group(SG) and kernel namespace. In this architecture, if any asset gets compromised by a hacker, these layers of isolation already

contained that attack and the hacker cannot easily leverage his level of access to other assets.

5.1.4 Implementation and evaluation

Here we are going to improve the security of the current infrastructure architecture from Single Trust Boundary (STB) to Dual Trust Boundary (STB) and afterwards to Zero Trust Network (ZTN) to complete our attack surface reduction phase. By moving toward Zero Trust Network (ZTN), technically we reduced the reachability to confidential data for internal and external threat actors.

As the first step, by adapting our infrastructure with Dual Trust Boundary network model, we were taking advantage of two layers of security which consists of the internal and external zones. These two layers can be categorised as internal and external networks. This means, by moving toward a DTB network security model we are building a better defence layer against an internally compromised component which can become more prevalent in organizations; For example If an internal or external threat actor compromise an asset in any environment it is not easy for him to leverage his access to other environments since each environment isolated from another environment by a layer of VPC and appropriate Network Access List (NACL). [20] [21]

Finally, we have been adapted to the most recommended network security model which is defined as the Zero Trust boundary, taking advantage of all layers of security that we had in the DTB but with a fundamental difference when it comes to the server security which means we put another layer of isolation around each server that we are hosting. Server isolation has been defined by reviewing IPTables and security group (SG) rules of each server. This will guarantee that in case one of our servers gets compromised, the attacker’ access level will stay limited just to the infected server, and it cannot leverage its access to other servers.

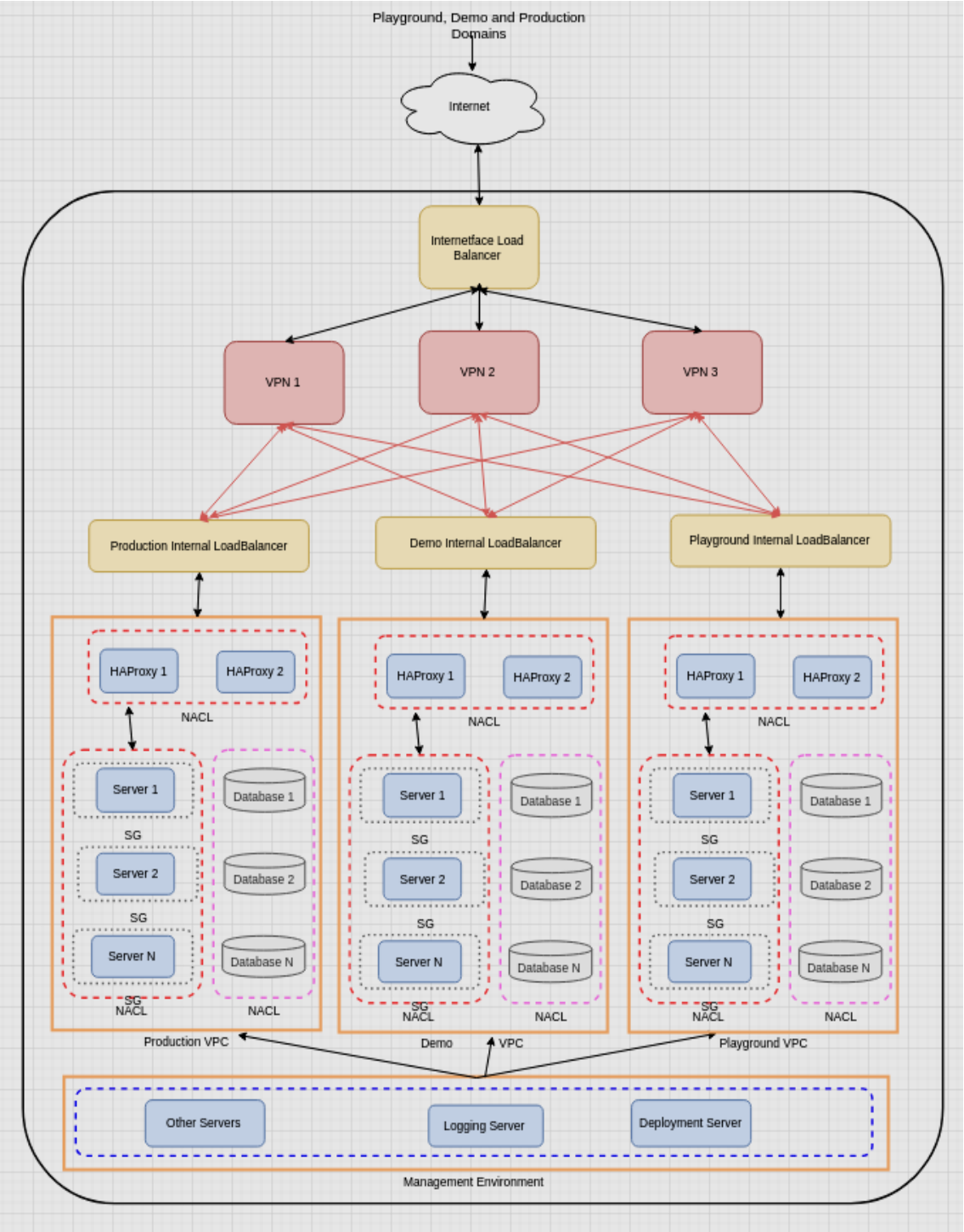

Figure 11: Distribusion's network architecture after adapting with Zero Trust Network

As can be seen in figure 11, in the current network architecture not only environments have two layers of isolation between themselves called as Virtual Private Cloud (VPC) and Network Access List (NACL), but also we there is another layer of isolation between each server called as Security Group (SG),

Implementing full attack surface reduction on an already existing infrastructure and applications is a hard task to complete, but I have tried to achieve this matter by breaking assets into different sections; In this section, I have tried to mitigate existing security risks through limiting access to critical assets, replacing, isolating or patching vulnerable libraries, applying more security groups and upgrading firewall rules, putting isolation between environments and servers, reviewing users access level and decrease it to the least level of privileged is also another important thing to do.

After evaluating the current business need for giving access to production data to employees; after recording the previous state of the number of access to production data, I have mitigated this access by cutting the number of access to this data. Prior and current status has been represented in table 6.

Data attack surface	Status	
	Before	After
How many people have access to production data?	14	10
How many people have the right to access to the production data?	5	4
How many people just have read access to the production data?	9	6

Table 6: status of before and after states of attack surface mitigation on confidential data.

Since with the current business need, it is not possible to cut individual access to the copy of production data which has been replicated on pre-production stages, I have decreased the level of replication from full replication to only those data that developers really need during the process of development; This has been discussed and agreed with developers. This has plenty of advantages, such as speed up the whole replication process, decrease the data storage, and decrease the risk threshold to the confidential data. As can be seen in table 7, after attacking surface mitigation on confidential data, the size of the data that developers can have access to has been dropped to five percentages of production data.

