National College of Ireland

Improving Probes and checks in Kube-Hunter to evaluate and repair Security Vulnerabilities in the Kubernetes manifest file.

MSc Research Project
Cloud Computing

Tenzin Tsephel
Student ID: x21176574

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Tenzin Tsephel
Student ID:	x21176574
Programme:	Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	14/08/2023
Project Title:	Improving Probes and checks in Kube-Hunter to evaluate and repair Security Vulnerabilities in the Kubernetes manifest file.
Word Count:	4081
Page Count:	17

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Tenzin Tsephel
Date:	7th August 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Improving Probes and checks in Kube-Hunter to evaluate and repair Security Vulnerabilities in the Kubernetes manifest file.

Tenzin Tsephel
x21176574

Abstract

This work is to solve the common security flaws discovered in Kubernetes manifest files that can be used to alter or obtain unauthorized access to objects in the Kubernetes cluster adept at identifying vulnerabilities at the cluster level, the Kube-Hunter program has limits when it comes to protecting it. In order to improve Kube-Hunter and make it more efficient at identifying and fixing file vulnerabilities, this work adds new code to its probes and checks. The algorithms for Role-Based Access Control (RBAC) Policy Misconfiguration Check be used to detect and fix improper container communication and excessively permissive access permissions, respectively. By delivering a more thorough security assessment of Kubernetes settings and encouraging safer Kubernetes usage in businesses, this research covers a better Kube-Hunter. Title: Does the Role-Based Access Control (RBAC) Policy Misconfiguration Check and Network Policy Misconfiguration Check algorithm enhances the Kube-hunter tool to detect and remediate security vulnerabilities?

Keywords:Kube-hunter, Role Base Access Control(RBAC), Kubernetes, Docker, Python.

1 Introduction

The digital revolution has fundamentally changed how organizations function and provide value. Businesses have embraced containerized solutions as a result of the requirement for digital infrastructure that is high-performing, resilient, and scalable. Kubernetes is a noteworthy solution that has surfaced in the modern digital scene. Kubernetes automates the deployment, scaling, and management of containerized applications as a potent open-source solution. Basically, it orchestrates container activities, greatly streamlining complicated procedures Burns et al. (2022).

Due to its extensive feature set, Kubernetes has swiftly taken over as the industry norm for container orchestration. It provides a wide range of capabilities that improve the productivity and effectiveness of software delivery. Just a few examples of Kubernetes' features include automated rollouts and rollbacks, horizontal scalability, load balancing, service discovery, and storage orchestration Burns et al. (2022). In Table 1 gives a brief overview of each component in general and This is a nice little introduction with some Architecture Workflow Figure 1

Nevertheless, despite the numerous benefits Kubernetes provides, using it is not without difficulties. Ensuring the security of Kubernetes manifest files is one of the

Kubernetes Component	**Basic Use-Case**
Pod	Smallest unit in Kubernetes model you create or deploy.
Service	Exposes an application running on a set of Pods as a network service.
Kube-proxy	Network proxy which reflects services as defined in Kubernetes API on each node.
API Server	Serves the Kubernetes API and acts as the frontend for Kubernetes control panel.
Scheduler	Schedules pods onto Nodes, considering resource availability.
etcd	Stores all cluster data used by Kubernetes to manage its state.
Controller	Responds to events within the cluster to maintain desired state of workloads.
Kubelet-agent	Ensures that containers are running in a pod on the node.
Secret	Stores sensitive information, like passwords and keys.

Table 1: Basic Use-Cases of Kubernetes Components

key challenges involved with this system. In the Kubernetes design, Kubernetes manifest files are essential. The instructions for building and administering Kubernetes objects, including pods, services, and deployments, are included in these files, which are commonly expressed in YAML or JSON. In essence, manifest files specify the desired state of each object in a Kubernetes cluster Martin and Martin (2021). As a result, they are very important in managing how the cluster's applications are executed Calderón-Gómez et al. (2021).

