GALGOTIAS UNIVERSITY

Greater Noida, Uttar Pradesh

PROJECT REPORT
ON

APPLICATION OF MACHINE LEARNING IN DEVOPS

Submitted in Partial fulfillment of the Requirements for the Degree of Bachelor of Technology in Computer Science & Engineering with Specialization in Artificial Intelligence and Machine Learning

By

Ayushi Kulshrestha	**19SCSE1180131**
Kriti Sharma	**19SCSE1010334**

Under the Guidance

of

Prof. Dr. Anupam Sharma

CERTIFICATE

Certified that the project work entitled **“ APPLICATION OF MACHINE LEARNING IN DEVOPS”** carried out by **Ms.Ayushi Kulshrestha**, ADM. NO. **19SCSE1180131 and Ms.Kriti Sharma,** ADM NO. **19SCSE1010334,** bonafide students of CMR Institute of Technology, in partial fulfillment for the award of **Bachelor of Engineering** in Computer Science and Engineering of the Galgotias University, Greater Noida, Uttar Pradesh.It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library.

The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

.

DECLARATION

We, the students of Computer Science and Engineering, CMR Institute of Technology, Bangalore declare that the work entitled "**APPLICATION OF MACHINE LEARNING IN DEVOPS**" has been successfully completed under the guidance of Dr. Anupam Sharma, Computer Science and Engineering Department,Galgotias University. This dissertation work is submitted in partial fulfillment of the requirements for the award of Degree of Bachelor of Engineering in Computer Science and Engineering during the academic year 2021- 2022.

Place:

Date:

Team members:

AYUSHI KULSHRESTHA (19SCSE1180131)

KRITI SHARMA (19SCSE1010334)

ABSTRACT

This project will deal with application of machine learning in Devops. The underlying notion for this is that devops teams do not closely look at data and instead focus on thresholds that satisfy a certain condition for action. When they do this, DevOps teams are unable to utilize the large data that they collect. They entirely focus on outliers which can trigger alerts of certain problems but do provide the much needed information. When ML gets induced in DevOps processes, one can be rest assured of the data getting monitored and analyzed at a continuous pace. ML applications look at the data along with predictive analytics to draw out meaningful insights.

ACKNOWLEDGEMENT

We take this opportunity to express my sincere gratitude and respect to Galgotias University, Greater Noida, Uttar Pradesh for providing me a platform to pursue my studies and carry out my final year project.

We would like to thank Prof.**Dr. Anupam Sharma,** Department of Computer Science and Engineering,Galgotias University, Greater Noida, who has been a constant support and encouragement throughout the course of this project.

We consider it a privilege and honor to express our sincere gratitude to **Dr.Balbindar Kaur ,** Professor **of** Department of SCSE , for the valuable guidance throughout the tenure of this project.

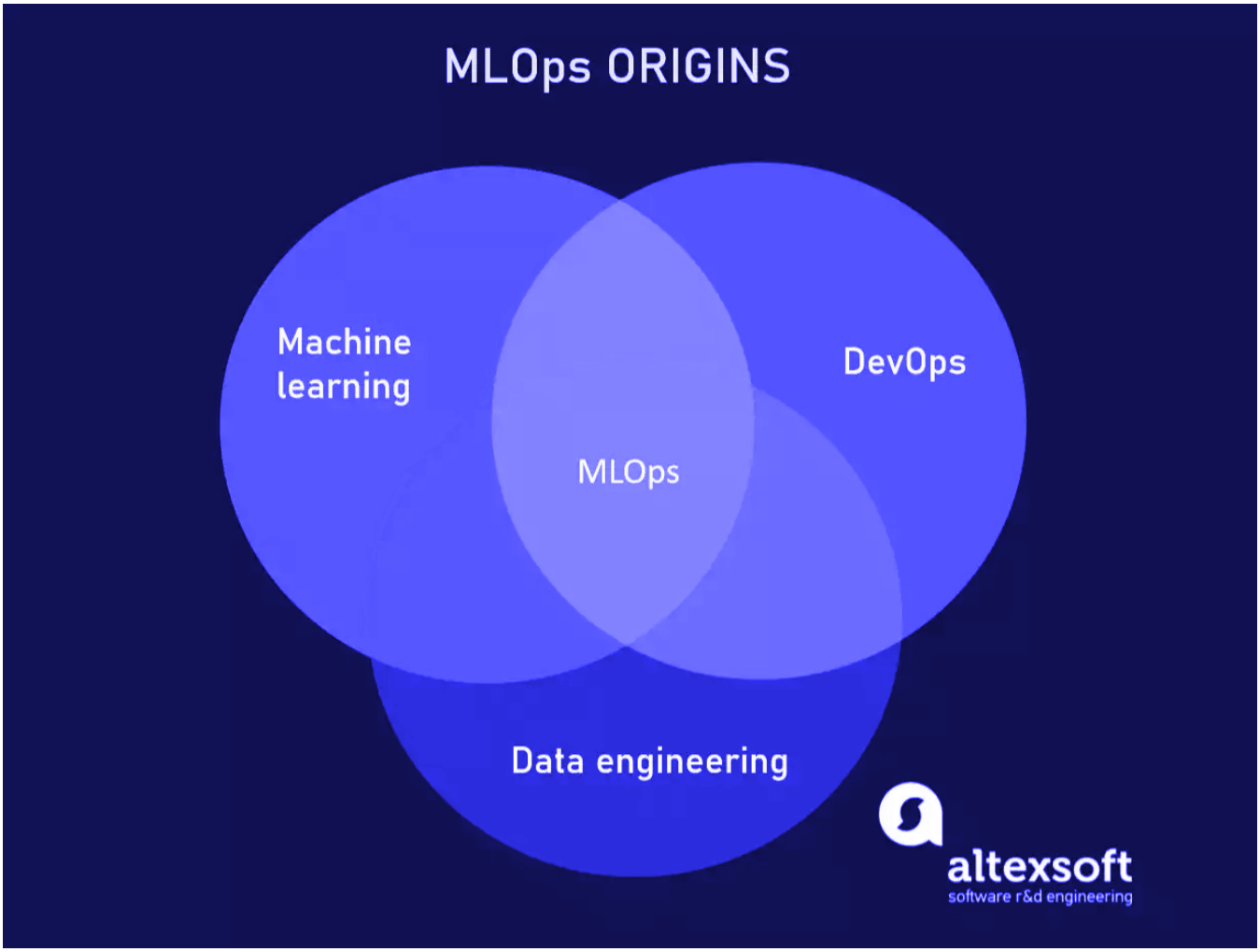
We also extend my thanks to all the faculty of Computer Science and Engineering who directly or indirectly encouraged me.

Finally, we would like to thank my parents and friends for all their moral support they have given me during the completion of this work.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION



In Data science world Machine learning is core approach to resolve the important life problems today's big data world ML plays vital role to produce best solutions to the customer Nowadays Machine Learning is buzz word but why the proportion of AI models created but not put in production environment is quite 90% with massive investment in data science teams platforms and infrastructure the quality of ai projects is dramatically increasing together with the no of missed opportunities unfortunately most projects not showing the business needs business introducing new risks that require

to be managed single technology won't help to unravel the issue today industry need machine learning with devops. MLops is solution to any or all problems in AI world .MLops delivers the capabilities that data science and it ops teams must to work together to deploy monitor and manage machine learning model in production environment and to manipulate their use in production environment MLops brings the most effective of repetitious development involved in training machine learning models and scalable and manageable model deployment it's currently missing puzzle within the enterprise AI world Machine learning operations (MLOps) is DevOps for machine learning processes Mlops helps data scientists to cooperate and increase the speed of delivery and quality of development models through monitoring affirmation and governance of machine learning models. MLOps supports the data science life cycle even as DevOps supports the application development life cycle.
As such, MLOps is predicted on the principles of DevOps. The goal of adopting MLOps is to push more efficient experimentation that will result in faster development and deployment of models. Devops is a combination of Development and operations methodology that brings developers and operators work together DevOps life-cycle devops automate one or more phases of devops life cycle Coding, building, testing, releasing, configuring, and monitoring.

- **Coding phase:** this phase includes development of code developers who write the code and push the code into source code management tools like github, gitlab etc.
- **Building phase:** This phase automatically creates the environment in step with the wants as an example team create Docker image for particular Machine learning model.
- **Testing phase:** This phase includes continuous testing tools, teams develop a separate environment for testing the code and it gives the standard of the code various testing techniques included in this phase.
  They are
  - 1. Unit testing
  - 2. Integration testing
  - 3. Configuration testing
  - Performance testing

- **Releasing phase:** this phase consists of releasing plan and automation. The team will decide how to release the product into the market.
- Configuring phase: this phase consists of automatic configuration management with the help of infrastructure as the code team will create the scripts according to the requirement script and automatically launch everything. As an example they launch running environments, databases etc.
- **Monitoring phase:** this phase continuously monitors the product whenever the production environment goes down; it notifies the developer about what number users are using the merchandise at a particular point of time.
  Pipeline: To achieve CI/CD developers create "pipelines", which demonstrate how to automatically build, test, release and configure software release. The pipeline is an event that tells how to execute the steps in sequential manner. If any step fails it will stop the entire pipeline and it gives feedback to the developer. CI/CD supports small commits and a large number of releases per day.

Phase	Tools
Build	Docker, Gradle.
Test	pytest, Cypress.
Release	Jenkins, Flux.

Configure	Terraform, Ansible.
Monitor	Sentry, Prometheus.

Creating CI/CD pipelines consists of different tools that are created to solve different tasks of devops lifecycle. Tools that help to run CI/CD pipeline on host OS are Github actions and Travis, tools that help to run CI/CD pipeline on own machines are Jenkins or argoCD. These tools provide a configuration language to run steps and tools as a pipeline for every single release there is a master plan to roll back the previous version in case the updated version fails.

An easy solution for rollback is running the previous version through CI/CD pipeline. A container is built from an image which contains a set of instructions to build the container with the help of a Docker file. We can create a container image, an image containing an operating system and software to run particular services container runs in a container run time, a process which manages the lifecycle of a container.

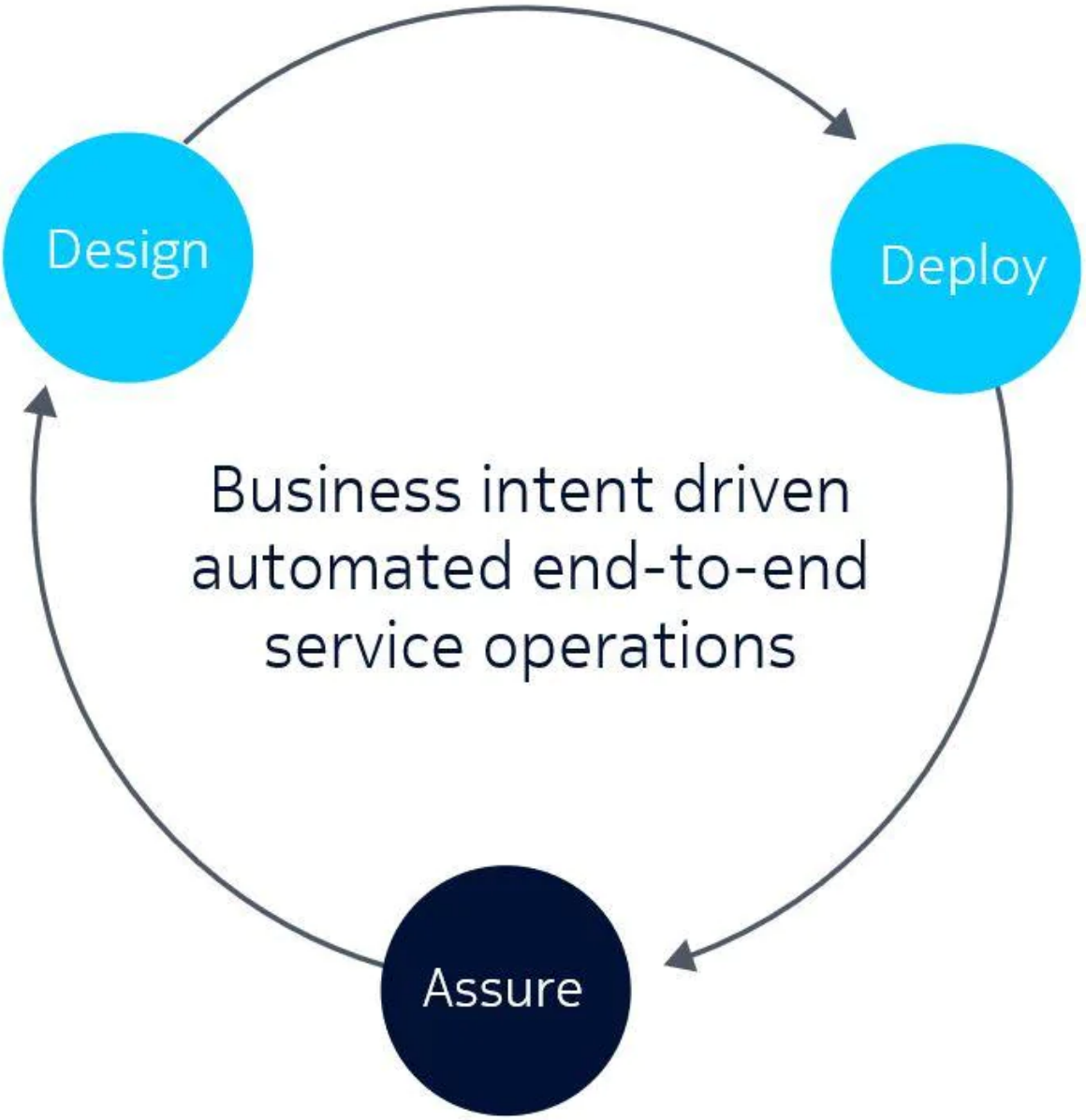
With the help of container technology we can launch the operating system in one second. Unexpectedly if the production environment goes down, container technology launches quickly as compared to virtualization. In a dynamic and large scale environment we have to change machines and services frequently. It is impossible to provide high availability. Container orchestration provides highly-available and robust service systems.