Different environments	Status	
	Production Data	Data Provision
Production environment	10 GB	10 GB
Demo environment	10 GB	0.5 GB
Playground environment	10 GB	0.5 GB

Table 7: Attack surface mitigation on data provisioning.

Table 8 represents the status of Distribution's attack surfaces after applying mitigation approaches; before states can be seen in table 5.

General attack surface status	Status	
	Yes	No
Is there any replica for confidential data?	✓	
Is there any backup policy in place for confidential data?	✓	
Is there any user has root access to confidential data?	✓	
Is there any encryption at rest in place for confidential data?	✓	
Is there any encryption in transit in place for confidential data?	✓	
Is there any authorisation method in place for accessing this data?	✓	
Is there any metrics that represent the status of confidential data?	✓	
Is there any authorisation needed for internet face applications?	✓	
Does any other person have root access to this data except admin?		✓
Does any other person have direct access to this data except admin?		✓
Does any user other than root, has access on docker daemon?		✓
Do applications have access to this data without an authorisation?	●	
Is there any 2FA in place for accessing confidential data?	✓	
Do vulnerable third-party libraries get spotted immediately?	✓	
Is there any precise isolation between applications?	✓	
Is there any fault-tolerant implementation for critical applications?	✓	
Is there any isolation between test and production environments?	✓	
Is there any fault-tolerant implementation for critical servers?	✓	
Is there any isolation in depth in place for mission-critical assets?	✓	
Is there any security awareness sessions for developers?	✓	
Is there any source code analyser in place for the written codes?	✓	
Is there any sandboxing for libraries to minimise the exploit impact?		●
Is there any use of vulnerability scanning tool in place?	✓	
Is there any robust incident response plan in place?	✓	

Table 8: Status of after attack surface reduction on confidential assets.

Dividing current Distribusion network infrastructure into two different layers with two type of load balancers, internal (non-internet face) and external (internet face) load balancers, gave us the ability to completely hide confidential data, privileged codes and API endpoints from unauthorised access.

Figure 12 represents the number of malicious requests to the Distribusion's API before, and after applying two stages of mitigation, the yellow section is before applying security mitigation. Axe x represents the days of the month, December 2018; axe y represents the number of incoming malicious requests to the Distribusion's API per minutes. Any request to the Distribusion's API which does not have expected pattern from clients API has been categorised as malicious requests, such as /admin.php which is not expected. Before mitigation, the main API was only reachable through an URL which was pointing to an internet face load balancer, which means anybody from outside was able to send some requests to the main API without any authorisation. To address this attack surface, I have divided the network area into two sections, internal and external areas; this concept has been elaborated more in detail in Zero Trust Network Architecture.

In this phase I have pointed the API application to internal load balancer (non-internet face load balancer), and then I have removed its connection to the external load balancer(Internet face load balancer); Soon after applying this two stages the number of malicious requests to the main API has been dropped till they totally got eliminated. As soon as the main API lost its direct internet access by pointing it to the internal load balancer and it got hidden behind VPN all the malicious request to it went away.

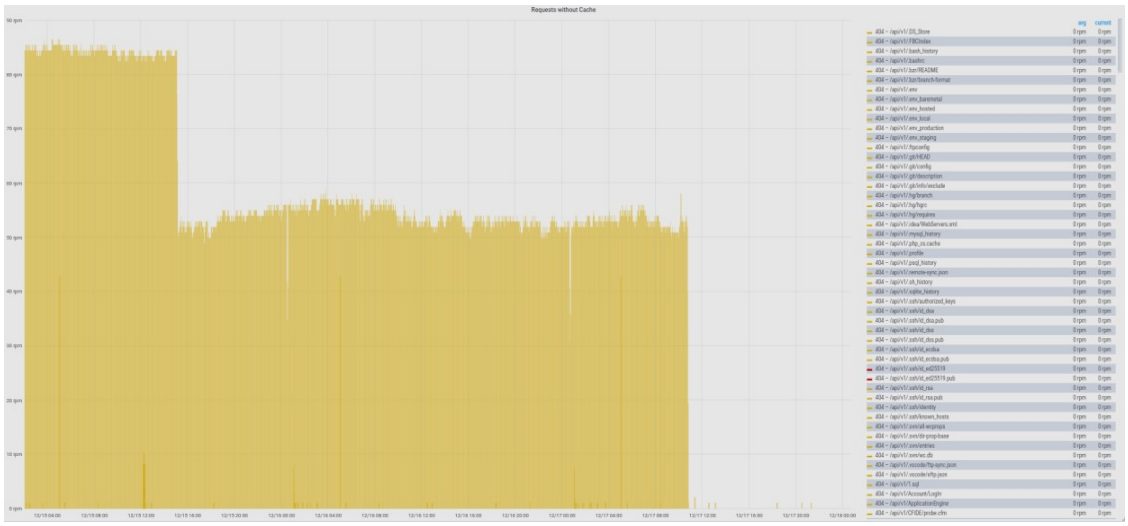

Figure 12: Two phases of drop on incoming fake requests to the API.

5.2 Moving Infrastructure to code

Adapting Infrastructure to code (IaC) has three main advantages, first of all, it can dramatically reduce human errors by cutting their direct touch on mission-critical assets; secondly, since adaptation with IaC bring speed the process, the requested tasks can deliver much faster and thirdly it can potentially increase situation awareness among infrastructure maintainers. [22]

In Distribusion Technologies GmbH we were experiencing ample amount of errors and unwanted incidents while maintaining infrastructure manually, such as assigning wrong security group (SG) and exposing ports unnecessarily to the wild on newly created servers. Moreover, the author has been decided to move the current infrastructure to code to reduce the possibility of happening these types of mistakes on infrastructure. In this phase of the adaptation process, the author has utilised one of well-known infrastructure automation tool, called Terraform and integrated it with AWS auto-scaling, Jenkins and Ansible to have a secure, fast and automated way for managing infrastructure which is free of human mistakes.