Kubernetes manifest files are essential, but despite this, they frequently include security flaws. These flaws can be attributed to a number of things, including poor role-based access control (RBAC) Burns et al. (2022), soft network regulations, and setup mistakes. These flaws might be used to gain access to Kubernetes clusters without authorization, alter Kubernetes objects, or even launch serious cyberattacks if left unchecked.

Case studies from the real world have shown the possible dangers of Kubernetes security. For instance, the electric vehicle manufacturer Tesla experienced a serious data breach in 2018 [1]. A Kubernetes console that was not password-protected was blamed for the incident. The console was accessed by cybercriminals, which exposed private information. The significance of correctly configured Kubernetes environments was amplified by this occurrence.

In another instance, a significant vulnerability in the Kubernetes API server was discovered in 2019 by Shopify, a well-known e-commerce site. Unauthorized requests might be accepted due to this vulnerability, which could result in data breaches or service interruptions [2]. These occurrences serve as a stark reminder of the security threats that

[1]Tesla Hack:`https://www.wired.com/story/cryptojacking-tesla-amazon-cloud/`

[2]Shopify:`https://www.hackerone.com/application-security/cloud-security-alliance-webin/ar-recap-avoid-breach-shopifys-andrew-dunbar`

Kubernetes deployments may experience. They emphasize how critical it is to have strong security solutions for Kubernetes security.

Kube-Hunter is a well-liked tool for identifying security flaws in Kubernetes clusters. A Kubernetes cluster is subjected to multiple probes and inspections by Kube-Hunter, an open-source vulnerability scanner. These tests and probes aid in locating potential security flaws. Significant insights into the security posture of Kubernetes clusters are frequently provided by Kube-Hunter. Although Kube-Hunter has shown to be useful in locating cluster-level vulnerabilities, it has some drawbacks.

Kube-Hunter's narrow emphasis on Kubernetes manifest files is a noteworthy drawback. Kubernetes manifest files may contain possible security flaws, however, Kube-Hunter does a great job of discovering any security breaches at the cluster level [3]. Due to the need for both cluster-level security and object-level security in Kubernetes systems, this constraint presents a significant difficulty. Consequently, there is a definite need for a more all-encompassing solution that can efficiently find and fix security flaws in Kubernetes manifest files.

Therefore, the goal of this study is to determine whether the Kube-hunter tool's ability to detect and fix security vulnerabilities is improved by the Role-Based Access Control (RBAC) Policy Misconfiguration Check.

The remainder of this document is structured as follows: The second section examines related research on Kubernetes security and the application of technologies like Kube-Hunter. The research methodology is described in Section 3 along with the data collecting, statistical methods, and research and assessment processes that were used. The design parameters for the new algorithms proposed to improve the Kube-Hunter tool are presented in Section 4 in detail. The execution of the suggested fix is discussed in Section 5. Section 6 offers an assessment by analyzing the findings and their consequences.

1.1 Research Topic

The main focus of this research is filling this gap. We suggest enhancing Kube-Hunter's skills to effectively assess and fix security flaws in Kubernetes manifest files."How Kube-Hunter can be enhanced to effectively evaluate and repair security vulnerabilities in Kubernetes manifest files from RBAC algorithm?" is the main research topic that directs the inquiry.

1.2 Motivation

In order to explicitly target potential vulnerabilities within Kubernetes and its manifest files, this research work entails building and implementing additional probes and checks into Kube-Hunter. Additionally, we want to create an algorithm that can automatically fix found flaws, enabling a setup that is more reliable and secure. This will explore the more thorough approach to guaranteeing the security of Kubernetes environments by concentrating on the security of Kubernetes manifest files. Additionally, we want to improve the effectiveness of security operations within Kubernetes and decrease the time and effort needed to remediate security vulnerabilities by automating the repair process.

The findings of this research work will have broad ramifications for enterprises using Kubernetes. We can assist enterprises in strengthening their security posture and lowering the risk of cyber-attacks by offering a tool that locates and fixes security flaws in

[3]Aquasec: `https://www.aquasec.com`

Kubernetes manifest files. Additionally, the results of this research may have a substantial impact on the larger field of cyber security. It contributes to the improvement for the knowledge of Kubernetes security and provides fresh perspectives on the creation of efficient security tools for containerized environments.