Nokia Orchestration Center automates orchestration of end-to-end services, including 5G slice-based services, in cross-domain, multi-vendor and multi-technology networks. Together with Nokia Assurance Center, the cloud-native microservices-based Orchestration Center module enables round-trip operations automation as part of Nokia's Digital Operations Center.

- Design and deploy new market driven digital services based on business-intent quickly and efficiently leveraging Bell Labs AI & ML technology
- Single-pane-of-glass and intuitive UI design simplifies complexity by providing a unified view of different network domains leveraging containerized, virtualized, and physical technologies
- Automation of both service fulfillment and multi-domain orchestration functions results in significant reductions in the time-to-value for new services

As service providers strive to increase their agility to support market requirements in the 5G era, a tighter closed-loop operating environment is required. Orchestration Center supports this objective on two levels:

- Market facing agility: Service providers will be forging B2B partnerships with vertical industrial players and Service Level Agreements will become an important currency for those relationships. Service providers require orchestration systems that can adopt and support contracted SLAs that are "designed-in" to the services during the commercial negotiations with the B2B partners.
- Virtualized and containerized infrastructure: To control their costs in an ever-growing demand for capacity, service providers are well along their transition to leverage cloud and virtualization techniques. This creates a multi-fold increase in network and infrastructure complexity—more moving parts—and this cannot be effectively automated at scale on legacy orchestration systems.



Modules

Orchestration Center leverages a modular concept with user/role-based interfaces for service and resource visibility, service design and service deployment.

DesignHub

DesignHub is used to create service compositions from re-usable building blocks and to publish service designs to northbound systems for consumption.

OrderHub

OrderHub provides 360° visibility and management of in-progress, service modifiable orders with an intuitive portal and fully amendable templates.

IntelligenceHub

IntelligenceHub capabilities include advanced root-cause analysis, automated capacity management, insights for automation, topology views and machine-learning algorithms.

Service Orchestration

Service Orchestration is the main sub-module of Orchestration Center. It automates and optimizes the service delivery utilizing AI and ML to decompose business-intent and deploy it accordingly to the network.

Service Design

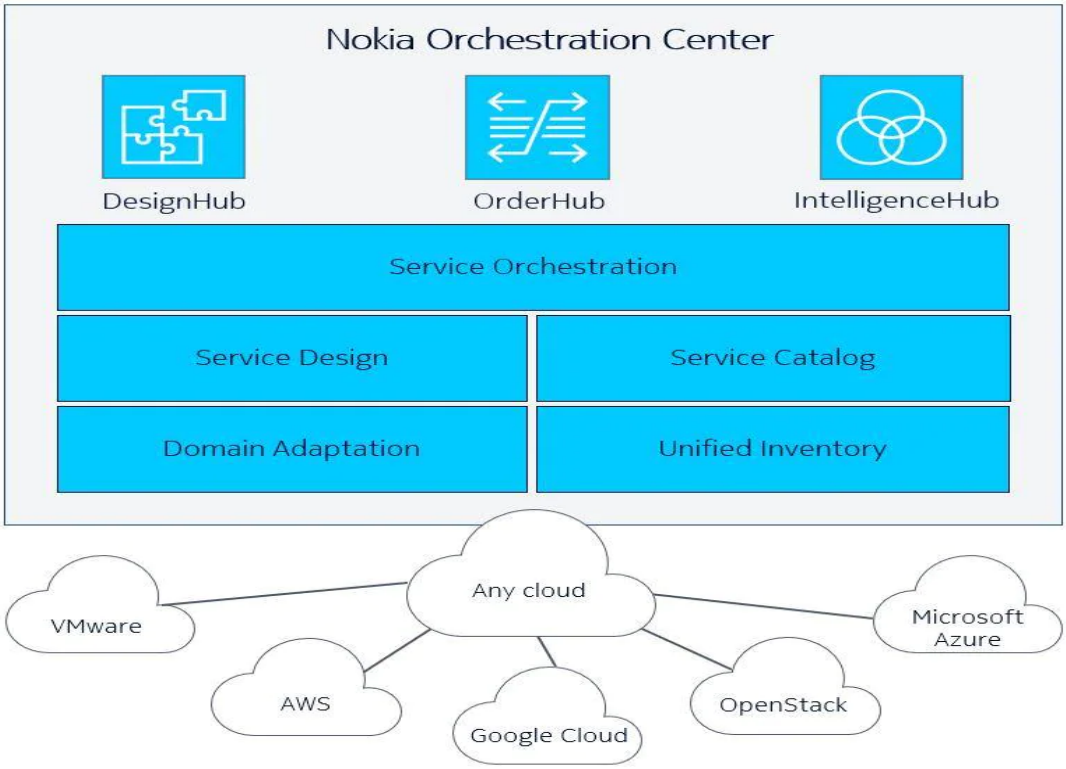
The Service Design sub-module provides service modelling capabilities that enable business-intent driven orchestration. Services can be composed with re-usable components that can be stored in the service catalog for further use.

Service Catalog

The Service Catalog manages and stores the designed service components with related technical service composition and workflow models. It also enables services to be published to CRM/CPQ systems for consumption when they are ready.

Domain Adaptation

The Domain Adaptation layer provides an abstracted view of the underlaying network domains towards service orchestration. This hides service design and orchestration complexity and enables end-to-end, high-level service design and operation.



LITERATURE REVIEW:

In literature, we can find studies which shows how to apply machine learning with devops Machine learning with operations (MLOPS) using tools and platforms to automate machine learning models the authors develop continuous integration for machine learning to allow user to provide wide range of integration in order to achieve continuous improvement one of the most critical challenges are data collection data extraction data cleaning.

Authors create application life cycle model based on MLOPs to optimize the manufacturing process the study shows how to apply Devops CI/CD pipelines with machine learning applications several researchers Karamitsos,Virmani, Erich have agreed that Agile transformation is essential to improve the efficiency of the companies, MLops fills the gap between business users and development teams they highlight the devops principles and guidelines to adapt continuous integration and continuous deployment which results increment in development process and improve the quality.

Many applications use machine learning techniques nowadays but which is not enough to produce great results when we integrate with devops. It gives great efficiency in any field like healthcare safety etc. The main goal of devops is to create cross functional teams where both operational and development tools work together. The entire goal of devops is to improve the business value in the IT industry, it produces best results in an agile world with the help of continuous integration and continuous deployment. Devops overcome the hurdles between operations and development and they collaborate both machine learning and devops, Machine learning+Devops (MLOPS).

1.5 Methodology

The industry Standard Process for Data processing(CRISP-DM)could be a process model with six phases that describes the data science life cycle.it helps to implement machine learning projects It was published in 1999 it's one among the foremost common methodology for data science projects. CRISP DM Methodology has six steps

- **Business understanding phase:** during phase specialized to understand the objectives and needs of the project.

- **Data understanding phase**: This phase contains initial data for experimental analysis this phase focus identify collect and analyzes the information to succeed in the project goal

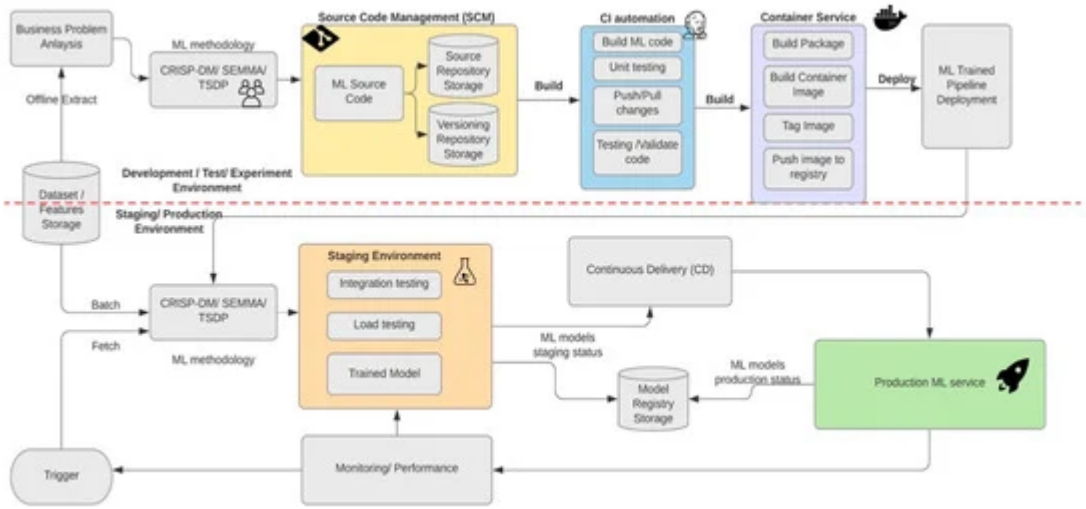
- **Data preparation phase:** during this phase data preparation takes place steps included in this process are feature extraction, data cleaning, data reduction, data selection, and transformation.
- **Modeling Phase:** This phase selects the machine learning model based on the requirement. Evaluation Phase: This phase looks at which model connects the business requirement. And also a review decides whether the business requirement is achieved or not.

- **Deployment Phase:** This final phase contains four tasks :

1. Plan deployment.
2. Plan monitoring and maintenance.
3. Produce final report.
4. Review project.

Machine learning automate pipeline with CI/CD The main purpose of this approach is continuous training and testing the machine learning model with the help of pipeline the term mlops combines various principles of Continuous integration and continuous deployment to automate the machine learning pipeline

The process of automatic model retrain in machine learning is possible with the 2 elements of devops such as 1) continuous integration 2) continuous deployment the automated CI/CD helps the data scientists in feature engineering model architecture and hyper parameters data scientists implement lot of things like building the model ,push the code into github, build deployment environment etc.



Main elements required for the implementation of machine learning CI/CD pipeline

1) Business problem analysis
2) Dataset features and storage
3) ML methodology
4) CI components
5) CD components
6) Automated ML triggering
7) Model registry storage
8) Monitoring and performance
9) Production ML service

First of all,we understand the business problem and How to solve this business problem using technologies, dataset features and storage to collect the features related to the problem, In machine learning, creating models is not a big deal. Collecting the right features is a major problem in today's machine learning world. Accuracy always depends on the right features.

Machine Learning methodology: This is the major step in mlops pipeline during the analysis of data we use the methodology CRISP methodology is one of the best and common methodology for Automated machine learning CI/CD pipeline Continuous integration: In this step build the code run the machine learning models. outcomes of this stage is pipeline components In machine learning creating the code is not big deal but training the machine learning models requires a lot of resources and manpower with the help of continuous integration we can do this things with minimum resources and minimal efforts Requirements for Continuous integration
1.Source code management tools like Git, github etc.

2.Insertion/deletion takes place in a repository that automatically updates and deploys in the production environment.

3.Run the machine learning code

4.Test and validate the code

5. Build the container image

6. push the container image into repository Continuous delivery: outputs of

Continuous integration pipeline components deployed in the production/staging environment the output of this step is testing machine learning model.

Jenkins is one of the most famous open source tool for continuous deployment it is written in java language this tools works with SCM tools like github The components of continuous deployment

1) Production environment: First we have to push machine learning models into the production environment. It contains all the required decencies to run that machine learning code output of this stage is testing the machine learning model.

2) Model registry storage: output of the production environment is testing machine learning. We have to put that machine learning model into the model registry storage.

3) Automated trigger: If there is any change in code or commit in repo trigger automatically fires and they will initiate a new build output of this updated machine learning model pushed in the production environment.

4) Performance monitoring: This is one of the most important steps for data scientists. Performance monitoring contains a lot of considerations like how much resources are used, efficiency of the model, throughput, and features available to the users.

5) Monitoring resources: In this step continuously observe the resources of the system like RAM CPU storage etc. a lot of resource monitoring tools available in the market like zabbix windows task manager.

Application of Machine Learning with Traffic Monitoring to Intrusion Detection in Kubernetes Deployments

Irene Ann Tony, Masters in Computer Science University of Dublin, Trinity College, 2021

Supervisor: Dr. Stefan Weber

Kubernetes has many methods to detect an attacker attempting to attack a Cluster such as firewalls and an Ingress API. However, it does not have any built-in solutions to detect whether an attacker is already inside. There are existing solutions involving the identification of malicious traffic and they have inspired certain aspects of the solution proposed in this dissertation.

This dissertation focuses on the ways to detect malicious activity inside the Kubernetes Cluster by monitoring the internal network traffic and detecting suspicious traffic with the help of machine learning. This research focuses on TCP and UDP packets between Pods and examines them to determine whether they are malicious or not. The proposed solution involves capturing all the traffic within the Node including the communications between Pods. The solution model sits inside a Node within a Pod and is implemented by passing the captured internal traffic through a neural network that is trained to detect different characteristics of malicious packets and flag the packets that seem suspicious.

This research provides a foundation for advanced studies in detecting malicious activity within a Kubernetes Cluster using traffic monitoring and machine learning. The proposed solution can be extended to learn to detect many different types of malicious activity.

Contents

List of Tables

List of Figures

Abbreviations

API Application Programming Interface ARP Address Resolution Protocol

CNN Convolutional Neural Network CPU Central Processing Unit

CRI-O Container Runtime Interface using Open Container Initiative (OCI) DDoS Distributed Denial of Service

DNS Domain Name System

Etcd /etc Distributed

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure IDE Integrated Development Environment IP Internet Protocol

ML Machine Learning

MLP Multilayer Perceptron

NDP Neighbor Discovery Protocol

RBAC Role Based Access Control

REST Representational State Transfer

SSH Secure Shell

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

Intrusion detection in Kubernetes revolves around the act of identifying an attacker who has bypassed the security protocols of a system and gained access to resources inside. Finding a solution that can execute this action can impact many users of the Kubernetes platform. This dissertation focuses on how these attackers can be identified with the use of traffic monitoring and machine learning.

This section provides an overview of the research topic and the motivation behind exploring it further. It also outlines the research questions this dissertation aims to answer.

1.1 Overview

During earlier times, virtual machines were the main way to run applications in different operating systems. This was implemented with the use of hypervisor-based virtualised environments that utilised the host computer’s hardware (Panagiotis (2020)). A more independent and mobile solution for virtualisation involved the introduction of Containers. Containerisation involves the virtualisation of operating systems in order to run an application, along with all its dependencies, in an isolated environment called a container (IBM (2019)). Containerisation platforms, such as Docker, are used to build, deploy and manage containerised applications (IBM (2020)).