5.2.1 Reducing human error

The main problem with manually managing infrastructure is relying heavily on direct human involvement; As it goes by itself humans are prone to making mistakes even on doing repetitive tasks, and this can easily turn to disaster while maintaining mission-critical assets. This claim has been proven in a benchmark study conducted by the Ponemon Institute on “Data Center Outages”, figure 13, it was found that twenty-four percentages of the infrastructure failures could be traced back to accidental/human error. [26]

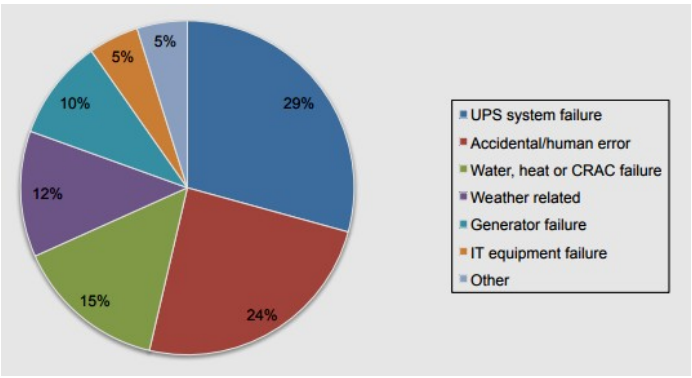

Figure 13: Primary root causes of unplanned outages on infrastructure. [26]

By having a clean and well written and tested code for managing infrastructure, we can reduce these types of human errors on mission-critical assets. A well-written code does not make mistakes in the middle of the night when an incident occurs and if the respective code has done a task correctly before we can be almost sure that it can perform that task with the same accuracy again and again.

5.2.2 Faster delivery

Having code and machine behind maintaining infrastructure rather than human can bring speed to the request's delivery time. By integrating this fast response time with AWS auto-scaling solution, now Distribusion's infrastructure can immediately scale up its required resources base on the current load once respective sensors observe a high spike in traffic toward themselves; This means we can mitigate potential DOS attacks and preserve the availability of our mission-critical assets in Distribusion.

To put this claim into the test, I have conducted a stress test against critical assets. In a normal working day of February 2019, average traffic load toward our main API warrs around 2100 request per minute (RPM) as it can be seen in figure number sixteen. Bottom left-hand side table, in figure 14, represents the response time from the API.

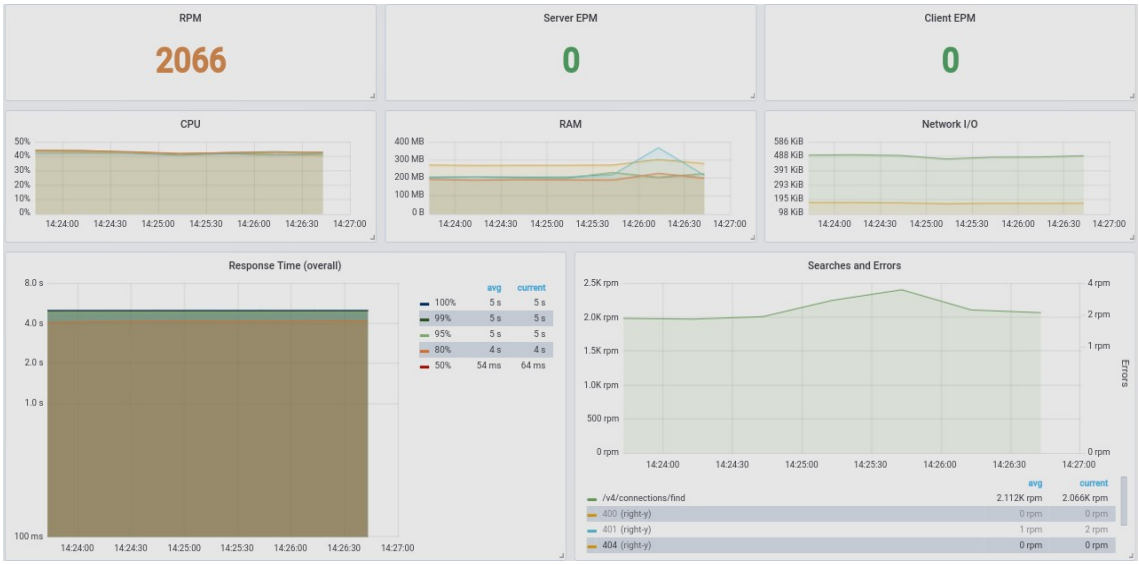

Figure 14: Distribusion's API response status before the stress test.

In my stress test, I have increased traffic load toward Distribusion's API to ten times more than normal state, and as it can be seen in figure number fifteen, still our API can successfully handle this load of traffic with the same response time.

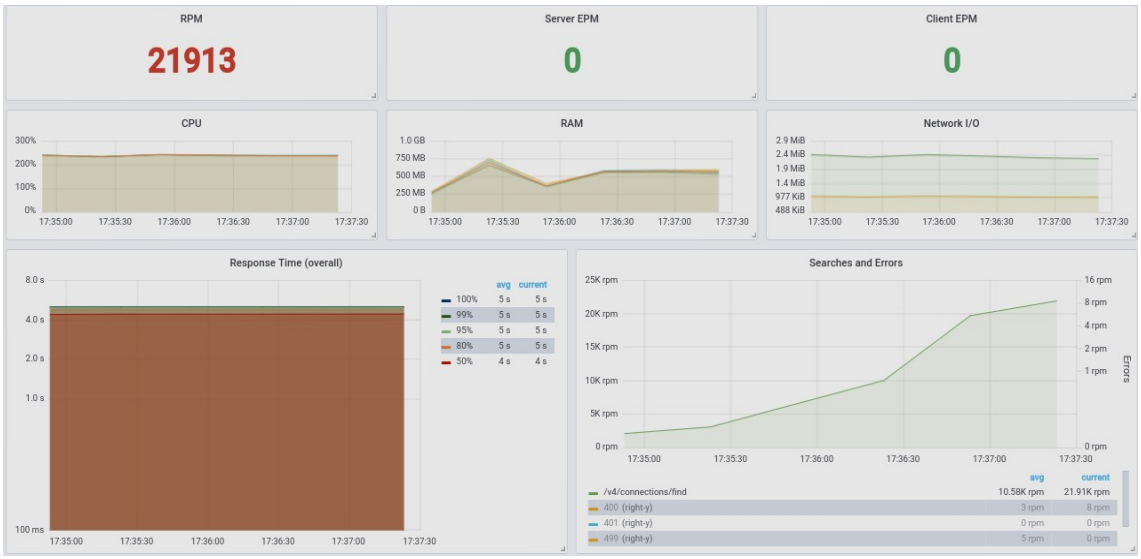

Figure 15: Distribusion's API response status during the stress test.

This elasticity is because of utilising AWS auto-scaling groups with corresponding CI/CD deployment pipeline. As soon as respective AWS auto-scaling groups, noticed a spike in the traffic load automatically provisioned new resources, then triggered corresponding CI/CD pipeline on Jenkins and finally deployed respective Ansible playbook against new resource to join it to the production cluster and redirect some of the traffic on that too. As soon as a decrease in traffic load observes for five minutes, the auto-scaling group adjust current resources to a reasonable amount by removing un-necessary resources.