In summary, Kubernetes has become a powerful tool for container orchestration, yet security issues persist Burns et al. (2022). This project seeks to significantly contribute to improving the security of Kubernetes environments and expanding the area of cyber security by concentrating on improving Kube-Hunter's skills to find and fix vulnerabilities in Kubernetes manifest files.

1.3 Work Format

The rest of this essay is structured as follows: The full literature review on Kubernetes security and the existing mitigation techniques are presented in the following section. The approach employed in this work endeavour is described in the part that follows. The outcomes of the inquiry are then discussed. The discussion of the consequences of the findings and recommendations for further research serves as the paper's conclusion.

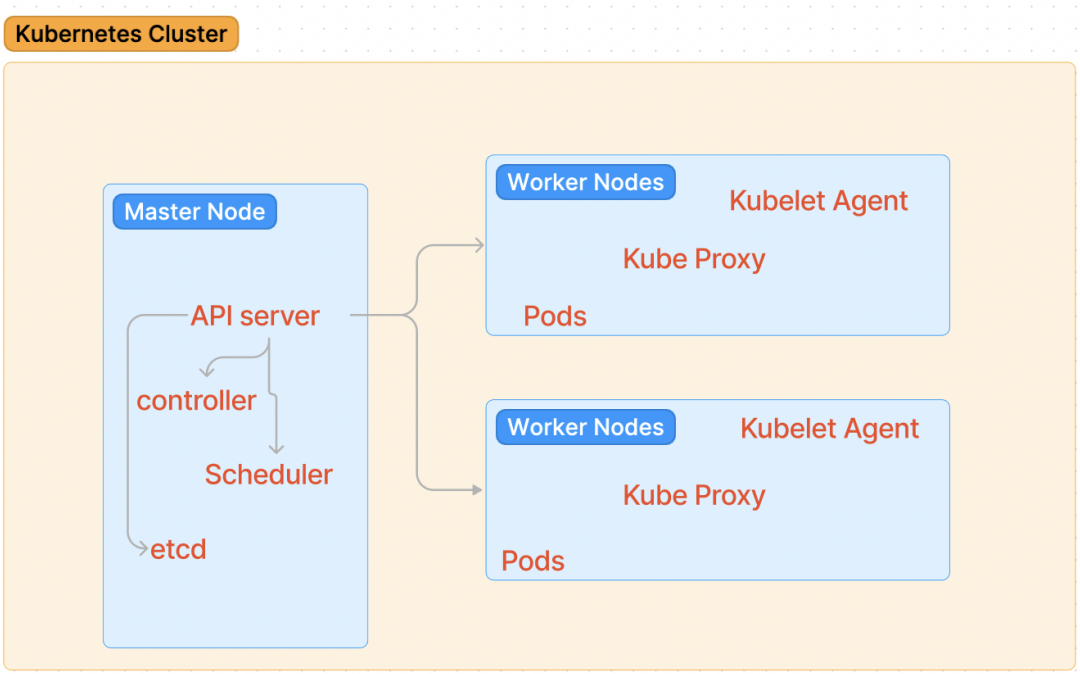

Figure 1: Kubernetes Architecture

2 Related Work

Before embarking onto the technical depth of Kubernetes Security, it is essential to know the fundamentals and work related to Kubernetes and it's security. One of the key aspects is fundamentally based on authentication and authorisation. By employing authentication services like OAuth, OpenID Connect, or client certificates, authentication establishes the legitimacy of a person or system trying to access the Kubernetes cluster.

On the other hand, authorization decides whether to give or deny access to particular resources within the Kubernetes cluster based on the rights and roles of the authenticated user. The Kubernetes Role-Based Access Control (RBAC) mechanism controls this procedure.

2.1 Microservices Kubernetes

Microservices, commonly referred to as the microservices architecture, organize a program as a group of tiny, independent services, each based on a distinct industry. Each service is independent of the others and implements a specific business capability, enabling autonomous development, deployment, updating, and scalingVayghan et al. (2018). Because services are independent of one another, distributed development is possible using the technologies best suited to each service's requirements. This independence also assures fault isolation and gives the system scaling options Dragoni et al. (2017). The decoupling of the services is further ensured by the normal management of each service's own independent database or data store. Microservices provide several benefits, including increased scalability, flexibility, and resilience, but they also pose difficulties in terms of complexity management, assuring data consistency, and handling inter-service communication.