In order to manage many Docker containers along with multiple applications, a more advanced platform was needed. Kubernetes is a container orchestrator for containerisation platforms like Docker. It manages multiple containers with applications across many hosts (Kubernetes (2021b)). All these new advancements have introduced

security concerns and challenges.

In recent years, there have been multiple attacks made to Kubernetes Clusters hosting applications for enterprises. Many attacks occur due to misconfigurations and not activating certain security features in the Kubernetes Cluster. A famous example is an attack on Capital One (Taylor (2020)) in 2019, which was caused by a misconfiguration in the Kubernetes Cluster that weakened the firewall. More complex attacks occur after the attacker gains access to the Cluster and plants malicious software scripts within a container image or Pod. An attack on Docker Hub (Taylor (2020)), in 2019, involved the attackers planting malicious images and whoever ran those images would become compromised and deploy cryptocurrency miners as means for the attacker to mine cryptocurrency.

Another famous attack is the compromisation of Tesla's Kubernetes Cluster where the attackers remained incognito even after gaining access to the Cluster. They ran scripts that used less CPU and listened on a nonstandard port to avoid being detected by internal security monitoring features (Taylor (2020)).

A similar attack is the Kinsing Malware attack, where attackers exploited a misconfigured Docker API port and ran a container that contained malicious software. The techniques used by the attackers involved exploiting the open port and using evasion tactics and lateral movement to deploy the crypto miner (Singer (2020)).

Attackers are becoming more aware of the ways malware is being detected and have adapted to becoming more unnoticeable. New ways of closely monitoring internal communications of a Kubernetes Cluster can benefit anti-malware software in detecting these attackers.

1.2 Motivation

Kubernetes is a popular platform used to manage numerous applications and as section 1.1 has highlighted, attackers are targeting Kubernetes Clusters and utilizing resources to run their processes such as crypto mining. Although Kubernetes has security protocols implemented to prevent attackers from entering the Cluster, it is evident that, in the case of an attacker gaining access, the structure is not equipped with processes that can immediately detect the intruder. The motivation behind this research is to explore intrusion detection in Kubernetes with the help of network monitoring and machine

1.3 Research Questions

The following are the research questions explored in this dissertation:

- What is an effective method to monitor network traffic in a Kubernetes Cluster?
- How can monitoring network traffic contribute to intrusion detection within a Kubernetes Cluster?

  The topics discussed in this dissertation will help form the answers to these questions.

1.4 Structure & Contents

This dissertation will explore Kubernetes, its internal structure, how its network can be monitored and how machine learning can contribute to identifying intruders using the monitored traffic.

- Chapter 1 provides an overview of Kubernetes and why intrusion detection is vital for its security. It also covers how this dissertation aims to explore a solution that can contribute to intrusion detection in Kubernetes along with the research questions proposed for this dissertation.
- Chapter 2 explores Kubernetes along with both its structural and networking architecture. In addition to that, it also explores machine learning and the different approaches that could benefit this research. Moreover, it outlines and analyses the existing work that is related to the proposed solution.
- Chapter 3 outlines the design of the proposed solution and explains the reasoning behind the chosen components that make up the overall solution.
- Chapter 4 explains how the proposed solution is implemented using the chosen tools.
- Chapter 5 evaluates the proposed solution by calculating its reliability, comparing it to the existing solutions and explaining its advantages and disadvantages.
- Chapter 6 concludes this dissertation by providing the answers to the research questions and discusses how this solution can be further improved in the future, along with final remarks.

Chapter 2

State of the Art

This section explains Kubernetes, its general and network architecture and the different types of security measures that are put in place within the platform in order to detect and prevent malicious attacks. It provides an overview of machine learning and introduces the concept of neural networks. It also outlines the existing research and solutions proposed to tackle the gaps within the security measures of Kubernetes.

2.1 Kubernetes

Kubernetes is an open-source container orchestration system that is used to deploy, scale and manage application containers automatically. Developed by Google, Kubernetes is a platform that is only accessible via a domain-specific Representational State Transfer Application Programming Interface (REST API) and supports a wider range of clients by applying higher-level versioning, validation, semantics, and policy (Burns et al. (2016)). The system works with many containerising tools including Docker. It helps manage containerised applications in different deployment environments such as physical virtual or cloud environments. It offers many features such as high availability, scalability and disaster recovery.

2.1.1 Architecture of Kubernetes

The basic architecture of Kubernetes is a Cluster, shown in figure 2.1, that is made up of a Master Node and multiple Worker Nodes where each Node has a kubelet process running in it. Kubelet is a process that allows the Nodes to communicate with each other within the Cluster.

Figure 2.1: A high level view of a Kubernetes Cluster consists of a Master Node and multiple Worker Nodes. The Master Node contains the API Server, Controller Manager, Scheduler and the Etcd, all of which are components that are needed for the management of the Cluster and its Nodes. The Worker Nodes each contain a Kubelet, a Kube-Proxy, a Container Runtime and one or more Pods that host containerised applications. All Nodes communicate with each other using a virtual network within the Cluster.

2.1.1.1 Master Node

The Master Node is also called the Control Plane and it contains certain components that are needed to control and manage the entire Cluster (Panagiotis (2020)). These components are the Application Programming Interface (API) Server, the Scheduler, the Controller Manager and the /etc Distributed (Etcd). These components work together to manage the entire Kubernetes Cluster.

- The API Server is a container that is the entry point to the entire Cluster. It exposes the Kubernetes API, a RESTful web server, that is the main point of interaction between the outside and the Cluster. Communication between all other components also occurs through the API Server.
- The Scheduler is a component that monitors the API Server in order to assign newly created Pods to a suitable Node within the Cluster. It chooses a Node based on the workload and the server resources available on it. A container is scheduled on to Nodes that can meet the resource and load requirement of that container.
- The Controller-Manager is a daemon that keeps an overview of the activity within the Cluster. It is responsible for maintaining the state of the Cluster and ensuring that it is in the preferred state. It is made up of separate controllers which are: Node Controller, Job Controller, Endpoints Controller and Service Account & Token Controller. Each controller is a process with a specific role. For example, the Node Controller is in charge of keeping all Nodes up and running meaning that if a Node goes down, the Controller will detect and respond to it (Kubernetes (2021c)).

- The Etcd is a key value storage that holds the current state of the Cluster in the form of configuration data of all the Nodes and Containers. It is used as a backing store for Kubernetes Clusters.

2.1.1.2 Worker Node

The Worker Nodes are also called the Data Plane (Panagiotis (2020)). Multiple Pods that contain containerised applications run inside Worker Nodes. There can be multiple Worker Nodes inside a Cluster and are all controlled by the Master Node. Like the Master Node, the Worker Node also has components that allow it to function properly.

• Kubelet is an agent that runs on all the Nodes in a Cluster, including the Master Node. It ensures that the containers are running within a Pod and that they are healthy. Since it is not recommended to run Pods inside the Master Node, the Kubelet agent mainly works within the Worker Nodes. It acts as a communication link between the Master and Worker Nodes. It watches the API Server in the Master Node for any commands associated with its Pods and executes them accordingly.

• Kube-proxy is a network proxy that runs on all the Nodes in a Cluster. It maintains the network configurations of all the Pods within the Node. It ensures that all its Pods and the Node can communicate with the other Pods and Nodes within the Cluster by using tools such as iptables, serves proxy and Pass-thru traffic. It also implements the Services Concept within Kubernetes that involves exposing an application running within a Pod as a Network Service (Kubernetes (2021e)). This involves assigning the Pods with a tag that will not change every time the Pod has to restart and change its Internet Protocol (IP) address.

• Container Runtime is the software that is in charge of running the containers inside the Node. Several Container Runtimes are supported by Kubernetes such as Docker, Containers, CRI-O and any other implementation of Kubernetes' Container Runtime Interface.

• A Pod is the smallest deployable unit the user can create and manage in Kubernetes. It is a layer of abstraction around a single container or multiple containers that need to work together (Kubernetes (2021d)). Containers within a Pod are run with the same configurations and share storage, network namespace, IP addresses and port space. The containers within a Pod communicate with each other using localhost. Containers will have to use the shared network ports to communicate with any entity that is outside of their Pod. Pods are managed by the Nodes they are assigned to since their creation and remain under that Node until they are deleted. If the Node that a Pod is assigned to dies, then the Pod is scheduled for termination. Pods are also ephemeral components meaning that they frequently go down and need to be recreated (Panagiotis (2020)). In this scenario, an identical Pod is created as a replacement with the same name but a different unique ID and IP address.

2.1.2 Networking in Kubernetes

A Kubernetes Cluster is made up of many components that work together to manage multiple containers. These components need to communicate with each other to carry out the necessary tasks of these containers. Containers that require communication with external entities also need a method to establish their connections. This section will outline some aspects related to networking in Kubernetes.

2.1.2.1 Container to Container

The most simple connection within a Kubernetes Cluster is between two containers within the same Pod. They communicate via localhost and port numbers. As mentioned in the previous section, containers within the Pod share many resources including a network namespace and a virtual Ethernet connection, eth0. From the perspective of the containers, the Pod is a single machine with its own network interface meaning it has its own IP address and a range of ports (Palmer (n.d.)). The Pod can allocate these ports to different containers, allowing them to associate different containers with their port numbers. For example, in figure 2.2, Container A runs on port 8080 and Container B runs on port 9000. Container A can communicate with Container B via localhost:9000 and vice versa via localhost:8080.

Figure 2.2: Two containers in the same Pod communicate with each other via localhost using their port numbers. Both containers share the same network namespace. Container A (left) is assigned the port number 9000 and Container B (right) is assigned the port number 8080. Container A sends requests and responses to Container B using localhost:8080. Container B sends requests and responses to Container A using localhost:9000.

2.1.2.2 Pod to Pod

When containerised applications within Pods need to communicate between each other, the connection path created between them varies depending on whether they are within the same Node or a different Node. This section describes how Pods communicate with each other within the same Node and different Nodes.

2.1.2.2.1 Within Same Node

As mentioned previously, each Pod has its own IP address, network namespace and virtual Ethernet connection called eth0. This eth0 is connected to a virtual Ethernet device in the Node called vethX. Since these connections are made for every Pod in the Node, the X in vethX is replaced by a number that represents the Pod that it is connected to, for example, veth1 and veth2. The Node's Network Bridge, called cbr0, uses these connections in order to deliver packets to and from the Pods. A Network Bridge is a bridge that connects two namespaces together (Sookocheff (2018)), in this case, it connects all the Pods in a Node together.

Figure 2.3: Two Pods in the same Node, named Pod 1 and Pod 2, communicate with each other using a Network Bridge called cbr0. Pod 1 sends a request through its eth0 (1) to its corresponding virtual Ethernet device in the Node called veth1 (2). cbr0 checks and matches the destination address to the address of Pod 2 (3). The request is sent to Pod 2 via veth2(4) and arrives at its destination (5).

Figure 2.3 shows how two Pods communicate with each other within the same Node. Pod 1 wants to send a packet to Pod 2. All Pods within a Node know each other's IP address. Pod 1 sends the packet to its eth0 (1) which is connected to veth1 (2) in the Node's namespace. Inside the Node's namespace, the cbr0 receives a request from Pod 1 with a destination IP address, it checks all the Pods connected to it for it's destination address and once found, the packet is sent to the Pod (3). In this case, the packet it sent to veth2 (4) as it is the connection associated with Pod 2. veth2 is connected to eth0 in Pod 2's namespace and this allows for the packet to reach its destination, Pod 2 (5).

2.1.2.2.2 Between Different Nodes

Pods can also communicate with other Pods in different Nodes. As outlined in figure 2.4, when Pod 1 in Node 1 wants to communicate with Pod 1 from Node 2, the same steps explained in the previous section, regarding Pod to Pod communication in the same Node, occurs. The only difference is that, when the cbr0 of Node 1 cannot find the Pod with the destination IP address, it sends the request up to the Cluster level (1). At this level, there is a routing table containing the IP address ranges for each Node in the Cluster. For example, all the Pods in Node 1 would have an IP address similar to 172.17.1.1 or 172.17.1.3, and all the Pods in Node 2 would have IP addresses similar to 172.17.2.1 or 172.17.2.3. The routing table is aware of this pattern and identifies any IP address with the format 172.17.2.x as a Pod within Node 2 and 172.17.1.x as a Pod within Node 1 (2). Once the request has been sent to the correct Node (3), the network bridge of that Node will identify its Pod's address (4) and send the request to the destination Pod, Pod 1.

Figure 2.4: Two Pods in different Nodes communicate with each other using a routing table. Node 1's cbr0 cannot find the destination address of the request sent and sends it to the routing table in the Cluster level (1). The routing table checks and matches the destination address to an address belonging to Node 2 (2) and sends it (3). Node 2's cbr0 recognises the destination address and delivers the request to its destination (4).

2.1.2.3 Services

As previously stated, Pods are ephemeral components meaning that they frequently go down and need to be recreated. When recreated, new IP addresses are assigned to these new Pods making it difficult for other Pods to keep track of the updated IP address associated with each Pod (Panagiotis (2020)).

The concept of Services provides a solution to this problem. It involves exposing an application running within a Pod as a Network Service by assigning the Pods with a tag that will not change every time the Pod has to restart and change its IP address. Services are implemented by the Kube-proxy component within the Worker Nodes containing the Pods.

2.1.3 Security Considerations and Threats in Kubernetes

Many security and privacy considerations have been made by Kubernetes to ensure the security of their user's Clusters. However, as mentioned in section 1.1, many successful attacks have taken place in the previous years as a result of utilising the attack surface of a Kubernetes Cluster. This section will outline the attack surface of Kubernetes and a closer look at some of these threats and the way Kubernetes prevents them from happening.

2.1.3.1 Kubernetes Threat Matrix

Azure Security Centre created a map that covers the attack surface of Kubernetes (Microsoft (2020)). This attack matrix is based on the Mitre ATT&CK framework (ATT&CK (n.d.)) which is a knowledge base of known cyber attack techniques categorised into nine tactics. The Kubernetes attack matrix, shown in figure 2.5, outlines the techniques attackers use to compromise the Cluster.