5.2.3 Situation awareness

Adaptation with IaC not only can leads to having an infrastructure immune of human mistakes and bring speed with better accuracy but also can lead to an infrastructure with a clear records of all the changes, which make it easy for administrator to roll back to the latest stable state of infrastructure in case the latest change break something.

6 Conclusion and Future work

This chapter contains a summary of this study. In this section, the author underlines what has been done and what might be subjected to future work for this topic.

6.1 Conclusion

In this thesis, we explored the steps that had been taken by the author to integrate missed security measurements into a working DevOps adaptation in a European start-up company called as, Distribusion Technologies GmbH. This security integration has been taken place in both application and infrastructure platform, which entitled Distribusion as a DevSecOps integrated company.

In the literature review, some research has been conducted on just one specific aspect of this shift, including container security, infrastructure hardening, and secures deployments. However, a complete shift from DevOps to DevSecOps on a real-world company which includes a lot of subcategories have not been performed.

As for conducting this study a wide spectrum of areas needed to touch in order to improve their security, this study had been conducted in three separate phases, establishing a reliable monitoring and alerting system, improving security in both software development and deployment cycles, and securing infrastructure by cutting human touch on it and applying deep level of isolation in different layer of its network.

In the first step, a reliable monitoring system had been established to provide clear transparency on functionality and security behaviours of Distribusion's network, hosts, applications and its containers. As this new level of transparency backed by two precise escalation policies, to notify the on-duty engineer as an immediate action and notify the next engineer if on-duty engineer did not acknowledge the ongoing incident for ten minutes, the reaction time to incidents has been decreased to ten minutes; This fact has been represented in figure 8. Moreover, this step's outcome also became a reliable

source of truth for recording before and after states of those assets which have been marked to improve their security.

In the second step, those security measurements that have been missed on both application development and deployment get injected to the current Continuous Integration and Continuous Delivery (CI/CD) pipeline. By close cooperation between development and DevOps teams security checks had been shifted to the early development stages, Architectural design or coding stages, of application development cycle; By doing so most of the potential security flaws get caught in the early stage of application development, and their chance on making their way to production has been decreased dramatically and even since December fifth, 2018 no security vulnerability has been pushed to the code source controller (Figure 9). This has been done by implementing frequent security scan against code, container base images on each application deployment and securing confidential secrets.

In the final step, the author moved already made infrastructure to code to keep mission-critical assets immune from direct human touch; Also since current infrastructure had been made without security consideration, the author has re-architected it to adapt it with the Zero Trust Network (ZTN) network architecture. By doing so, some level of isolation had been implemented between container applications, host and network environments; this leads to reducing the attack surface on these entities. In the current infrastructure, if any application container gets compromised, it is not that easy for an attacker to leverage his access level to other containers; the same thing is true for attacks against hosts or environments.

Overall, by close cooperation between the author as only DevOps engineer with development team Distribusion Technologies GmbH has been adapted to DevSecOps. However, the current level of automation on integrating security into all mission-critical processes will make security as an inevitable part for all business developments in the future, but still, there is plenty of room for improvement.

6.2 Future work

Because integrating security to the production environment of an organisation which initially has not been made by security in mind is a time consuming and tedious task

and plenty of elements need to touch even for applying a small security mitigation tasks; still, some areas left which will remain as future works. These areas are including following steps which I am planning to work on them in the future:

1. Cutting all human touch on mission-critical assets by full adaptation with immutable infrastructure; by doing so, misbehaviour asset automatically gets archived for future investigations and automatically get replaced with a new asset. This step decreases the downtime to a minimum when incidents happen.

2. As always there is some level of risk that any critical server can get compromised by any mean, replacing two weeks old production servers with freshly installed servers can clean up all possible backdoor in case asset already owned by the attacker; This process called as retiring old critical assets.

3. Since containers are running just like a normal process when an incident happens they get killed, and their remaining footprint on the host is not enough for future forensic investigation to detect the root cause of the incident. The author is planning to archive the crashed container somehow before it gets replaced with its new replica.

4. Having some services left un-containerised means they do not have that abstraction layer which container put around those services. Moving these remaining services into the container will implement the concept of isolation in-depth for all production application in Distribusion.

7 References

[1] Francois Raynaud, “DevSecOpsWhitepaper”, in https://devsecops.com, 2018; DevSecCon Conference, 2018; *Available: https://www.devseccon.com/wp-content/uploads/2017/07/DevSecOps-whitepaper.pdf*

[2] Thomas F. Düllmann, Christina Paule, André van Hoorn, “Exploiting devops practices for dependable and secure continuous delivery pipelines”, 2018; RCoSE '18 Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering Pages 27-30, Gothenburg, Sweden — May 29 - 29, 2018; *Available: https://dl.acm.org/citation.cfm?id=3194763*

[3] Samu Toimela, “Containerization of telco cloud applications”, Aalto University, 2017; *available: https://pdfs.semanticscholar.org/a11c/cd19a9d37083484b3e20832e2eb1e1c64c18.pdf*

[4] Katia Gomes, “The Importance of DevSecOps”, DeKalb university, 2018; *Available: https://commons.lib.niu.edu/bitstream/handle/10843/17868/The%20Importance%20of%20DevSecOps.pdf?sequence=1&isAllowed=y*

[5] Hasan Yasar, “Integrating Security in DevOps”, Carnegie Mellon University, 2017; *Available: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=502780*

[6] Tony Hsiang-Chih Hsu, "Security Monitoring in Hands-On Security in DevOps: Ensure continuous security, deployment, and delivery with DevSecOps Kindle Edition”, 2018.

[7] Ulli Hafner, “Warnings Next Generation Plugin”, Jenkins documentation, 2019; *Available: https://wiki.jenkins.io/display/JENKINS/Warnings+Next+Generation+Plugin*

[8] Gabriel Avner, “3 GitHub Security Updates You Should Know”, 2019; *Available: https://resources.whitesourcesoftware.com/blog-whitesource/3-github-security-updates*

[9] Richard Chirgwin, “Your code is RUBBISH, says GitHub. Good thing we're here to save you”, published in theregister.co.uk, 2018; *Available:*

https://www.theregister.co.uk/2018/03/23/github_dependency_scanner

[10] Strip secure payment, Published in stripe.com documentations; *Available: https://stripe.com/docs/payments/3d-secure*

[11] American Homeland cybersecurity, "Software Assurance"; *Available: https://www.us-cert.gov/sites/default/files/publications/infosheet_SoftwareAssurance.pdf*

[12] Authors and affiliations, "DevOps' Shift-Left in Practice: An Industrial Case of Application", 2019, First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018; *Available: https://link.springer.com/chapter/10.1007/978-3-030-06019-0_16*