2.1.1 Role-Based Access Control (RBAC)

The authors [8] discussed A Role-Based Access Control (RBAC) project's successful implementation at Siemens ICN. The necessity of establishing role definitions inside corporate functional structures rather than less reliable organizational structures was one of the main conclusions. Existing job descriptions were inappropriate for defining positions in terms of specific access permissions because they lacked sufficient information. In order to develop role definitions, the project turned to business process documentation. The project placed a strong emphasis on the need for security rules to be intimately connected to the business processes that IT systems support in order to ensure a more thorough understanding and revision of access rights choices. Furthermore, because of aspects like workload, change management, and documentation, it is critical in large businesses to use tools for job identification and maintenance Roeckle et al. (2000). Hence, The significance of strong authentication and authorisation systems within Kubernetes has been highlighted by previous studies. The effective incorporation of this security data into process models served as the basis for the creation of role catalogues using tools. This strategy produced thorough models that successfully combined the process, role, and access rights levels, improving the administrative chores in an RBAC system.

2.1.2 Kubernetes Security Policy

The establishment of Kubernetes-specific security policies is crucial for protecting Kubernetes clusters. These policies establish guidelines and limitations intended to protect the Kubernetes cluster from possible security threats Panagiotis (2020). These may include recommendations for network segmentation, access controls, container isolation, and authentication and authorization procedures. The researcher emphasizes how Kubernetes' initial design, particularly its placement policy, has limits when it comes to effectively handling the dynamic and variable nature of emerging geo-distributed systems Rossi et al.

(2020). The authors respond to this by introducing ge-kube, a Kubernetes extension that includes network-aware scheduling tools and self-adaptation methods.

This research does not specifically address security, but by ensuring that programs are put and scaled correctly in accordance with network parameters and performance data, these enhancements may inadvertently improve the security of applications running on geographically distributed resources. The application and maintenance of Kubernetes' built-in security capabilities, such as Generic Policies, Pod Security Policies, and Network Policies, can be challenging. The need for a simpler, more streamlined approach to policy administration is frequently mentioned in literature. According to several studies, this issue might be resolved by the creation of further Kubernetes features or third-party solutions.

2.1.3 Vulnerability scanning

A crucial security procedure known as vulnerability scanning involves the identification and assessment of potential security flaws in a system or application. Software tools that scan the system or application for known security weaknesses like out-of-date software, configuration problems, or unpatched vulnerabilities are frequently used in this process. The need for vulnerability screening to ensure secure deployment is frequently mentioned in the existing literature on Kubernetes Shamim et al. (2020). Studies have shown that a number of variables, including the calibre of the vulnerability databases used by the scanning tools and the possibility of false positives and negatives, might restrict the efficacy of these scans. The authors Bose et al. (2021) stress the need to quickly identify and fix security flaws in Kubernetes manifests. It concluded that a significant 0.79% of contributions are security-related after examining 5,193 commits from 38 open-source repositories, indicating a problem with under-reporting in the sector. This discovery is significant for improving the probes and checks in Kube-Hunter, a program created to identify and fix security flaws in Kubernetes. In line with the paper's recommendation for rigorous research to prevent potential significant security breaches, Kube-Hunter can be better equipped to find and address latent vulnerabilities by comprehending the nature and prevalence of these under-reported faults.