Figure 2.5: The attack matrix of Kubernetes lists the different techniques used by attackers to achieve the nine tactics outlined in the Mitre ATT&CK framework (ATT&CK (n.d.)) which are Initial Access, Execution, Persistence, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement and Impact.

• Initial Access: The Initial Access tactic is the act of the attacker gaining access to the Cluster. This can be attained through simple methods such as using the credentials of an authorised user or entering through an exposed endpoint. More complex methods can also be executed such as compromising container images in the registry and having a user run them in the Cluster.

- Execution: Once the attacker has gained access to the Cluster, they need to run their malicious code within it. This is the Execution tactic: how the attacker executes their code. Techniques to achieve this include creating a new container or running an exec command, such as kubectl exec, within an existing container if the attacker has permissions for these actions.
- Persistence: The Persistence tactic involves techniques to allow the attacker to have a backup path into the Cluster in case they lose their initial access. One of the techniques involves the use of backdoor containers: containers that run malicious code. Attackers can create backdoor containers and ensure that a number of these containers are deployed and running within the Cluster so that they can use them to gain access again.
- Privilege Escalation: To access more resources and classified data, attackers need to attain more privilege within the Cluster. The Privilege Escalation tactic involves techniques that can be used to gain higher privileges within the Cluster. One way is gaining access to a container with special privileges that allows direct access to the host's resources. Kubernetes implements Role-Based Access Control within Clusters. A Cluster-admin is a role with high privilege. Binding to the cluster-admin or any other higher privileged roles can allow an attacker to utilise those privileges.
- Defense Evasion: As seen in some of the attacks that occurred in the real world, attackers follow methods to stay hidden once inside the Cluster. The Defense Evasion tactic involves using techniques such as clearing logs and giving the attacker's backdoor containers names that follow the naming convention used by the Cluster.
- Credential Access: Some attackers break into a system to steal valuable data such as credentials. Credential access tactics are used to get these valuable credentials. One technique is to retrieve Kubernetes Secrets that are stored within the Cluster. These secrets contain sensitive data and credentials.
- Discovery: Discovery tactics are used as a way for the attacker to examine the Cluster before moving around. Attackers can use the API Server, the Kubelet API, the Dashboard or the general

internal network, that is used by Pods to communicate with each other, to explore the Cluster.

- Lateral Movement: Once the attacker has examined the environment, they use Lateral Movement tactics to move around the Cluster. Some of the techniques used

• Impact: Many things can be achieved by the attacker once the Cluster has been compromised. The Impact tactic consists of techniques that attackers use to change the Cluster's normal state in a negative way such as deleting data or changing configurations. In many of the examples mentioned, the attackers used the Cluster to mine cryptocurrency. Attackers can also perform a denial of service attack that prevents the availability of the service and denies its users access.

2.1.3.2 Security Measures Taken to Prevent Certain Threats

This section will discuss some of the many security methods that Kubernetes implemented in order to prevent attackers from gaining access to Clusters.

Kubernetes Clusters are very easy for attackers to detect since they all listen to a range of distinctive ports that are well defined (McCune (n.d.)). Etcd, a component inside a Kubernetes Cluster, is used as a key value storage that holds the state of the Cluster in the form of configuration data of all the Nodes and containers. If an attacker gains access to this component, they can easily explore the entire Cluster. Since the setup of Kubernetes does not enable encryption by default, the Secrets stored inside the Etcd are in the form of plaintext. Some users may expose the Etcd to the Worker Nodes without Transport Layer Security (TLS), authentication and authorization (Panagiotis (2020)). As Etcd listens to port 2379, it is easy for attackers to find it since this port is indexed by Shodan: a search engine for devices connected to the Internet (McCune (n.d.)).

Security measures have to be taken to secure the Etcd such as making it mandatory to enable the option requiring all who require access to it need to have valid certificates. A firewall is also set up around it to make it more difficult for attackers to gain access (Panagiotis (2020)).

All Kubernetes Clusters contain a component called the API Server. API Server is a container that is the main entry point to the Cluster. It exposes the Kubernetes API, a RESTful web server, that is the main point of interaction between the outside and the Cluster. Communication between all other components also occurs through the API Server. The API has a secure endpoint that can be connected at port 6443 and an insecure endpoint at port 8080. If an attacker gains access to the insecure endpoint, then they can avoid all authentication and authorisation protocols that are in place and

make API requests to the port. Although the default settings of a Cluster disables this insecure endpoint, it may be still possible for an attacker to change this configuration and gain access to internal information within the Cluster (Panagiotis (2020)).

As this is the only endpoint that is available to the public, there are many security measures that are put in place in order for a user to gain access to the Cluster via the API's port. Transport Layer Security (TLS) is implemented for the API's security and all users requesting access will have to be authenticated and authorised before gaining access. The Hypertext Transfer Protocol (HTTP) request is examined and the system verifies that the Cluster knows the user through the use of either passwords, certificates or tokens. If the user is valid, the request that they are trying to make is examined. Since Kubernetes uses

Role-Based Access Control (RBAC), the system makes sure that the existing permissions granted to the user is enough to complete their request. Once the request has been authorised and authenticated, it has to be checked by the admission controller. This controller ensures that any request coming from the outside complies with a set of rules. For example, each Kubelet, a component that sits in every Node and relays the API requests to its destination, can only make requests to Pods within its own Node and not any other Pod (Panagiotis (2020)).

These security measures, however, do not always succeed as it is still possible to accidentally expose this port to attackers like the incident with Tesla. The attackers gained access to Tesla's Cluster through its exposed dashboard as it had no authentication methods enabled (Panagiotis (2020)). Since the Dashboard utilises the API, the attackers gained access to the Cluster.

Other than gaining access to the Cluster through exposed endpoints, attackers can get in through the help of containers. Kubernetes Pods run containers with different applications. By default, a token that grants access to the Kubernetes API is assigned to a container. If RBAC is not configured, then the token may grant the container with admin-level privileges (McCune (n.d.)). If one of the containers became compromised, then the attacker would have access to the Pod that hosts it and all the information within it. In worse cases, if this compromised container has a privileged flag, then the attacker can use these privileges to gain access to the rest of the Cluster and all the data within it (McCune (n.d.)). To avoid this, users are encouraged to configure RBAC and avoid running containers as privileged.

As Kubernetes has security and privacy considerations put in to prevent attackers from entering its Clusters, it needs to improve its ways of detecting an attacker that is already inside.

2.2 Machine Learning

This dissertation examines how machine learning can be utilised for intrusion detection in a Kubernetes Cluster. Machine Learning (ML) is a subset of Artificial Intelligence (V K (2019)). The aim of machine learning is to allow machines to be able to make decisions without explicit instructions coded into them (Pant (2019)). This involves creating algorithms that can adjust themselves to generate accurate results. For this research, algorithms can be created to identify malicious activity inside a Cluster. There are many ways to implement an algorithm that can learn.

2.2.1 Supervised Learning

In Supervised Learning, an ML algorithm is given training data in the form of a labelled dataset. A labelled dataset contains inputs and their expected outputs (for Geeks (2019)). Using this training data, the algorithm has to create a function that is able to map the inputs to the outputs. Once this function has been created, it can be tested against new data to review the output.

There are two types of supervised learning algorithms: Classification and Regression.

• Classification learning involves the output of the learning algorithm to be discrete or categorical. Classification can either be binary such as true or false, or multi class such as small, medium or large (for Geeks (2019)). An example of a problem that can be solved using classification learning is determining whether an email is spam or not. The characteristics of the email is the input and the output is binary classification of either spam or not spam. Another example is determining handwritten letters of the English

alphabet. The handwritten letters are the input and the output is multi class since it can be classified as one of the 26 letters.

• Regression learning involves the output of the learning algorithm to be continuous or real values. Examples include values of money, probability and metric measurements. The algorithm attempts to get an answer as close as possible to the expected value. The closer the output is to the expected value, the lower the error value and therefore, better accuracy.

2.2.2 Unsupervised Learning

In Unsupervised Learning, there is no form of guidance on what the right answer is. The algorithm has to discover the patterns of a dataset on its own and categorise the differences and similarities in order to determine the final output.

There are two types of supervised learning algorithms: Clustering and Association.

• Clustering involves the learning algorithm sorting the dataset into groups based on their similarities. An example of a problem that can be solved using this algorithm is grouping customers by their purchasing behaviours and patterns (Pant (2019)).

• Association involves the learning algorithm discovering patterns and linking those patterns with outcomes such as days with temperatures over 30 degrees can be linked to the days with the highest ice cream sales and therefore high ice cream sales can be associated with high temperatures.

2.2.3 Reinforcement Learning

In Reinforcement Learning, the algorithm learns by receiving rewards and penalties. The algorithm receives rewards for performing correctly and penalties for performing incorrectly. This process allows the algorithm to learn on its own and make every decision based on its past experience, similar to a child or young animal learning about its environment.

2.2.4 Deep Learning

Deep Learning is a subset of machine learning. It contains more complex algorithms than machine learning. These algorithms begin as simple implementations but accumulate in order to form multiple layers therefore adding complexity and earning the name of “Deep” Learning (V K (2019)).

2.2.4.1 Neural Networks

A Neural Network is the basic method used for deep learning solutions. It is designed to replicate the function of the human brain with multiple neurons connected to form a network of layers. It contains an

input layer, output layer and multiple hidden layers used to process all the data. These layers contain neurons with weights that give priority to certain characteristics of the input data. Neurons contain activation functions that also contribute to determining the output (Sharma (2017)). These weights and activation functions work together to decide how the neuron processes the data before passing it onto the next layer and generating a suitable output.

Neural networks can be designed differently with the basic functionality. Two types in particular are the Multilayer Perceptron and Convolutional Neural Networks.

• Multilayer Perceptron: A Multilayer Perceptron (MLP) is made up of the most simple type of neural network, a perceptron. A perceptron is a neural network with only an input and output layer. It does not contain any hidden layers, making it simple. An MLP is similar to a regular perceptron but it contains a hidden layer. It utilises backpropagation, in order to become more accurate with every iteration. Backpropagation is a supervised learning algorithm that is used to train feedforward neural networks. All the nodes within the hidden layer and the output layer contain non linear activation functions that, along with the multiple layers, help to sort and categorise data that is not linearly separable (Uniqtech (2018)).

• Convolutional Neural Networks: A Convolutional Neural Network (CNN) is more complex than an MLP. Unlike an MLP which requires weights to determine the importance of data, a CNN can prioritise the data on its own (V K (2019)). This feature allows for a reduction of preprocessing time for the data making it easier to train. It contains a convolution layer that performs a convolution operation along with an activation function. It also contains a Kernel, which is a small grid of parameters used to extract features from the data. It is used to process images due to its composition of data in the form of a grid pattern (Yamashita et al. (2018)).

2.3 Related Projects

This section will give an overview of the existing research and implementations that are relevant or closely related to intrusion detection in Kubernetes and the solution proposed in this paper.

2.3.1 Monitoring Kubernetes Clusters with Dedicated Sidecar Network Sniffing Containers

A research recently completed by Karode (2020) involves the use of sidecar network sniffing containers that utilise deep packet inspection in order to inspect the network traffic within a Kubernetes Cluster. Both the completed research and this current dissertation research share the same aim: Intrusion detection in Kubernetes. The paper introduces the idea of monitoring the internal traffic by inspecting each packet that is transmitted within each Pod. The solution also consists of a dashboard where users can monitor the network and evaluate the flagged malicious activity. The design is based on the architecture of Prometheus which is an open-source network tool used to monitor the network traffic.

A downside to this solution is that the sidecar container needs to be deployed within every Pod in the Cluster in order to inspect all the packets being transmitted within the Cluster. This allows each instance inside all the Pods to consume a large and uncommon amount of resources that belong to the containers in the Pod. To avoid this, a similar solution can be implemented that only has to be deployed once inside every Node within the Cluster instead of every Pod. Another limitation to this solution is that the user configures the rules that are used during the deep packet inspection of a packet. The user’s knowledge of a malicious pattern may not be enough to create a specific rule that captures it.

2.3.2 Security by Simple Network Traffic Monitoring

Tsunoda & Keeni (2012) discuss the method of monitoring network traffic in order to detect a security breach. They introduce a technique that can be used to analyse network traffic called Category Transform. This technique allows the dissection of many packets in detail and studies the characteristics of each of the packet's fields. The study also mentions how monitoring the Address Resolution Protocols (ARP) and Neighbour Discovery Protocols (NDP) can allow the monitoring of all the devices connected to the network.

The paper also discusses how traffic volume monitoring can help to detect intruders. Large variations in the traffic flow can be a sign of suspicious activity. It analyses the traffic volume by converting it using the category transform technique and discusses how this can help detect suspicious activity such as port scans and attacks such as Distributed Denial of Service (DDoS) attacks.

2.3.3 Detecting Network Intrusions via Statistical Analysis of Network Packet Characteristics

Bykova et al. (2001) outlines how to detect network intrusions by analysing the characteristics of network packets. The study aims to gather and examine Transmission Control Protocol (TCP) packets on a network that violate the existing standards and understand how they are capable of causing harm. This paper points out specific traits that suspicious packets will have within different attributes of the packet including the packet size, IP header, the IP address and the source and destination port numbers. Many of these characteristics are believed to be caused by poor TCP implementations but some of the packets analysed were not a part of the normal network traffic. Checking for these characteristics in network packets within the Cluster can help to detect malicious activity.

2.3.4 Detection of Malicious Encrypted Web Traffic Using Machine Learning

This research aims to focus on detecting malicious HyperText Transfer Protocol Secure (HTTPS) traffic without decrypting the packet. Shah (2018) proposes an ML solution that analyses the characteristics of HTTPS certificates. The ML solution utilises the Extreme Gradient Boosting (XGBoost) Algorithm to determine if the characteristics of the HTTPS certificate indicate malicious activity. The characteristics of the certificates are extracted from log files that are generated using a network security monitoring tool called Zeek (Zeek (n.d.)). The extracted characteristics become the input for the ML algorithm and are classified based on whether they are suspicious or not. This idea proves that it is possible to create an ML solution model that takes in packet characteristics as its input to determine if it is a normal or malicious packet.