[13] National Institute of Standards and Technology, "The Economic Impacts of Inadequate Infrastructure for Software Testing", 2002; *Available: https://www.nist.gov/sites/default/files/documents/director/planning/report02-3.pdf*

[14] DevSecOps: "How to Seamlessly Integrate Security Into DevOps", 2018; *Available: https://cdn2.hubspot.net/hubfs/1958393/White_Papers/devsecops_how_to_seamlessly__315283.pdf?t=1482418124868*

[15] Simon Maple, Tom Preston-Werner, "10 GitHub Security Best Practices", 2018; *Available: https://snyk.io/blog/ten-git-hub-security-best-practices/*

[16] Paule Christina, "Securing DevOps: detection of vulnerabilities in CD pipelines", 2018; *Available: https://elib.uni-stuttgart.de/bitstream/11682/10046/1/2018-04-17-masterthesis-final-christina-paule.pdf*

[17] Michael Falk, Oscar Henriksson, "Static Vulnerability Analysis of Docker Images", 2017; *available: http://www.diva-portal.org/smash/get/diva2:1118087/FULLTEXT02.pdf*

[18] Stan Wisseman, "Third-party libraries are one of the most insecure parts of an application", 2016; *Available: https://techbeacon.com/security/third-party-libraries-are-one-most-insecure-parts-application*

[19] John Kindervag with Stephanie Balaouras and Lindsey Coit, "Build Security Into Your Network's DNA: The Zero Trust Network Architecture", Forrester Research, Inc.,2010; *Available: www.virtualstarmedia.com/downloads/Forrester_zero_trust_DNA.pdf*

[20] Louis Columbus, "6 Best Practices For Increasing Security In AWS In A Zero Trust World", 2019; *available:*

https://www.forbes.com/sites/louiscolumbus/2019/01/04/6-best-practices-for-increasing-security-in-aws-in-a-zero-trust-world/

[21] Andy Smith, “AWS Security Best Practices in a Zero Trust Security Model”, represented on AWS summit, 2018; *available: https://www.slideshare.net/AmazonWebServices/aws-security-best-practices-in-a-zero-trust-security-model-dem08-toronto-aws-summit*

[22] Christopher Null, “Infrastructure as code: The engine at the heart of DevOps”, in tech beacon, 2016; *Available: https://techbeacon.com/infrastructure-code-engine-heart-devops*

[23] Simon Lepik, “Building a Secure Software Supply Chain using Docker”, 2017; *Available: https://hdms.bsz-bw.de/frontdoor/deliver/index/docId/6321/file/20170830_thesis_final.pdf*

[24] Robert Anderson, “From Bare Metal to Private Cloud: Introducing DevSecOps and Cloud Technologies to Naval Systems”, 2018; *Available: https://etd.auburn.edu/bitstream/handle/10415/6318/RobertAnderson_MastersThesisV2.pdf?sequence=2&isAllowed=y*

[25] Samu Toimela, “Containerization of telco cloud applications”, 2017; available: *https://pdfs.semanticscholar.org/a11c/cd19a9d37083484b3e20832e2eb1e1c64c18.pdf*

[26] Ponemon Institute, “Cost of Data Center Outages”, 2011; Available from: *https://www.ponemon.org/local/upload/file/2011%20Cost_of_Data_Center_Outages.pdf*

[27] Christina Paule, Thomas F. Dullmann, and Andre van Hoorn, “Vulnerabilities in Continuous Delivery Pipelines? A Case Study”, University of Stuttgart and Novatec Consulting GmbH; A*vailable: https://www.novatec-gmbh.de/wp-content/uploads/Paper_Vulnerabilities-in-Continuous-Delivery-Pipelines-Case-Study_paper-Christina-Paule_20190416.pdf*

[28] Michale Hagara, “Evaluation of Infrastructure as a Code for Enterprise Automation”, University of Masaryakiana, Spring 2018; *Available: https://is.muni.cz/th/liwzz/IaC-Final-el.pdf*

[29] Evan Gilman, Doug Barth, “Zero Trust Networks”, July 2017; *Available: https://www.oreilly.com/catalog/errata.csp?isbn=9781491962190*

[30] Dr.Ealana James, “Action research write up”, *Available: https://www.youtube.com/watch?v=ybyErm7zABI*

Appendix 1 – Brakman Application Security report – Before Mitigation

```
Loading scanner.

== Brakeman Report ==

Brakeman Version: 4.1.0
ScanDate: 2019-01-20 16:16:28 +0200
Duration: 0.551165492 seconds
ChecksRun: BasicAuth, BasicAuthTimingAttack, ContentTag, CreateWith,
CrossSiteScripting, DefaultRoutes, Deserialize, DetailedExceptions, DigestDoS,
DynamicFinders, EscapeFunction, Evaluation, Execute, FileAccess,
FileDisclosure, FilterSkipping, ForgerySetting, HeaderDoS, I18nXSS, JRubyXML,
JSONEncoding, JSONParsing, LinkTo, LinkToHref, MailTo, MassAssignment,
MimeTypeDoS, ModelAttrAccessible, ModelAttributes, ModelSerialize,
NestedAttributes, NestedAttributesBypass, NumberToCurrency, PermitAttributes,
QuoteTableName, Redirect, RegexDoS, Render, RenderDoS, RenderInline,
ResponseSplitting, RouteDoS, SQL, SQLCVEs, SSLVerify, SafeBufferManipulation,
SanitizeMethods, SelectTag, SelectVulnerability, Send, SendFile,
SessionManipulation, SessionSettings, SimpleFormat, SingleQuotes,
SkipBeforeFilter, SprocketsPathTraversal, StripTags, SymbolDoSCVE,
TranslateBug, UnsafeReflection, ValidationRegex, WithoutProtection, XMLDoS,
YAMLParsing

== Overview ==

Controllers: 15
Models: 3
Templates: 27
Errors: 0
Security Warnings: 3

== Warning Types ==

Cross-Site Scripting: 3

== Warnings ==

Confidence: High
Category: Cross-Site Scripting
Check: CrossSiteScripting
Message: Unescaped parameter value
Code: params[:external_id]
File: app/views/payments/processing.haml
```

```
Line: 15

Confidence: Medium
Category: Cross-Site Scripting
Check: SanitizeMethods
Message: loofah gem 2.1.1 is vulnerable (CVE-2018-8048). Upgrade to 2.1.2
File: Gemfile.lock
Line: 161

Confidence: Medium
Category: Cross-Site Scripting
Check: SanitizeMethods
Message: rails-html-sanitizer 1.0.3 is vulnerable (CVE-2018-3741). Upgrade to
rails-html-sanitizer 1.0.4
File: Gemfile.lock
Line: 227
```