2.1.4 Systems Log

The Kubernetes cluster's components and running containers generate log data, which is collected and archived as part of the logging process. Kubernetes comes with a number of built-in logging features, including the capacity to read and manage logs via the Kubernetes API, interface with external logging services, and send container output to log files. Despite being crucial, logging is frequently mentioned as one of the more difficult facets of Kubernetes security Burns et al. (2016). According to the study, uses a log model technique to create a diagnostic tool for declaratively deployed cloud apps. The research uses clustering methods to collect typical examples while utilizing the stability of cloud platforms, where the majority of workloads are in their desired state, to construct the model. Because it places less emphasis on the chronological sequence of log entries when building and matching the reference model, the tool, dubbed LogDC, can diagnose problems during an application's whole lifecycle Xu et al. (2017) Kubernetes' decentralized architecture can make thorough logging challenging. Additionally, examining logs for potential security concerns necessitates a significant amount of knowledge and resources.

2.1.5 Kubernetes Namespace

The resources in a cluster can be divided using Kubernetes namespaces so that each namespace can have its own set of resources with unique names Ibryam and Huß (2022). This functionality helps preserve isolation and makes cluster management easier for teams or projects that share a single cluster. Although namespace separation is a useful feature, studies have shown that it is possible for user errors and configuration mistakes, which could result in security problems. As a result, stricter regulations are required to enforce namespace separation.

The distributed key-value store etcd is used by Kubernetes to store crucial cluster data, including resource state and configuration options Ibryam and Huß (2022). To guarantee the security and integrity of this data, it is essential to encrypt and restrict access to the etcd cluster. Despite the fact that Kubernetes has procedures for protecting etcd, such as authentication, authorisation, and transport security, studies have shown that mistakes or neglect in these areas can result in the exposure of crucial data. Furthermore, strong disaster recovery strategies and the significance of backing up etcd data are regularly emphasized in the literature.

2.1.6 Security and performance

The study of Rahman et al. (2023) examines security flaws in Kubernetes manifest files. The authors propose 11 different categories of security misconfigurations and quantify them using their static analysis tool, SLI-KUBE, through a thorough empirical investigation of thousands of Kubernetes manifests from 92 open-source sources. The discovery of 1,051 security misconfigurations highlights the value of security-focused code reviews and the usage of static analysis during the creation of Kubernetes manifests. It serves as a crucial reference point for enhancing the probes and tests in Kube-Hunter, a program designed to identify and fix security flaws in Kubernetes manifest files. The other researchers Viktorsson et al. (2020) compares and contrast Kata and gVisor, two Kubernetes container runtimes, in terms of security and performance. The study's findings suggest that more stringent security controls are associated with higher costs, both in terms of deployment time and potential application performance losses. Interestingly, Kata outperformed gVisor in terms of application performance for both compute and network-bound apps, while having more security layers and subsequently longer deployment durations. According to the study, the security environment for container runtimes is quickly changing, and regular evaluations by academics are essential for comprehending current trends, design trade-offs, and potential constraints.

2.1.7 Rollbacks mechanism

New versions of applications or services can be delivered to a cluster without experiencing any downtime or disruptions thanks to Kubernetes' continuous update functionality. Rolling updates, which gradually replace outdated versions with new ones while maintaining a sufficient number of replicas and evenly distributing traffic, are frequently used to do this. Despite its value in keeping applications and services current, literature has pointed out potential security issues connected with ongoing updates Mendonça et al. (2019). Making sure that updated containers don't introduce new vulnerabilities into the cluster is a huge problem. The Kubernetes community has also talked a lot about managing rollbacks in case of incorrect updates and assuring zero downtime during updates.

2.1.8 Infrastructure as a service

Helm charts and Kubernetes manifest files are crucial tools for creating and maintaining applications that are deployed on Kubernetes. Helm charts make it easier to deploy and manage applications, whereas manifest files specify the expected state for an object in a cluster Sayfan (2017). Numerous studies stress how crucial it is to manage Helm charts and manifest files appropriately in order to avoid security problems [16]. For instance, unsecured dependencies in Helm charts or the exposing of sensitive data in manifest files might have serious security repercussions. However, research suggests that these factors are frequently disregarded in Kubernetes security procedures. The thorough study of gray literature from Kumara et al. (2021) demonstrates how academics and practitioners alike are becoming more interested in Infrastructure-as-Code (IaC). The paper examines implementation challenges, design issues, and fundamental concepts while classifying the best and worst IaC techniques utilizing languages like Ansible, Puppet, and Chef. The knowledge gained from this study into the creation and upkeep of IaC can be used to improve Kube-Hunter's probes and checks. The approaches for identifying and fixing security vulnerabilities in Kubernetes manifest files can be improved and enhanced by following best practices and avoiding known problems. Utilizing the knowledge gained from the larger IaC community, the investigation of common practices in IaC languages can result in a more robust, effective, and secure implementation in the Kubernetes environment.