2.3.5 Summary of Related Projects

It is evident that, from examining the works of Karode (2020) and Shah (2018), the idea of intrusion detection using network monitoring has been proven to be effective. Tsunoda & Keeni (2012) and Bykova et al. (2001) have also proven that certain characteristics exist that can lead to the identification of malicious traffic. While Karode's solution requires manual configuration of rules that the system will use to detect the malicious activity, Shah utilises machine learning to identify the patterns in the traffic. Karode utilises deep packet inspection that involves examining the different fields of an IP packet whereas Shah examines the HTTPS certificates of the normal and malicious packets.

These papers prove that it is possible to implement an intrusion detection system that utilises traffic monitoring and machine learning to identify a malicious packet using the characteristics defined with training data.

Table 2.1 highlights the aims of the different papers, their findings and what influence they have on the design of the solution being proposed in this dissertation.

Title of Paper Aim Findings/Solution Inspirations for Current Research

Monitoring Kubernetes Clusters with Dedicated Sidecar Network Sniffing Containers	Inspect network traffic within Kubernetes	Container can be deployed in every perform packet inspection on all the Custer traffic	Create a solution that does not need to be deployed on every Pod.

		Pod to deep	
Security by Simple Network Traffic Monitoring	Show how network security management can be obtained using basic traffic monitoring	A new traffic analysis technique called Category Transform that involves transforming traffic volume to traffic category.	Use traffic patterns to determine irregular activity within the network.

		This identifies usage within the network. technique illegal patterns	
Detecting Network Intrusions via Statistical Analysis of Network Packet Characteristics	Obtain information	A list of traits found within packet headers that include invalid IP address, port numbers and TCP flags.	Use packet characteristics to identify irregular packets within the network.

	on attacks examining characteristics of malicious Packet headers malicious by the		
Detection of Malicious Encrypted Web Traffic Using Machine Learning	Detect malicious HTTPS traffic without decrypting the packet.	Extract traffic data from log files and use XGBoost Algorithm to classify the packets as malicious or not.	Use machine learning to determine whether the packet is malicious or not.

Table 2.1: A summary of the four papers related to this research and the inspirations they have provided this solution. Both Karode (2020) and Shah (2018) propose projects that detect malicious activity and both Tsunoda & Keeni (2012) and Bykova et al. (2001) propose characteristics that help identify malicious packets. The solution presented in this dissertation aims to achieve similar outcomes based on these existing projects while attempting to overcome their limitations.

Chapter 3

Design

This section explains the design of the proposed solution in this dissertation and outlines the aspects that influenced it such as how the traffic is captured, how the malicious packets are identified and how a machine learning algorithm can be created to classify the packets.

3.1 Overview of the Approach

This research aims to design a solution that is capable of detecting whether an intruder exists within a Kubernetes Cluster with the help of machine learning. The proposed solution consists of a method to capture all the traffic flowing within a Node and pass it into a neural network that is deployed and running within a Pod in the same Node. The neural network determines whether there are suspicious packets with the use of characteristics that can be used to identify malicious packets.

3.2 Part 1: Capturing Traffic

A Kubernetes Cluster has an internal network that all the components within it use to communicate with each other. If an intruder enters the Cluster, they would need to move around, find valuable data and then steal it by transferring it out of the Cluster. These actions require the use of the internal network. Monitoring the internal network traffic can help in identifying suspicious activities and declare the Cluster to be compromised.

A Pod is the smallest deployable unit in a Kubernetes Cluster. Pods are deployed within a Worker Node in the Cluster and communicate with each other through the Node's network namespace. This means that monitoring a Node's network namespace can reveal all the traffic flowing in and out of the Pods within that Node. An attacker with access to a container inside a Pod will need to communicate with other

components in order to execute their plan and thus, prompting them to use the network namespace of the Pod's Node.

Inside the Node's network namespace, there is a network bridge, cbr0, that directs all requests going to and from the Pods. As illustrated in figure 3.1, cbr0 has access to all the virtual Ethernet connections with the Pods. Listening to these connection interfaces will result in a way to monitor all traffic flowing through the Node. Once the traffic is captured, it can be sent to a Pod that runs the intrusion detection algorithm to be examined.

Figure 3.1: The bridge, cbr0, has access to all the connection interfaces, which are represented as black dotted lines, in the Node's namespace. Monitoring these connections will allow the administrator to capture the traffic.

3.3 Part 2: Identifying Malicious Characteristics of Packets

The traffic flow within the Node is expected to contain a mix of network protocols such as Address Resolution Protocol, Domain Name System protocol, Hypertext Transfer Protocol, Transmission Control Protocol and User Datagram Protocol.

3.3.1 Network Protocols within the Cluster

- Address Resolution Protocol (ARP) is a data link layer network protocol (Engine (n.d.)). It is used to find the physical machine address, also referred to as the media access control (MAC) address, of an IP address. ARP is used by cbr0 when it checks all the Pods for the destination IP address of a request.
- Domain Name System protocol (DNS) is an application layer network protocol. It is used to map hostnames to their respective IP addresses such as converting domain names of websites to IP addresses (Engine (n.d.)). The protocol uses a distributed database containing name servers as a lookup. In a Kubernetes Cluster, Pods can host a coreDNS server (Kubernetes (2021f)). Pods that want to send requests to websites send a DNS query to the coreDNS server to receive the IP address of the desired destination.
- Hypertext Transfer Protocol (HTTP) is an application layer protocol (Engine (n.d.)). It is a stateless protocol that requires a client-server connection where the client sends requests to the server and the server sends back responses with the required data. Pods that require a connection with an external web server that support this protocol will send HTTP requests to the server and receive responses.
- Transmission Control Protocol (TCP) is a transport layer protocol. It is a connection oriented protocol meaning that the two applications have to establish a connection first before transferring data between them. TCP also utilizes acknowledgements and negative acknowledgements to ensure the data between the two applications is not lost. HTTP protocols use TCP as their transmission protocol (Docs (2019)). Large DNS queries over 512 bytes also use TCP as their transmission protocol (NS1 (2018)).
- User Datagram Protocol (UDP) is a transport layer protocol. It is used for the transmission of data that is sensitive to time (Datta (2020)). Unlike TCP, UDP is not connection-oriented. It sends data without an initial connection being made which results in faster transmission but can result in loss of

data due to the lack of a packet recovery mechanism. DNS queries use UDP as their default transmission protocol due to its low latency.

3.3.2 Scope

For the purpose of this research, the solution will focus on the traffic sent out from each Pod which uses application protocols such as HTTP and DNS. These protocols utilise the TCP or UDP transmission protocols and, therefore, produce IP packets that can be examined using the guidelines proposed by Bykova et al. (2001).

3.3.3 Characteristics of Malicious Packets

Bykova et al. (2001) outlines specific characteristics of a TCP packet that can indicate suspicious activity. Certain patterns observed within these three fields in an IP packet can help in designing the proposed solution.

• Packet Size: According to Bykova et al. (2001), the total length of a packet should be larger than the length of its header. TCP packets with extremely small lengths are also suspicious due to a method used to bypass firewalls that involves the splitting of packet headers, therefore making the packet short. Large packets should also be treated suspiciously as some attackers may not attempt to lay low and instead, steal large amounts of data. This would result in a packet with an abnormally large payload that can be detected.

• IP Address: The IP address of a packet is not protected from attackers. Attackers can spoof IP addresses. This involves replacing the real source address of the packet with a fake address that belongs to the attacker. This can result in the response to that packet being sent to the attacker's fake IP address. Keeping a track of regular IP addresses that are familiar and safe will help to detect new IP addresses which could be from an attacker. According to Bykova et al. (2001), there is an attack named the Land Attack that involves the source and destination IP address of a packet being the same.

• Port Number: Bykova et al. (2001) states that the source and destination port numbers of the packet cannot be zero. Also, it is suspicious for a packet to have the same destination and source port. Another suspicious characteristic regarding port numbers is that port 22 is assigned to Secure Shell (SSH): a cryptographic network protocol that is utilised by applications such as command-line (Loshin & Cobb (2019)). This means that if a packet has port 22 as its destination port, it can indicate that an attacker is trying to SSH into the Pod with the use of a command line.

3.4 Part 3: Machine Learning

Applications of machine learning would be useful in helping detect malicious activity inside a Kubernetes Cluster because of its ability to recognise unusual patterns and identify them as malicious using its past experiences or training. Previous machine learning solutions have proved to be successful for network

traffic inspection such as the XGBoost algorithm used by Shah (2018). This research explores this further and aims to design a solution that uses another machine learning algorithm.

Using the selected fields of the IP packet such as the length of the payload, and the source and destination IP address and port numbers, a machine learning algorithm can identify malicious patterns of the IP packet.

3.4.1 Selecting a Learning Method

The ML model needs to follow a learning method to learn using the datasets provided before being able to examine and detect malicious traffic. There are three methods that can be used to create a machine learning algorithm: Supervised, Unsupervised and Reinforcement Learning. A method that suits the purpose of this project is selected.

Reinforcement learning involves the algorithm learning through its environment however, this problem set does not contain an environment to explore. Since unsupervised learning finds its own patterns within the dataset, there is no guarantee that it will pick up on the necessary patterns agreed on in section 3.3. In supervised learning, the dataset is based on set patterns defined by the programmer that determine if a packet is malicious or not. Supervised learning can take into account the characteristics of a malicious packet that will be defined using the training dataset that is passed into the algorithm.

Within supervised learning, there are two types of algorithms that can be used: Regression or Classification. Regression is used where a continuous value output is required such as weight, speed or distance. However, Classification is used where a categorical output is required such as true or false, 0 or 1, or hot or cold. It is used to categorise the data into groups which, in this solution, is either Normal packet or Malicious packet. Classification is selected as the type of algorithm suited for this solution.

3.4.2 Selecting a Neural Network

A neural network is a deep learning algorithm that can be trained to identify certain data that has different characteristics within a dataset.

Two types of neural networks researched in this paper are the Multilayer Perceptron (MLP) and the Convolutional Neural Network (CNN). Since CNNs are more suitable for handling grid-based datasets such as images and this input dataset consists of IP packet fields, the MLP is selected as the type of neural network that is used to identify malicious packets.

3.4.3 Overall Machine Learning Solution

The final design of the machine learning solution, shown in figure 3.2, consists of a Multilayer Perceptron Neural Network that takes in the selected fields of an IP packet as inputs and labels them as malicious or normal as outputs. The algorithm uses supervised learning meaning that it uses training data to understand the patterns within the dataset and try to shape a function that will output the expected outputs for its respective input training data. The algorithm is a classification algorithm meaning that the output can be categorised into either Malicious or Normal.

Figure 3.2: The architecture of the machine learning solution consists of a Classifier Neural Network. This model takes in the attributes of the packets captured from the Node's internal network and determines whether they are malicious or not.

3.5 Tools

Implementing this design requires the use of different tools and libraries. The main tools that can aid in the implementation of this project are described in this section. It also outlines why these tools are selected to be used in this project.

3.5.1 Minikube

In order to monitor traffic flowing within a Kubernetes Cluster, there needs to be a Cluster available for local development. Minikube is a tool that allows a user to run Kubernetes locally as a single-node Kubernetes Cluster inside a virtual machine (Kubernetes (2021a)). Using this tool, it is possible to create Pods and deploy docker containers running applications inside the Node. It is a great tool to help beginners locally explore a Kubernetes Cluster.

3.5.2 Lens

Controlling and monitoring a Kubernetes Cluster through the command line is complex, however, there are tools to help with that such as Lens. Lens is an open-source Integrated Development Environment (IDE) for Kubernetes Clusters (Lens (n.d.)). This tool can be used to monitor the traffic in the Node and capture it before getting it examined by the algorithm.

3.5.3 Machine Learning Libraries

Implementing a neural network from scratch is time-consuming and complex. Fortunately, there are many libraries available online that can be used to create a neural network. Scikit-learn is an open source machine learning library that can be used to implement various machine learning algorithms in Python (Scikit (n.d.b)). It provides basic machine learning algorithms and models that support supervised and unsupervised learning. An MLP is also available to be implemented and only requires the correct training data (Scikit (n.d.a)).

Using these tools, the design of the proposed solution can be implemented.

Chapter 4

Implementation

This chapter outlines how the proposed intrusion detection solution is implemented. It explains the steps taken in setting up a local Kubernetes Cluster and capturing the traffic within the Node. Furthermore, it explains how to create a dataset that can be used to train a Multi-layer Perceptron Neural Network to detect malicious packets using classification.

4.1 Traffic Monitoring

This section describes the steps taken in capturing data from a Kubernetes Cluster.

4.1.1 Set Up Kubernetes Cluster

As mentioned in section 3.5, Minikube is used to create a single-node Kubernetes cluster that can be used for local development. After installing the tool, the cluster can be initialised by typing the command minikube start into the terminal.

Currently, no Pods exist in the single node cluster and, therefore, will need to be created. In order to create a Pod, a Deployment will have to be created. A Deployment creates a ReplicaSet and a ReplicaSet can create Pods. A Deployment can be created by running the following line in the terminal:

```
kubectl create deployment pod-one --image=k8s.gcr.io/echoserver:1.4
```

Kubectl is the command line interface for Kubernetes and is installed along with Minikube. The pod-one argument in the command can be replaced with any other name desired for the Pod. Multiple Pods can be created in a similar way.

Lens is used to monitor this minikube cluster as it is an IDE for Kubernetes Clusters. 31

Lens also provides a feature to deploy applications from the Helm charts to Kubernetes clusters. Helm is a package manager for Kubernetes (Helm (2019)). Using this feature, a Wordpress (Wordpress (2015)) application is deployed into the cluster and now runs inside a Pod in its own name. Another Pod hosts a mariadb for the Wordpress application.

The Kubernetes cluster is now ready to be monitored as it has Pods that will communicate with each other, see figure 4.1.

Figure 4.1: The single-node cluster has four Pods: Two pods utilised by WordPress and two running a container with an image of an echo server.

4.1.2 Capturing Traffic

In order to monitor the network traffic within the cluster, a command-line packet analyser called tcpdump is utilised (Tcpdump (n.d.)). Lens provides a feature to SSH into the Node as the root user. This allows command-line arguments to be executed within the Node. Using this access, tcpdump can be installed in the Node using apt-get which is a command-line tool in Linux that handles packages (die.net (n.d.)).