Appendix 2 – Brakman Application Security report – After Mitigation

```
Loading scanner.

== Brakeman Report ==

Brakeman Version: 4.3.0
ScanDate: 2019-03-05 12:05:28 +0200
Duration: 0.524165334 seconds
ChecksRun: BasicAuth, BasicAuthTimingAttack, ContentTag, CreateWith,
CrossSiteScripting, DefaultRoutes, Deserialize, DetailedExceptions, DigestDoS,
DynamicFinders, EscapeFunction, Evaluation, Execute, FileAccess,
FileDisclosure, FilterSkipping, ForgerySetting, HeaderDoS, I18nXSS, JRubyXML,
JSONEncoding, JSONParsing, LinkTo, LinkToHref, MailTo, MassAssignment,
MimeTypeDoS, ModelAttrAccessible, ModelAttributes, ModelSerialize,
NestedAttributes, NestedAttributesBypass, NumberToCurrency, PermitAttributes,
QuoteTableName, Redirect, RegexDoS, Render, RenderDoS, RenderInline,
ResponseSplitting, RouteDoS, SQL, SQLCVEs, SSLVerify, SafeBufferManipulation,
SanitizeMethods, SelectTag, SelectVulnerability, Send, SendFile,
SessionManipulation, SessionSettings, SimpleFormat, SingleQuotes,
SkipBeforeFilter, SprocketsPathTraversal, StripTags, SymbolDoSCVE,
TranslateBug, UnsafeReflection, ValidationRegex, WithoutProtection, XMLDoS,
YAMLParsing

== Overview ==

Controllers: 18
Models: 0
Templates: 23
Errors: 0
Security Warnings: 0
```