2.1.9 Security Challenges ahead

Common security flaws in Kubernetes clusters are frequently hunted for using the open-source application Kube-Hunter. But current research indicates that while Kube-Hunter is good at spotting known vulnerabilities, it might not be as good at spotting more complex or obscure threats. Additionally, its focus is typically on runtime security and excludes the examination of Helm charts or Kubernetes manifest files. The authors at Minna et al. (2021) state, when analyzed using the standard security paradigm, Kubernetes' network abstractions may result in "unexpected attacks" despite making it easier to deploy and manage cloud applications. This change in network security modelling has significant effects on how vulnerabilities are located, assessed, and fixed in Kubernetes manifests. Thus, the Kube-Hunter tool can be viewed as a crucial tool for adapting established security procedures to a cloud-native environment. Kube-Hunter's probes and checks can be adapted to stop the unintended attack vectors identified in this study by identifying and addressing the unique networking components and potential weak points in Kubernetes.

There are still some security gaps in a number of areas, including authentication, authorization, policy management, vulnerability scanning, logging, namespace separation, network security, continuous updates, and manifest file and despite the fact that the security of Kubernetes has been significantly improved by current tools and methodologies Carrión (2022). There is a need for enhancement so that tools like Kube-Hunter can handle a larger variety of potential dangers, even if they are essential for locating and addressing security flaws Binnie and McCune (2021). This highlights the need for ongoing research on Kubernetes security, such as the research that the study proposes Revuelta Martinez (2023).

2.2 Analysis and gaps

Related Work	Concept	Highlights	Issues/Gaps
2.1.1 Role-Based Access Control (RBAC)	Security rules based on business processes	Focus on the necessity of establishing role definitions in corporate structures. Utilization of business process documentation for role definitions.	The inadequacy of existing job descriptions in defining roles based on specific access permissions.
2.1.2 Kubernetes Security Policy	Kubernetes-specific security policies	Introduction of ge-kube, an extension that includes network-aware scheduling tools and self-adaptation methods.	Limitations in Kubernetes' initial design in handling the dynamic nature of emerging geo-distributed systems. Need for a simpler policy administration.
2.1.3 Vulnerability Scanning	Identification and Assessment of potential security flaws	Use of software tools for Scanning system or Applications for known security weaknesses.	Constraints in the efficacy of scans due to the quality of vulnerability databases and the possibility of false positives/negatives.
2.1.4 Systems Log	Kubernetes logging features	Utilization of a log model technique for creating a diagnostic tool for cloud apps.	Difficulty in thorough logging due to Kubernetes' decentralized architecture. Need for substantial knowledge and resources for examining logs.
2.1.5 Kubernetes Namespace	Namespace division in Kubernetes clusters	Facilitates resource isolation and easier cluster management.	Possibility of user errors and configuration mistakes leading to security issues. Need for stricter regulations for namespace separation.
2.1.6 Security and Performance	Comparison of Kata and gVisor runtimes	Kata outperforms gVisor in terms of application performance while providing more security layers.	Increased security controls might lead to higher costs in terms of deployment time and potential performance losses.

Table 2: Comparative Analysis of Related Works

3 Research Methodology

The methodology used in this study is a hybrid of quantitative and qualitative research methods. This mixed method study strategy was chosen because it gives a balanced manner to evaluate the effectiveness of the Kube-Hunter upgrades. It entails the creation and integration of algorithms, as well as a thorough examination of the produced data to assess their effectiveness in detecting and remediating vulnerabilities.

3.1 From Selection to Execution

The presented technique was chosen to improve the Kube-Hunter tool's capacity to find and resolve Kubernetes manifest file security vulnerabilities. The principal approach was the creation and implementation of a novel algorithm called Role-Based Access Control (RBAC) Policy Misconfiguration Check. Previous research had shown the need for RBAC in Kubernetes cluster security maintenance.