As previous chapters have explained, a Kubernetes Node has a network namespace that is shared by all the Pods and their virtual Ethernet connections also called interfaces. Tcpdump can list all these interfaces using the following command: tcpdump -D. Figure 4.2 shows all the interfaces that are in this Node and the Pod they are connected to. The docker0 is the network bridge instead of the expected cbr0 (Docker (2017)). Since all packets sent to and from the Pods in the Node go through the bridge, listening to docker0 will allow access to all the traffic within the Node.

Figure 4.2: All interfaces in the Minikube Node along with the names of the Pods they are connected to.

The following command uses tcpdump to capture all the traffic flowing through the docker0 interface and then save it as a pcap file (ReviverSoft (n.d.)).

```
tcpdump -s 0 -i docker0 -w capture.pcap
```

The pcap file will contain a copy of the traffic captured including TCP, UDP and ARP packets. It can be opened and read using Wireshark, a network protocol analyser (Wireshark (2017)), as seen in figure 4.3.

Figure 4.3: A glimpse into a pcap file containing internal traffic of the Minikube Node. The traffic is captured using tcpdump and viewed using Wireshark.

4.2 Intrusion Detection Model

The intrusion detection model is implemented with the use of a multi-layer perceptron neural network available from Scikit which is a machine learning library. As this model uses supervised learning, it will need a training dataset that maps all the inputs to their expected outputs. This dataset is created using the

traffic captured and locally generated malicious packets that are designed using the characteristics discussed in section 3.3.

4.2.1 Creating the Dataset

A dataset consisting of normal and malicious packets needs to be generated in order to train and test the intrusion detection model. The normal packets are taken from the traffic captured and the malicious packets are generated due the limitation of accessing real malicious traffic. This section describes the steps taken in creating this dataset.

4.2.1.1 Dissect Normal Traffic Data

The packets captured in the pcap file are used as the normal data that trains the neural network to understand what is not a malicious packet. They need to be converted into training inputs for the neural network. To achieve this, a python module named dpkt (Song (2019)) is used to read and parse the file along with the packets in it. The pcap file is read into the model and the desired packet fields are extracted to create the input dataset. For each packet, its length, source IP address, destination IP address, source port number and destination port number are extracted and stored in respective List objects. Once stored in the Lists, the attributes of a packet can be accessed using the same index value for all the attribute Lists.

4.2.1.2 Create Malicious Data

The training input dataset also requires malicious packets so that the neural network can recognise what a malicious packet would look like. Unfortunately, generating real malicious packets was not possible during the time this study was being conducted so the malicious packets utilised for training are designed using certain characteristics that are outlined in section 3.3.

• Packet Size: As per the characteristic of malicious packets with unusually small or large sizes, the malicious packets created in this study are assigned a random number within a specified range as their size attribute. A range for the malicious packet sizes is created after observing the size of the normal packets captured. It is observed that no packet has a size attribute below 50 bytes or above 8000 bytes. Since the minimum size of a packet is 21 bytes (Hall (2019)), any size below that size is invalid and any size between 21 bytes and 50 bytes is considered to be not normal compared to the patterns observed in this cluster.

Since the maximum size of a packet is 65,535 bytes (Hall (2019)), any packet that has a size between 8000 bytes and the maximum value is considered to be not normal when compared to the non-malicious packets within this cluster.

• IP Address: As it is difficult to determine whether the owner of an IP address is an attacker or if the address has been spoofed, this model is designed to suspect any new IP address that has never communicated with the cluster. The malicious packets created contain different IP addresses to those that are observed in the normal dataset of packets.

• Port Number: As per the characteristics outlined, the malicious packets created can have the same source and destination port number or a port number of 0. The packet can also have a destination port number of 22 which indicates that it is an attempt to access the Pod via SSH.

All these malicious characteristics are assigned randomly to the generated dataset of malicious packets so that they have different combinations and will be similar to real malicious packets. Figure 4.4 shows some examples of the generated malicious packets.

4.2.1.3

Figure 4.4: Packet a) has the malicious characteristic of having a different source IP address from the ones observed in the normal packets. Packet b) has a length greater than 8000 bytes which no normal packet has exceeded. Packet c) has a different source address and a length shorter than 50 bytes which no normal packet has gone below. Packet d) has an invalid source port of 0.

Dataset Object

The attributes of the generated malicious packets are inserted into the attribute lists containing the normal packet attributes. These lists represent the final training input dataset. These lists are used to create a dataset object described in figure 4.5. This object contains an attribute for each list and it is used to pass the training input data to be preprocessed before being passed into the neural network.

Figure 4.5: The dataset object containing all the lists of the packet attributes. All normal packets captured from the Minikube Node and malicious packets generated using defined characteristics are included in this dataset object.

4.2.2 Pre Processing Dataset for Neural Network

The MLPClassifier is imported from the Scikit library to be used as the neural network in this intrusion detection model. Once the neural network is initialised, it needs to be trained with the training dataset. The packet attributes within the training dataset need to be preprocessed before it can be passed into the neural network. The function that fits the model to the training data takes in an array or matrix as the input for training and another array or matrix as the expected output.

4.2.2.1 Create Expected Output Values for Training

An array of 0’s and 1’s is where 0 means that the packet is normal and 1 means that the packet is malicious. Its index values correspond to the index values of the lists containing the packet’s attributes. This array is passed into the fitting function along with the processed input dataset.

4.2.2.2 Convert Non-Numerical Values

The neural network can only take in numerical data as its input. The IP address attribute is a non numerical value and, therefore, needs to be converted into a numerical value. A LabelEncoder from Scikit is used to assign unique numerical tags to string values as seen in the following code snippet 4.1.

```
# Create labels to replace string values

str_cols = df.columns[df.columns.str.contains('(?:IP)')] labels = {c:LabelEncoder() for c in
str_columns}

# Replace values in the dataset with their respective labels

for col, label in labels.items():
   df[col] = labels[col].fit_transform(df[col])
```

Listing 4.1: All the columns with string values to be converted are identified and the label encoder goes through each value and assigns a numerical value to it. For example, the IP address of 172.17.0.2 in figure 4.6 is assigned a tag of 2 in figure 4.7, 172.17.0.1 is assigned a tag of 1. These tags will replace their non numeric values for every entry that exists in the selected column.

Figure 4.6: Input dataset containing all normal and malicious packets before any preprocessing required for the neural network is performed on it.

Figure 4.7: Input dataset after the conversion of the IP addresses, which were initially represented as strings, to a numeric label.

4.2.2.3 Normalisation

When the values are all numeric, they have to be normalised as the neural network can only process values between 0 and 1. Normalisation involves scaling the values between 0 and 1. The code snippet 4.2 shows how all values in all the columns were normalised.

Listing 4.2: All values within the dataset are normalised before being passed into the neural network.

The inputs will now all be converted to numerical values between 0 and 1. Figure 4.8 shows the format the input data set will be passed into the neural network.

4.2.3 Training and Testing

The processed dataset is split into two sub datasets. One dataset will be used to train the neural network model and the other will be used to test it.

The input dataset and its respective expected outputs are passed into the function that fits the model to the training data.

```
clfr.fit(train_df, train_results)
```

```
target_column = ['1']

col_label = list(set(list(df.columns))-set(target_column)) df[col_label] = df[col_label]/df[col_label].max()
```

Figure 4.8: Input dataset after normalisation. The dataset has fulfilled all the requirements to be passed into the neural network as training and testing data

Once the model has been trained, it is tested using the test input data. The predict function takes the input data and returns the output determined by the trained model. The generated output can be compared to the expected output to determine the accuracy of the model.

ans = clfr.predict(test_df) 4.2.4 Output

The output of the neural network is initially an array of 1s and 0s. These values are mapped to the labels Malicious and Normal respectively. The final output of the intrusion detection model is a table displaying the packet attributes and whether that packet is Malicious or Normal as shown in figure 4.9.

4.3 Summary of Implementation

Using the knowledge regarding the Kubernetes Cluster architecture and machine learning models, an intrusion detection model was developed. The traffic captured by the root user in a Kubernetes Cluster is passed into the intrusion detection model which consists of a neural network that implements classification. This intrusion detection model processes the traffic and converts it into a dataset object that is further processed before being passed into the neural network. The model determines whether the packets inside the dataset are malicious or not. The solution uses datasets to train the ML model before evaluating any other datasets.

As displayed in figure 4.9, the model generates a table of the examined packets along with its details and its status of either being Malicious or Normal, therefore implementing intrusion detection in a small scale Kubernetes Cluster.

Figure 4.9: The output of the Intrusion Detection Model. Each packet is classified as either Malicious or Normal based on the characteristics and patterns the model has been trained to identify.

Chapter 5

Evaluation

This chapter evaluates the proposed solution model by conducting tests using multiple datasets and analyzing the results produced. In addition, the chapter explains whether the initial aims of this research were accomplished and the strengths and weaknesses of the proposed solution when compared to the related works and itself. It also outlines the challenges faced during this project.

5.1 Analysing the Output

The previous section mentions how the dataset for training and testing the model is created. Using these datasets, the model is trained and then tested. The results obtained are then analysed in order to evaluate the reliability of the solution.

5.1.1 Metrics

To evaluate the performance of the model, certain attribute values are recorded and used in calculations to obtain metrics. These values are recorded for three different datasets and are defined below in relation to the proposed model:

- True Positives (TP): The number of actual malicious packets that are predicted as Malicious by the model.
- True Negatives (TN): The number of actual normal packets that are predicted as Normal by the model.
- False Positives (FP): The number of actual normal packets that are predicted as Malicious by the model.

• False Negatives (FN): The number of actual malicious packets that are predicted as Normal by the model.

Metrics such as Accuracy, True Positive Rate and False Positive rate are used to evaluate the reliability of the model and are defined below (Nighania (2019)):

• Accuracy (ACC) is measured by calculating the percentage of the results that are correct such as TN and TP out of the total number of results which is TN + TP + FN + FP. This metric is used to evaluate the number of times the model is correct with its output. It is represented using the following formula:

ACC = TP +TN (5.1) TP +FP +TN +FN

• True Positive Rate (TPR) is measured by calculating the percentage of the correct number of positive results predicted (TP) out of the total number of results that are actually positive which is TP + FN. This metric is used to evaluate the number of times the model's positive prediction is correct out of all the actual positive results. A TPR of 80% means that the model can predict 80% of all the positive results correctly. In this study, a TPR of 80% would indicate that 80% of the malicious packets present in the dataset is classified as malicious by the model. It is represented using the following formula:

TPR = TP (5.2) TP +FN

• False Positive Rate (FPR) is measured by calculating the percentage of the number of negative results that were predicted positive (FP) out of the total number of results that are actually negative which is FP + TN. This metric is used to evaluate the number of times the model's positive prediction is incorrect. A FPR of 10% means that 10% of the positive predictions made by the model is incorrect. In this study, an FPR of 10% would indicate that 10% of the packets classified as malicious is incorrect. It is represented using the following formula:

FPR = FP (5.3) FP +TN

5.1.2 Results

The model will be assessed using three datasets of different sizes. These datasets contain training and testing data and the values of TP, FP, TN, FN are obtained from these tests. Dataset 1 is the smallest set, as shown in figure 5.1, it is tested with a testing dataset of size 274 that is composed of 56% normal and 44% malicious packets. Dataset 2, as described in figure 5.2, has a testing dataset of size 1494 that is composed of 73% normal and 27% malicious packets. Dataset 3, as described in figure 5.4, is the largest dataset evaluated with a testing dataset of size 3496 that is composed of 71% normal and 29% malicious packets. The values of TP, FP, TN, FN obtained from passing dataset 1, dataset 2 and dataset 3 into the model are shown in figure 2.1.

Normal Packets Malicious Packets Total

Training Data

615

480 1095

Testing Data Total

154 769 120 600 274 1369

Table 5.1: The number of malicious and normal packets in Dataset 1.

Normal Packets Malicious Packets Total

Training Data

4372
1600
5972

Testing Data Total

1094 5466 400 2000 1494 7466

Table 5.2: The number of malicious and normal packets in Dataset 2

Normal Packets Malicious Packets Total

Training Data

9982

4000 13982

Testing Data

2496
1000
3496

Total

12478 5000 17478

Table 5.3: The number of malicious and normal packets in Dataset 3

Using the values of TP, FP, TN, FN obtained from passing dataset 1, dataset 2 and dataset 3 into the model, the metrics ACC, TPR and FPR are calculated as per the formulas 5.1 , 5.2 and 5.3. The results are described in figure 2.1.

TP TN FP FN ACC(%) TPR(%) FPR(%)

Dataset 1 Dataset 2 Dataset 3

589 753 16 11 1972 5460 6 28 4876 12443 33 124

Average

98.02 98.16 2 99.5 98.6 0.1

99.79 97.52 0.26

99.1 98.09 0.78

5.2

Table 5.4: The metrics collected from the model processing three datasets. The ACC, TPR and FPR for this model are 99%, 98% and 0.78% respectively. This result may vary depending on the addition of real malicious packets instead of the ones generated for this purpose.

Analysis of Solution

This section recaps on the initial aim of this research and whether the proposed solution has achieved it. It also compares the proposed solution to the works that inspired it and highlights its strengths and limitations compared to them.

5.2.1 Results of Research

The aim of this project was to create a solution that conducts intrusion detection within a Kubernetes Cluster. As a popular infrastructure that deploys and manages containerised applications, Kubernetes Clusters became a target for many attackers. Companies such as Tesla and Docker Hub are victims of attacks that have raised awareness of the security concerns and challenges of Kubernetes Clusters. Attackers mainly utilised the Cluster to mine cryptocurrency which meant that they would utilise the internal communication network of the Cluster. This opened up the idea of monitoring the internal network traffic of the Cluster in order to detect suspicious activity. In order to understand the internal network of the Cluster, deep research into the architecture and specifically network architecture of a Kubernetes Cluster was conducted. Once a clear understanding of the Cluster's network architecture was reached, research began for a solution to detect intrusion.