Appendix 3 – Bench Container Security Report – Before mitigation

```
# ------------------------------------------------------------------------
# Docker Bench for Security v1.3.4
#
# Docker, Inc. (c) 2015-
#
# Checks for common best-practices around deploying Docker containers in
production.
# Inspired by the CIS Docker Community Edition Benchmark v1.1.0.
# ------------------------------------------------------------------------

Initializing Sun Jan 11 07:45:46 UTC 2019

[INFO] 1 - Host Configuration
[WARN] 1.1  -  Ensure a separate partition for containers has been created
[NOTE] 1.2  -  the container host has been Hardened
[INFO] 1.3  -  Docker is up to date
[INFO]       Using 18.03.0, verify is it up to date as deemed necessary
[INFO]       Your OS may provide support and security maintenance for Docker
[INFO] 1.4  -  only trusted users are allowed to control Docker daemon
[INFO]       docker:x:991:user1,user2,…. (Real username censored)
[WARN] 1.5  -  auditing is configured for the Docker daemon
[WARN] 1.6  -  auditing is configured for Docker files and directories -
             Docker .service
             Docker .socket
             /etc/docker
             /etc/default/docker
             /etc/docker/daemon.json
             /usr/bin/docker-containerd
             /usr/bin/docker-runc
/var/lib/docker

[INFO] 2 - Docker daemon configuration
[WARN] 2.1  -  net traffic is restricted between containers on the default
bridge
[WARN] 2.2  -  the logging level is set to 'info'.
[PASS] 2.3  -  Docker is allowed to make changes to iptables
[WARN] 2.4  -  insecure registries are not used
[PASS] 2.5  -  aufs storage driver is not used
[INFO] 2.6  -  TLS authentication for Docker daemon is configured
[INFO]       Docker daemon not listening on TCP
[INFO] 2.7  -  the default ulimit is configured appropriately
[INFO]       Default ulimit doesn't appear to be set
```

```
[WARN] 2.8  - Enable user namespace support
[PASS] 2.9  -  the default cgroup usage has been confirmed
[PASS] 2.10  -  base device size is not changed until needed
[WARN] 2.11  -  that authorisation for Docker client commands is enabled
[WARN] 2.12  - centralised and remote logging is configured
[INFO] 2.13  -  operations on the legacy registry (v1) are Disabled
(Deprecated)
[PASS] 2.14  -  live restore is Enabled (Incompatible with swarm mode)
[WARN] 2.15  -  Userland Proxy is Disabled
[PASS] 2.16  -  daemon-wide custom seccomp profile is applied, if needed
[WARN] 2.17  -  experimental features are avoided in the production
[WARN] 2.18  -  containers are restricted from acquiring new privileges

[INFO] 3 - Docker daemon configuration files
[WARN]3.1  -  Ensure docker files ownership is set to root:root –
             docker.service
             docker.socket
             daemon.json
             /etc/docker
             /etc/default/docker
             /etc/docker/daemon.json
             /usr/bin/docker-containerd
             /usr/bin/docker-runc
             /var/lib/docker
             Docker server certificate
             Docker server certificate key file
             registry certificate file
             CA certificate file
             TLS certificate file
[WARN]3.2  -  docker file permissions are set to 644 –
             docker.service
             docker.socket
             daemon.json
             /etc/docker
             /etc/default/docker
             /etc/docker/daemon.json
             /usr/bin/docker-containerd
             /usr/bin/docker-runc
             /var/lib/docker
             Docker server certificate
             Docker server certificate key file
             registry certificate file
             CA certificate file
             TLS certificate file

[INFO] 4 - Container Images and Build File
[WARN] 4.1  -  Ensure a user for the container has been created
[WARN]       Running as root: xxx-production_zzz.1.sgclu7z9ruowop33e
[WARN]       ...
[WARN]       Running as root: vvv-production_sss.1.gayqupaitnqatjdmh
[NOTE] 4.2  -  that containers use trusted base images
[NOTE] 4.3  -  unnecessary packages are not installed in the container
```

```
[NOTE] 4.4  -  images are scanned and rebuilt to include security patches
[WARN] 4.5  -  Content trust for Docker is Enabled
[WARN] 4.6  -  HEALTHCHECK instructions have been added to the container image
[WARN]       No Healthcheck found: [alpine:3.8]
[INFO] 4.7  -  update instructions are not used alone in the Dockerfile
[NOTE] 4.8  -  setuid and setgid permissions are removed in the images
[INFO] 4.9  -  COPY is used instead of ADD in Dockerfile
[NOTE] 4.10  -  secrets are not stored in Dockerfiles
[NOTE] 4.11  -  verified packages are only Installed

[INFO] 5 - Container Runtime
[WARN] 5.1  -  Ensure AppArmor Profile is Enabled
[WARN]       No AppArmorProfile Found: xxx-production_zzz.1.sgclu7z9ruowop33e
...          ...
[WARN]       No AppArmorProfile Found: vvv-production_sss.1.gayqupaitnqatjdmh
[WARN] 5.2  -  Ensure SELinux security options are set, if applicable
[WARN]       No SecurityOptions Found: xxx-production_zzz.1.sgclu7z9ruowop33e
...          ...
[WARN]       No SecurityOptions Found: vvv-production_sss.1.gayqupaitnqatjdmh
[WARN] 5.3  -  Linux Kernel Capabilities are restricted within containers
[WARN] 5.4  -  privileged containers are not used
[WARN] 5.5  -  sensitive host system directories are not mounted on containers
[PASS] 5.6  -  ssh is not run within containers
[WARN] 5.7  -  privileged ports are not mapped within containers
[NOTE] 5.8  -  only needed ports are open on the container
[WARN] 5.9  -  the host's network namespace is not shared
[WARN]       running with networking mode 'host': xxx-production_zzz.1.sgclu
...          ...
[WARN]       running with networking mode 'host': vvv-production_sss.1.gafyq
[WARN] 5.10  -  memory usage for the container is limited
[WARN]       running without memory restrictions: xxx-production_zzz.1.sgclu
...          ...
[WARN]       running without memory restrictions: vvv-production_sss.1.gafyq
[WARN] 5.11  -  CPU priority is set appropriately on the container
[WARN]       running without CPU restrictions: xxx-production_zzz.1.sgclu
...          ...
[WARN]       running without CPU restrictions: vvv-production_sss.1.gafyq
[WARN] 5.12  -  the container's root filesystem is mounted as read-only
[WARN]       running with root FS mounted R/W: xxx-production_zzz.1.sgclu
...          ...
[WARN]       running with root FS mounted R/W: vvv-production_sss.1.gafyq
[WARN] 5.13  -  incoming container traffic is bound to a specific interface
[WARN]       Port being bound to wildcard IP: 0.0.0.0 in xxx-production
...          ...
[WARN]       Port being bound to wildcard IP: 0.0.0.0 in vvv-production
[WARN] 5.14  -  'on-failure' container restart policy is set to '5'
[WARN]       maximum retry count is not set to 5: xxx-production_zzz.1.sgclu
...          ...
[WARN]       maximum retry count is not set to 5: vvv-production_sss.1.gafyq
[PASS] 5.15  -  the host's process namespace is not shared
[PASS] 5.16  -  the host's IPC namespace is not shared
[INFO] 5.17  -  host devices are not directly exposed to containers
```

```
[INFO]      maximum retry count is not set to 5: xxx-production_zzz.1.sgclu
...          ...
[INFO]      maximum retry count is not set to 5: vvv-production_sss.1.gafyq
[INFO] 5.18  -  the default ulimit is overwritten at runtime, only if needed
[INFO]      Container no default ulimit override: xxx-production_zzz.1.sgclu
...          ...
[INFO]      Container no default ulimit override: vvv-production_sss.1.gafyq
[PASS] 5.19  -  mount propagation mode is not set to share
[PASS] 5.20  -  the host's UTS namespace is not shared
[PASS] 5.21  -  the default seccomp profile is not Disabled
[NOTE] 5.22  -  docker exec commands are not used with privileged option
[NOTE] 5.23  -  docker exec commands are not used with user option
[WARN] 5.24  -  cgroup usage is confirmed
[WARN]       Confirm cgroup usage: xxx-production_zzz.1.sgclu
...          ...
[WARN]       Confirm cgroup usage: vvv-production_sss.1.gafyq
[WARN] 5.25  -  container is restricted from acquiring additional privileges
[WARN]       Privileges not restricted: xxx-production_zzz.1.sgclu
...          ...
[WARN]       Privileges not restricted: vvv-production_sss.1.gafyq
[WARN] 5.26  -  container health is checked at runtime
[WARN]       Health check not set: xxx-production_zzz.1.sgclu
...          ...
[WARN]       Health check not set: vvv-production_sss.1.gafyq
[INFO] 5.27  -  docker commands always get the latest version of the image
[WARN] 5.28  -  PIDs cgroup limit is used
[WARN]       PIDs limit not set: xxx-production_zzz.1.sgclu
...          ...
[WARN]       PIDs limit not set: vvv-production_sss.1.gafyq
[PASS] 5.29  -  Docker's default bridge docker0 is not used
[PASS] 5.30  -  the host's user namespaces is not shared
[PASS] 5.31  -  the Docker socket is not mounted inside any containers
[INFO] 6 - Docker Security Operations
[INFO] 6.1  - Avoid image sprawl
[INFO]       There are currently: 78 images
[INFO] 6.2  - Avoid container sprawl
[INFO]       There are currently a total of 180 containers, with 180 of them
currently running
[INFO] 7 - Docker Swarm Configuration
[PASS] 7.1  -  swarm mode is not Enabled, if not needed
[PASS] 7.2  -  the minimum number of manager nodes have been created in a
swarm
[PASS] 7.3  -  swarm services are bound to a specific host interface
[WARN] 7.4  -  data exchanged between containers are encrypted on different
nodes on the overlay network
[WARN]       Unencrypted overlay network: xxx-production_zzz.1.sgclu
...          ...
[WARN]       Unencrypted overlay network: vvv-production_sss.1.gafyq
[INFO] 7.5  -  Docker's secret management commands are used for managing
secrets in a Swarm cluster
[WARN] 7.6  -  swarm manager is run in auto-lock mode
[NOTE] 7.7  -  swarm manager auto-lock key is rotated periodically
```

```
[INFO] 7.8  -  node certificates are rotated as appropriate
[INFO] 7.9  -  CA certificates are rotated as appropriate
[INFO] 7.10  -  management plane traffic has been separated from data plane
traffic

[INFO] Checks: 105
[INFO] Score: -8
```

Appendix 4 – Bench Container Security Report – After Mitigation

```
# ------------------------------------------------------------------------
# Docker Bench for Security v1.3.4
#
# Docker, Inc. (c) 2015-
#
# Checks for common best-practices around deploying Docker containers in production.
# Inspired by the CIS Docker Community Edition Benchmark v1.1.0.
# ------------------------------------------------------------------------

Initializing Sun Jan 11 07:45:46 UTC 2019

[INFO] 1 - Host Configuration
[PASS] 1.1  -  Ensure a separate partition for containers has been created
[NOTE] 1.2  -  the container host has been Hardened
[INFO] 1.3  -  Docker is up to date
[INFO]       Using 18.09.0, verify is it up to date as deemed necessary
[INFO]       Your OS may provide support and security maintenance for Docker
[INFO] 1.4  -  only trusted users are allowed to control Docker daemon
[INFO]       docker:x:991:user1,user2,…. (Real username censored)
[PASS] 1.5  -  auditing is configured for the Docker daemon
[PASS] 1.6  -  auditing is configured for Docker files and directories -
             docker.service
             docker.socket
             /etc/docker
             /etc/default/docker
             /etc/docker/daemon.json
             /usr/bin/docker-containerd
             /usr/bin/docker-runc
/var/lib/docker

[INFO] 2 - Docker daemon configuration
[PASS] 2.1  -  net traffic is restricted between containers on the default bridge
[PASS] 2.2  -  the logging level is set to 'info'.
[PASS] 2.3  -  Docker is allowed to make changes to iptables
[PASS] 2.4  -  insecure registries are not used
[PASS] 2.5  -  aufs storage driver is not used
[INFO] 2.6  -  TLS authentication for Docker daemon is configured
[INFO]       Docker daemon not listening on TCP
[INFO] 2.7  -  the default ulimit is configured appropriately
[INFO]       Default ulimit doesn't appear to be set
```

```
[PASS] 2.8  - Enable user namespace support
[PASS] 2.9  -  the default cgroup usage has been confirmed
[PASS] 2.10  -  base device size is not changed until needed
[PASS] 2.11  -  that authorisation for Docker client commands is enabled
[PASS] 2.12  - centralised and remote logging is configured
[INFO]  2.13  -  operations  on  the  legacy  registry  (v1)  are  Disabled
(Deprecated)
[PASS] 2.14  -  live restore is Enabled (Incompatible with swarm mode)
[WARN] 2.15  -  Userland Proxy is Disabled
[PASS] 2.16  -  daemon-wide custom seccomp profile is applied, if needed
[PASS] 2.17  -  experimental features are avoided in the production
[PASS] 2.18  -  containers are restricted from acquiring new privileges

[INFO] 3 - Docker daemon configuration files
[PASS] 3.1  -  Ensure docker files ownership is set to root:root –
             docker.service
             docker.socket
             daemon.json
             /etc/docker
             /etc/default/docker
             /etc/docker/daemon.json
             /usr/bin/docker-containerd
             /usr/bin/docker-runc
             /var/lib/docker
             Docker server certificate
             Docker server certificate key file
             registry certificate file
             CA certificate file
             TLS certificate file
[PASS] 3.2  -  docker file permissions are set to 644 –
             docker.service
             docker.socket
             daemon.json
             /etc/docker
             /etc/default/docker
             /etc/docker/daemon.json
             /usr/bin/docker-containerd
             /usr/bin/docker-runc
             /var/lib/docker
             Docker server certificate
             Docker server certificate key file
             registry certificate file
             CA certificate file
             TLS certificate file

[INFO] 4 - Container Images and Build File
[PASS] 4.1  -  Ensure a user for the container has been created
[PASS] 4.2  -  that containers use trusted base images
[NOTE] 4.3  -  unnecessary packages are not installed in the container
[NOTE] 4.4  -  images are scanned and rebuilt to include security patches
[PASS] 4.5  -  Content trust for Docker is Enabled
[PASS] 4.6  -  HEALTHCHECK instructions have been added to the container image
```

```
[INFO] 4.7  -  update instructions are not used alone in the Dockerfile
[NOTE] 4.8  -  setuid and setgid permissions are removed in the images
[INFO] 4.9  -  COPY is used instead of ADD in Dockerfile
[PASS] 4.10  -  secrets are not stored in Dockerfiles
[NOTE] 4.11  -  verified packages are only Installed

[INFO] 5 - Container Runtime
[WARN] 5.1  -  Ensure AppArmor Profile is Enabled
[WARN]       No AppArmorProfile Found: xxx-production_zzz.1.sgclu7z9ruowop33e
...          ...
[WARN]       No AppArmorProfile Found: vvv-production_sss.1.gayqupaitnqatjdmh
[PASS] 5.2  -  Ensure SELinux security options are set, if applicable
[PASS] 5.3  -  Linux Kernel Capabilities are restricted within containers
[PASS] 5.4  -  privileged containers are not used
[PASS] 5.5  -  sensitive host system directories are not mounted on containers
[PASS] 5.6  -  ssh is not run within containers
[PASS] 5.7  -  privileged ports are not mapped within containers
[NOTE] 5.8  -  only needed ports are open on the container
[PASS] 5.9  -  the host's network namespace is not shared
[PASS] 5.10  -  memory usage for the container is limited
[PASS] 5.11  -  CPU priority is set appropriately on the container
[PASS] 5.12  -  the container's root filesystem is mounted as read-only
[WARN] 5.13  -  incoming container traffic is bound to a specific interface
[WARN]       Port being bound to wildcard IP: 0.0.0.0 in xxx-production
...          ...
[WARN]       Port being bound to wildcard IP: 0.0.0.0 in vvv-production
[PASS] 5.14  -  'on-failure' container restart policy is set to '5'
[PASS] 5.15  -  the host's process namespace is not shared
[PASS] 5.16  -  the host's IPC namespace is not shared
[PASS] 5.17  -  host devices are not directly exposed to containers
[PASS] 5.18  -  the default ulimit is overwritten at runtime, only if needed
[PASS] 5.19  -  mount propagation mode is not set to share
[PASS] 5.20  -  the host's UTS namespace is not shared
[PASS] 5.21  -  the default seccomp profile is not Disabled
[NOTE] 5.22  -  docker exec commands are not used with privileged option
[NOTE] 5.23  -  docker exec commands are not used with user option
[PASS] 5.24  -  cgroup usage is confirmed
[PASS] 5.25  -  container is restricted from acquiring additional privileges
[PASS] 5.26  -  container health is checked at runtime
[INFO] 5.27  -  docker commands always get the latest version of the image
[PASS] 5.28  -  PIDs cgroup limit is used
[PASS] 5.29  -  Docker's default bridge docker0 is not used
[PASS] 5.30  -  the host's user namespaces is not shared
[PASS] 5.31  -  the Docker socket is not mounted inside any containers

[INFO] 6 - Docker Security Operations
[INFO] 6.1  - Avoid image sprawl
[INFO]       There are currently: 78 images
[INFO] 6.2  - Avoid container sprawl
[INFO]       There are currently a total of 180 containers, with 180 of them
currently running
```

```
[INFO] 7 - Docker Swarm Configuration
[PASS] 7.1  -  swarm mode is not Enabled, if not needed
[PASS] 7.2  -  the minimum number of manager nodes have been created in a swarm
[PASS] 7.3  -  swarm services are bound to a specific host interface
[PASS] 7.4  -  data exchanged between containers are encrypted on different nodes on the overlay network
[INFO] 7.5  -  Docker's secret management commands are used for managing secrets in a Swarm cluster
[PASS] 7.6  -  swarm manager is run in auto-lock mode
[PASS] 7.7  -  swarm manager auto-lock key is rotated periodically
[INFO] 7.8  -  node certificates are rotated as appropriate
[INFO] 7.9  -  CA certificates are rotated as appropriate
[INFO] 7.10  -  management plane traffic has been separated from data plane traffic

[INFO] Checks: 105
[INFO] Score: 18
```