The idea of analysing the network traffic in order to identify malicious activity within the cluster has been researched before and different versions of solutions exist, two of which were referred to while conducting this research in section 2.3. One of the solutions involved the use of a machine learning algorithm which led to the idea of using deep learning with neural networks to analyse the network traffic. To utilise machine learning, the model would need to be trained using sample malicious traffic. As it was challenging to

capture real malicious traffic, further research was conducted into the features that identify malicious traffic in order to create samples to train the solution model. Characteristics outlined in section 3.3 were used to generate training data to create the solution model. As of now, the model produces promising results however, it does not come without fault. The following section discusses how this solution compares with the existing solutions.

5.2.2 Comparison to Existing Solutions

In section 2.3, figure 2.1 outlines a summary of the different research that exists regarding this solution. The summary also contains a section that indicates the inspirations that came from those existing studies. This section compares the achievements and drawbacks of those solutions to the one presented in this dissertation and whether the inspirations defined in the summary table were implemented.

5.2.2.1 Monitoring Kubernetes Clusters with Sidecar Network Sniffing Containers

The dissertation by Karode (2020) outlines the idea of monitoring the internal traffic by inspecting each packet that is transmitted within each Pod. It proposes a real-time traffic monitoring solution that can be accessed using a dashboard. The malicious traffic is identified using the user-configured rules and alerts. However, the sidecar container needs to be deployed within every Pod in the Cluster in order to inspect all the packets being transmitted within the Cluster.

To overcome this disadvantage of this existing solution, the proposed solution is designed to only be deployed once in every Node. Using the administrator access, the traffic flowing between all the Pods within a Node can be captured and sent to the intrusion detection model that is deployed in only one of the Pods in that Node. The proposed solution also does not require the configurations of rules to define malicious traffic and instead utilises a machine learning model to identify them. The machine learning model is trained using normal and malicious traffic in order to be able to identify suspicious packets.

On the contrary, the proposed solution, however, does not provide real-time monitoring which is an integral part of traffic monitoring and is implemented by Karode.

5.2.2.2 Security by Simple Network Traffic Monitoring

Tsunoda & Keeni (2012) discusses the method of monitoring network traffic in order to detect a security breach. They mention how monitoring the ARP and NDP protocols can allow the effective monitoring of all the devices connected to the network. The research also outlines how monitoring traffic volume can help to detect intruders.

The proposed solution did not achieve the aim of analysing traffic patterns such as traffic volume to determine irregular activity within the network. The solution instead focused only on the individual packets and their attributes. ARP or NDP protocols are also not analysed in the proposed solution as it focuses on TCP and UDP packets.

5.2.2.3 Detecting Network Intrusions via Analysis of Packet Characteristics

Bykova et al. (2001) points out specific traits of the fields of a TCP packet that identify as malicious. Some of the fields discussed include the packet size, IP header, IP address and port numbers. Checking for these characteristics in network packets within the Cluster can help to detect malicious activity.

The proposed solution utilises this study to train and test its intrusion detection model. Characteristics that involve packet fields such as IP address, port number and packet length are used to generate malicious data to train this model. However, not all the characteristics outlined in the study were able to be used while creating the malicious data, such as characteristics that involve the TCP Flag and Time To Live (TTL) field.

5.2.2.4 Detection of Malicious Encrypted Web Traffic Using Machine Learning

Shah (2018) proposes a machine learning model that analyses the characteristics of HTTPS certificates using the XGBoost algorithm and determines if the characteristics indicate malicious activity.

The proposed solution is inspired by the use of machine learning in this study and uses a multilayer perceptron neural network instead of the XGBoost algorithm. However, the proposed solution examines the fields of TCP packets and does not go into a deeper examination of HTTPS certificates. The proposed solution does not conduct such deep checks on packets and does not take into account the encryption of packets creating a disadvantage.

5.3 Advantages and Disadvantages

This section outlines the advantages and disadvantages of the proposed intrusion detection solution.

5.3.1 Advantages

- Low Power Consumption: Unlike the sidecar container, which is a solution that requires to be deployed in every Pod, this solution involves the model only to be deployed once within the Node. In the case of the sidecar container being deployed inside every Pod, each sidecar communicates with the central dashboard creating more traffic within the network that is being monitored. This increase in the amount of traffic that needs to be monitored along with the multiple sidecars processing data concurrently requires a large amount of resources such as processing power. Compared to the sidecar container solution, the proposed ML model consumes less power and generates less traffic as only a single instance needs to be deployed within a Node and the traffic monitoring is carried out through a root access into the Node.
- Automatic Recognition of Malicious Characteristics: This solution utilises the method of supervised machine learning in order to adapt and detect unusual activity in the network. Unlike the sidecar container solution, there is no need to configure any particular rules to detect the packets within the

flow of traffic. Instead, the multilayer perceptron neural network analyses the packets and detects any malicious ones with the help of its previous knowledge. Any new pattern that is observed in the traffic flow will be automatically detected once it is included in the training data.

5.3.2 Disadvantages

- Not Realtime: The solution relies on captured pcap files which are captured by the admin/root user. This is a disadvantage since this allows the traffic to be analysed sometime after a delay. This increases the amount of time an attacker can stay hidden within the cluster. The question of whether making this solution run in real-time is possible or not is debatable.

- Not Analysing Traffic Patterns: The current solution is restricted to checking certain fields of the IP packet including the IP addresses, port numbers and the length of the packet. Traits such as the volume of traffic are not monitored in the proposed solution. Tsunoda & Keeni (2012) explain that the variations of the volume of traffic packets can indicate if there is an attacker within the cluster performing unauthorised actions.

• Limited to TCP and UDP Packets: The current proposed solution only inspects TCP and UDP packets. The study conducted by Tsunoda & Keeni (2012), outlines how other protocol types such as ARP and NDP can also aid in detecting intruders. ARP Spoofing is also a technique used by attackers and it cannot be detected by this model since it involves analysing ARP requests.

• Reliance on Previous Attack Data: As this model uses supervised machine learning, it requires sample data to learn the patterns produced by malicious attacks. The model will need to be updated with newer malicious packet patterns in order to keep up with the newer attack methods. As seen in section 1.1, attackers are finding newer ways to enter and remain inside the cluster for longer periods of time. In order for this model to remain reliable, it needs to be updated with these new methods. These constant updates will result in the redeployment of the model's Pod. A delay to this deployment can result in the model being inactive for a period of time creating a vulnerability.

5.4 Challenges

One of the main challenges faced during this research is the time restriction due to the current pandemic. Understanding the internal architecture of Kubernetes Clusters was difficult as it is a complex system with multiple components and methods. In addition, understanding the network architecture of the Minikube Node was also challenging as monitoring the Node's traffic was an aim of the study. Another major challenge faced during this research, is the attempt in simulating an attack in order to gather some training data for the model. The difficulty in capturing real malicious data was the inspiration behind training the model using mocked malicious data created using traits that have been identified by Bykova et al. (2001).

Chapter 6

Conclusions & Future Work

This chapter concludes this dissertation by outlining the initial research questions introduced in section 1.3, whether the questions have been answered and the future works that can be built on top of this research.

6.1 Outcomes

This section outlines the outcome of this research and whether it answers the research questions. In section 1.3, two research questions were addressed that this research aimed to answer. These questions are answered below:

- What is an effective method to monitor network traffic in a Kubernetes Cluster?

  In section 2.1, the internal network of Kubernetes is explained. It highlights the paths taken by packets from one application to another within the Cluster. Existing solutions like Karode (2020)'s sidecar container that monitors the network activity in each Pod consumes high processing power, however, the proposed solution monitors the network within the Node. Monitoring network activity within the Node consumes less power, unlike the sidecar container, and all internal and external Pod activity can be monitored with one deployment. Capturing network traffic from within a Node gives full insight into the network traffic with less power consumption.

- How can monitoring network traffic contribute to intrusion detection within a Kubernetes Cluster?

  As explained by Bykova et al., the packets within the network traffic possess many characteristics that can be identified with that of an attacker's packet. Once these characteristics have been observed, the packet can be flagged as suspicious or malicious. As pointed out in section 1.1, the attackers mainly sat inside the Cluster and moved around in order to use the resources to mine for cryptocurrency. Monitoring the traffic allows malicious characteristics of a packet to be detected within the network and alert the owner if anything needs to be changed.

6.2 Future Work

In terms of intrusion detection in a Kubernetes Cluster, the ideal and optimal solution would consist of a model with all the strengths of the previous attempts of implementing this solution. The solution will work in real-time and will also have a 100% success rate at identifying malicious packets. This ideal solution will make sure the entire network is monitored and will analyse every packet and traffic flow pattern in order to detect an intruder.

The solution proposed in this dissertation demonstrates a proof of concept that describes how intrusion detection in a Kubernetes Cluster can be achieved through the use of traffic monitoring and machine learning. It can also be developed further to get closer to the optimal solution. This section describes the

different ways this solution can be improved in the future or how similar solutions can be implemented with a different design.

6.2.1 Real Time Monitoring

One of the biggest drawbacks of this model is that it does not monitor the network traffic in real time. As demonstrated in the study regarding the use of sidecar containers (Karode (2020)), it is possible to implement a solution that supports real time monitoring. The packets captured in real time can be passed into the trained model and evaluated individually. Only suspicious packets will alert the administrator.

6.2.2 Additional IP Protocol Analysis

The current solution only analyses TCP and UDP packets within the captured traffic. This can be improved in the future to include other protocols such as ARP and NDP. ARP Spoofing (Imperva (n.d.)) is an attack that allows an attacker to obstruct communication links between two devices in the network. The model can detect this attack if it is updated to analyse ARP requests.

6.2.3 Monitoring Traffic Volume

This research has examined many characteristics of malicious network traffic and some, such as the IP addresses and Port numbers, are used to train the intrusion detection model. However, this model focuses on the internals of an individual packet and not the traffic as a whole. The variation in the volume of traffic is an important indicator of malicious activity and implementing a method for analysing the traffic variations will benefit this solution and increase its reliability. A new model will have to be designed that takes into account not only the individual packets but also the entire traffic flow.

6.2.4 Reduced Reliance on Training with Malicious Data

One of the challenges faced while implementing this solution is generating malicious packets that are similar to real ones. The current model depends on the quality of malicious training data in order to learn its traits and identify them in other network packets. Future work can overcome this by creating a model that does not rely on training data with malicious characteristics but instead learns the patterns of normal data and only suspects anything different to be malicious. This can also solve the need of frequent updates as the model does not need updates on malicious traffic.

6.3 Closing Remark

In this day and age, the use of containerised applications is becoming popular and container management platforms like Kubernetes are being used by many large organisations such as Capital One, Tesla and Microsoft Azure (Taylor (2020)). The large popularity has allowed its attack surface to be exposed and attackers target Kubernetes Clusters. Even though Kubernetes clusters have security protocols to handle attacks, many have gained access through different methods. This dissertation focused on an efficient method that can be used to detect intrusion within a Kubernetes Cluster. Through this study, it is evident that Kubernetes is not fully secure and attackers can execute their processes within the cluster without being detected. The existing solutions have their own drawbacks such as consuming a high level of resources or being limited to examining a certain protocol type such as HTTPS instead of the entire network traffic.

The proposed solution in this dissertation addresses the main problem of intrusion detection in Kubernetes by capturing internal network traffic and analysing it with the use of a neural network model in order to determine whether the packet is malicious or not. The solution can stand as a proof of concept for those who wish to explore further into intrusion detection in Kubernetes. The solution can also be improved further to create a solution closer to the ideal intrusion detection system desired by Kubernetes Clusters.

Bibliography

ATT&CK, M. (n.d.), ‘Mitre att&ck’.

URL: https://attack.mitre.org

Burns, B., Grant, B., Oppenheimer, D., Brewer, E. & Wilkes, J. (2016), ‘Borg, omega,

and kubernetes’, Queue 14, 70–93.

Bykova, M., Ostermann, S. & Tjaden, B. (2001), Detecting network intrusions via a statistical analysis of network packet characteristics, in ‘Proceedings of the 33rd Southeastern Symposium on System Theory (Cat. No.01EX460)’, pp. 309–314.

Datta, S. (2020), ‘Popular network protocols — baeldung on computer science’.

URL: https://www.baeldung.com/cs/popular-network-protocols die.net (n.d.), ‘apt-get(8) - linux man page’.

URL: https://linux.die.net/man/8/apt-get

Docker (2017), ‘Customize the docker0 bridge’.

URL: http://docs.docker.oeynet.com/engine/userguide/networking/default network/custom- docker0/

Docs, M. W. (2019), ‘An overview of http’.

URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview Engine, M. (n.d.), ‘Network protocols’.

URL: https://www.manageengine.com/network-monitoring/network-protocols.html for Geeks, G. (2019), ‘Supervised and unsupervised learning - geeksforgeeks’.

URL: https://www.geeksforgeeks.org/supervised-unsupervised-learning/

Hall, E. (2019), ‘Internet core protocols: the definitive guide by eric hall’.

URL: https://www.oreilly.com/library/view/internet-core- protocols/1565925726/re04.html

Helm (2019), ‘Helm’.

URL: https://helm.sh/

IBM (2019), ‘Containerization-a-complete-guide’.

URL: https://www.ibm.com/cloud/learn/containerization IBM (2020), ‘What is docker?’.

URL: https://www.ibm.com/cloud/learn/docker Imperva (n.d.), ‘Arp spoofing’.

URL: https://www.imperva.com/learn/application-security/arp-spoofing/

Karode, S. P. (2020), Monitoring kubernetes clusters with dedicated sidecar network sniffing containers, Master’s thesis, School of Computer Science and Statistics at Trinity College Dublin.

Kubernetes (2021a), ‘Install tools’.

URL: https://kubernetes.io/docs/tasks/tools/

Kubernetes (2021b), ‘Kubernetes’.

URL: https://github.com/kubernetes/kubernetes/

Kubernetes (2021c), ‘Kubernetes components’.

URL: https://kubernetes.io/docs/concepts/overview/components/

Kubernetes (2021d), ‘Pods’.

URL: https://kubernetes.io/docs/concepts/workloads/pods/

Kubernetes (2021e), ‘Service’.

URL: https://kubernetes.io/docs/concepts/services-networking/service/

Kubernetes (2021f), ‘Using coredns for service discovery’.

URL: https://kubernetes.io/docs/tasks/administer-cluster/coredns/

Lens (n.d.), ‘Lens documentation’.

URL: https://docs.k8slens.dev/v4.2.2/

Loshin, P. & Cobb, M. (2019), ‘Secure shell (ssh)’.

URL: https://searchsecurity.techtarget.com/definition/Secure-Shell McCune, R. (n.d.), ‘A hacker’s guide to kubernetes security’.

URL: https://techbeacon.com/enterprise-it/hackers-guide-kubernetes-security

Microsoft (2020), ‘Threat matrix for kubernetes’.

URL: https://www.microsoft.com/security/blog/2020/04/02/attack-matrix- kubernetes/

Nighania, K. (2019), ‘Various ways to evaluate a machine learning models performance’.

URL: https://towardsdatascience.com/various-ways-to-evaluate-a-machine-learning-models-performance-230449055f15

NS1 (2018), ‘Dns protocol explained’.

URL: https://ns1.com/resources/dns-protocol

Palmer, M. (n.d.), ‘Kubernetes networking guide for beginners’.

URL: https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-networking-guide-beginners.html

Panagiotis, M. (2020), Attack methods and defenses on kubernetes, Master’s thesis, Department of Digital Systems at University of Piraeus.

URL: https://dione.lib.unipi.gr/xmlui/bitstream/handle/unipi/12888/Mytilinakis mte1822.pdf

Pant, A. (2019), ‘Introduction to machine learning for beginners’.

URL: https://towardsdatascience.com/introduction-to-machine-learning-for- beginners-eed6024fdb08

ReviverSoft (n.d.), ‘Pcap file extension - what is .pcap and how to open?’.

URL: https://www.reviversoft.com/en/file-extensions/pcap

Scikit, L. (n.d.a), ‘1.17. neural network models (supervised) — scikit-learn 0.23.1 documentation’.

URL: https://scikit-learn.org/stable/modules/neural networks supervised.html

Scikit, L. (n.d.b), ‘Getting started — scikit-learn 0.23.2 documentation’. URL: https://scikit-learn.org/stable/gettingstarted.html

Shah, J. (2018), Detection of malicious encrypted web traffic using machine learning, Master’s thesis, Department of Electrical and Computer Engineering at University of Victoria.

URL: http://dspace.library.uvic.ca/bitstream/handle/1828/10313/Shah Jay MEng 2018.pdf

Sharma, S. (2017), ‘Activation functions in neural networks’.

URL: https://towardsdatascience.com/activation-functions-neural-networks- 1cbd9f8d91d6

Singer, G. (2020), ‘Threat alert: Kinsing malware attacks targeting container environments’.

URL: https://blog.aquasec.com/threat-alert-kinsing-malware-container-vulnerability

Song, D. (2019), ‘Dpkt — dpkt 1.9.2 documentation’.

URL: https://dpkt.readthedocs.io/en/latest/

Sookocheff, K. (2018), ‘A guide to the kubernetes networking model’.

URL: https://sookocheff.com/post/kubernetes/understanding-kubernetes-networking- model/#pod-to-pod

Taylor, T. (2020), ‘5 kubernetes security incidents and what we can learn from them’.

URL: https://techgenix.com/5-kubernetes-security-incidents/ Tcpdump (n.d.), ‘Tcpdump & libpcap’.

URL: https://www.tcpdump.org/index.html

Tsunoda, H. & Keeni, G. M. (2012), Security by simple network traffic monitoring, in ‘Proceedings of the Fifth International Conference on Security of Information and Networks’, SIN ’12, Association for Computing Machinery, New York, NY, USA, p. 201–204.

URL: https://doi-org.elib.tcd.ie/10.1145/2388576.2388608

Uniqtech (2018), ‘Multilayer perceptron (mlp) vs convolutional neural network in deep learning’.

URL: https://medium.com/data-science-bootcamp/multilayer-perceptron-mlp-vs-convolutional-neural-network-in-deep-learning-c890f487a8f1

V K, A. (2019), ‘What is deep learning: Definition, framework, and neural networks’.

URL: https://www.toolbox.com/tech/artificial-intelligence/tech-101/what-is-deep-learning-definition-framework-and-neural-networks/

Wireshark (2017), ‘Wireshark’.

URL: https://www.wireshark.org Wordpress (2015), ‘Wordpress.com’.

URL: https://wordpress.com/create

Yamashita, R., Nishio, M., Do, R. K. G. & Togashi, K. (2018), ‘Convolutional neural

networks: an overview and application in radiology’, Insights into Imaging 9, 611–629.

Zeek (n.d.), ‘The zeek network security monitor’.

URL: https://zeek.org

Appendix A1

A1.1 Python Script for Creating a Dataset from a pcap File

```
import dpkt
from dpkt.compat import compat_ord
```

```
import socket
import random

class TrafficReader():
```

```
def __init__(self):
    self.srcIP_List = []
    self.dstIP_List = []
    self.len_List = []
```

```
self.srcPort_List = []
self.dstPort_List = []
self.malPacketCount = 5000
self.malStartIndx = 0
```

---

referenced from https://dpkt.readthedocs.io/en/latest/ →
print_packets.html

with open('capture4.pcap', 'rb') as f: pcap = dpkt.pcap.Reader(f)

for timestamp, packets in pcap:
eth = dpkt.ethernet.Ethernet(packets)

if not isinstance(eth.data, dpkt.ip.IP): print('Non␣IP␣Packet␣type␣not␣supported␣%s\n'
% eth.

→ data.__class__.__name__) continue

Extract the packet form the Ethernet frame

ip = eth.data iptype = 'None'

Extracts the Protocol type of packet for future use and → the port numbers associated with the packet.

Referenced from https://stackoverflow.com/questions

→ /55853914/need-help-outputting-the-source-and-

→ destination-port-number-to-the-user-reading if ip.p == dpkt.ip.IP_PROTO_TCP:

TCP = ip.data iptype = 'TCP'

srcport = TCP.sport

dstport = TCP.dport
elif ip.p == dpkt.ip.IP_PROTO_UDP:

UDP = ip.data

iptype = 'UDP' srcport = UDP.sport dstport = UDP.dport

if ip.p == dpkt.dns.DNS:

```
    print("DNS")
```

```
self.srcIP_List.append(self.inet_to_str(ip.src))
self.dstIP_List.append(self.inet_to_str(ip.dst))
self.len_List.append(ip.len)
self.srcPort_List.append(srcport)
self.dstPort_List.append(dstport)
```

```
# Function is taken from https://dpkt.readthedocs.io/en/latest/    →
_modules/examples/print_packets.html#mac_addr

def mac_addr(self, address):
```

```
"""Convert a MAC address to a readable/printable string
Args:

address (str): a MAC address in hex form (e.g. '\x01\x02\x03\    → x04\x05\x06')
```

Returns:
    str: Printable/readable MAC address

"""

return ':'.join('%02x' % compat_ord(b) for b in address)

Function is taken from https://dpkt.readthedocs.io/en/latest/ →
_modules/examples/print_packets.html#inet_to_str

def inet_to_str(self, inet):

  """Convert inet object to a string

Args:
    inet (inet struct): inet network address

Returns:
    str: Printable/readable IP address

"""

First try ipv4 and then ipv6 try:

return socket.inet_ntop(socket.AF_INET, inet)

---

except ValueError:
return socket.inet_ntop(socket.AF_INET6, inet)

Returns the index where the malicious data will start

```
def getMid(self):
    length = len(self.dstIP_List)
    mid = int(length * 0.8)
    return mid
```

Returns a dataset object containing normal and malicious data

```
def getTrafficObj(self):
    self.addFailData()
```

---

data = { 'Src_IP':self.srcIP_List, 'Dst_IP':self.dstIP_List

'Length':self.len_List, 'SrcPort':self.srcPort_List,

'DstPort':self.dstPort_List,

} return data

def getNormalPacketCount(self):

return len(self.len_List) - self.malPacketCount

def getMaliciousPacketCount(self):

return self.malPacketCount

```
def getMalDataIndex(self):
    return self.malStartIndx
```

```
# Create Malicious Packets and Add to dataset

def addFailData(self):
    mid = self.getMid()
    self.malStartIndx = mid
```

```
for x in range(self.malPacketCount):
    index = mid + x

\# Malicious characteristics chosen for the current packet    → being
created
```

```
port = bool(random.getrandbits(1))
ip = bool(random.getrandbits(1))
size = bool(random.getrandbits(1))

if (port == False and ip == False and size == False) :
```

---

size = True

Size is malicious

if size == True:

Small range or Large range

maliciousRange = random.getrandbits(1)

malSize = 0

---

length is too small

```
if maliciousRange == 0:
    malSize = random.randint(0, 50)


# length is too small
```

```
elif maliciousRange == 1:
    malSize = random.randint(8000, 65535)


self.len_List.insert(index,malSize)
```

```
# Size not malicious


else :
    self.len_List.insert(index,random.randint(51, 7999))
```

```
# Port number is malicious
```

```
if port == True:
    maliciousPortChoice = random.randint(0, 2)
```

```
\# Port number is 0
if maliciousPortChoice == 0:

   portType = random.randint(0, 1)
    if portType == 0:

      self.dstPort_List.insert(index,0)
```

```
   self.srcPort_List.insert(index,72)
elif portType == 1:

   self.dstPort_List.insert(index,443)
    self.srcPort_List.insert(index,0)
```

```
# Source port number is 22


if maliciousPortChoice == 1:
    self.dstPort_List.insert(index,22)
    self.srcPort_List.insert(index,8000)
```

```
# Source and Destination port numbers are the same


if maliciousPortChoice ==



self.dstPort_List.insert(index,39066)
self.srcPort_List.insert(index,39066)
```

```
# Port number is not malicious


else :
    self.srcPort_List.insert(index,39066)
    self.dstPort_List.insert(index,8443)
```

```
# IP address is malicious. It is unfamiliar to the network

if ip == True: self.srcIP_List.insert(index,'192.168.0.110')
```

```
self.dstIP_List.insert(index,'192.168.49.2')

# IP address is not malicious

else : self.srcIP_List.insert(index,'172.17.0.2')
```

```
self.dstIP_List.insert(index,'172.17.0.1')
```

Listing A1.1: This Python class creates a dataset from a pcap file

```
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.preprocessing import LabelBinarizer, LabelEncoder
from sklearn.model_selection import train_test_split
from tabulate import tabulate
import pandas as pd

import trafficReader
```

```
import numpy as np

# This class is based on an example from https://www.pluralsight.com/ →
guides/machine-learning-neural-networks-scikit-learn
```

```
# Intialise data of lists.

tr = trafficReader.TrafficReader()
data = tr.getTrafficObj()
```

Create DataFrame

df = pd.DataFrame(data)
table_df = pd.DataFrame(data)

Create labels to replace string values
Referenced from https://stackoverflow.com/questions/47312695/python-

→ sklearn-value-error-could-not-convert-string-to-float str_columns =
df.columns[df.columns.str.contains('(?:Protocol|IP)')]

---

labels = {c:LabelEncoder() for c in str_columns}

Replace values in the dataset with their respective labels

for col, label in labels.items():
    df[col] = labels[col].fit_transform(df[col])

```
# Normalise all the values in the dataset

target_column = ['1']
col_label = list(set(list(df.columns))-set(target_column))

df[col_label] = df[col_label]/df[col_label].max()
```

```
# Create an array of expected results for the dataset

y = listofzeros = [0] * tr.getNormalPacketCount()
length = len(listofzeros)
mid = tr.malStartIndx
```

```
for x in range(tr.getMaliciousPacketCount()):
    index = mid + x
    y.insert(index, 1)
```

Divide the dataset by 8:2 to obtain the training and testing datasets → respectively.

```
cutoff = int(tr.getMalDataIndex() + (tr.malPacketCount * 0.8))
train_df = df.iloc[:cutoff]
```

```
test_df = df.iloc[cutoff:]
```

```
train_results = y[:cutoff]
test_results = y[cutoff:]
```

Initialise and train the MLP using the training dataset

```
clfr = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, → 2),
random_state=1, max_iter=400)
```

```
clfr.fit(train_df, train_results)
```

Test the trained model

```
ans = clfr.predict(test_df)

# Draw Table to display results
```

```
test_table_df = table_df.iloc[cutoff:]
malCnt = 0
normCnt = 0
packetVerdict = []
```

```
packetAnalysis = []
for x in range(len(ans)):

val = ans[x]
```

CHAPTER 8

CONCLUSION

In this paper we present Machine learning integration with Devops CI/CD principles to improve the performance and business values .we present use case of Machine Learning methodology to solve the industry problem CRISP CM is one of the best machine learning methodology used in Data science projects. We found that manual machine learning needs a lot of resources and manpower and it lags the business organization. Automated machine learning with devops improves a lot of things in today's data science industry it produces great results in business, marketing and also it produce less waste as compared to manual machine learning, It makes deployment and integration easier. Machine learning model lifecycle is different from actual software development .it requires a lot of things data collection data cleaning, feature selection, setup environment after deployment monitoring and maintenance takes place ,Machine learning alone can't do these things effectively, Devops provide continuous integration and continuous deployment principles for these type of problems. development and operational teams are working in many areas like manufacturing industry, marketing industry ,healthcare industry but they are not achieve greater efficiency because they are not integrate machine learning with devops continuous development. This study shows that integration of machine learning with devops (MLOPs) In this case both development and operational team work together to optimize the process and produce great results in data science industry

REFERENCES

[1] A research paper on Application of Devops in the improvement machine learning process

[2]A research paper on Applying DevOps Practices of Continuous Automation for Machine Learning

[3] Beginning MLOps with MLFlow Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure by Sridhar AllaSuman and Kalyan Adar

[4] Data Science Solutions on AzureTools and Techniques Using Databricks and MLOps by Julian Soh and Priyanshi Singh

[5] Master's thesis Master's Programme in Data Science Designing an open-source cloud-native MLOps pipeline by Sasu Mäkinen March 12, 2021

[6]https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automationpipelines-in-machine-learning

[7]https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuousdelivery-explained.html

[8] https://www.datascience-pm.com/crisp-dm-2/

[9] https://en.wikipedia.org/wiki/DevOps

[10] https://www.docker.com/resources/what-container

[11] https://www.redhat.com/en/topics/devops/what-cicd-pipeline

[12] https://www.jenkins.io/doc/