Data was collected by scanning various Kubernetes clusters known for security misconfigurations with the upgraded Kube-Hunter. These clusters, which simulated real-world settings, were created to evaluate the tool's effectiveness in finding vulnerabilities and successfully remediating them. The given Figure 2 provides the workflow of the overall implementation. This study's cluster selection reflected popular Kubernetes use cases and various security configurations, allowing for an objective evaluation of Kube-Hunter's performance.

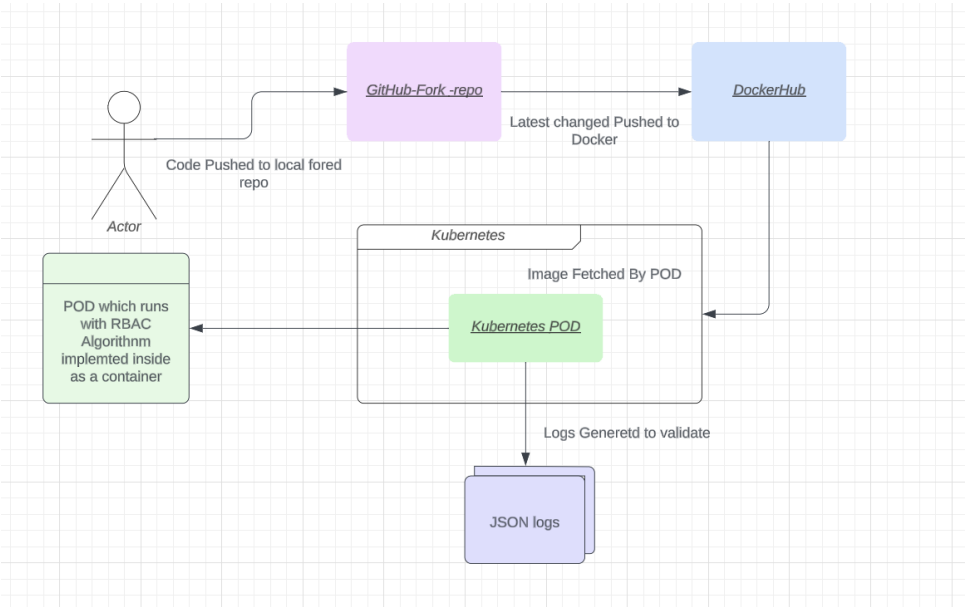

Figure 2: Workflow Architecture

Following data collection, a combination of descriptive and inferential statistical methods was used to evaluate the data. Measures of central tendency and variability were included in the high-level perspective of the data offered by descriptive statistics. On the other hand, whether the improvements to Kube-Hunter led to a statistically significant improvement in its performance required inferential statistics, specifically hypothesis testing. The development and integration of the new algorithm into Kube-Hunter, the creation of Kubernetes clusters with known security flaws, the scanning of these clusters with the enhanced Kube-Hunter and recording of the results, and finally the analysis of the gathered data using the selected statistical techniques made up the four main stages of the entire research process. This end-to-end technique made sure that every aspect of the tool's functioning, from data collecting to final outcomes, was evaluated thoroughly.

4 Design Specification

The main method used in this study was the integration of an algorithm into Kube-Hunter, which was developed to analyze Role-Based Access Control (RBAC) policies in Kubernetes manifest files Sandhu (1998). The implementation was supposed to improve Kube-Hunter's capacity to detect and fix potential security flaws in RBAC policies. Despite careful design, the algorithm ran into unexpected challenges, exposing the real-world difficulties inherent in Kubernetes security.

4.1 Algorithm description

The RBAC Policy Misconfiguration Check algorithm, developed to investigate Kubernetes RBAC objects in an unusual two-tier scrutiny procedure, is at the heart of this research paper. The first tier examines the access rights provided to roles and cluster roles, looking for unduly liberal access rights. The second tier examines role bindings and cluster role bindings with the goal of identifying situations where subjects such as users, groups, or service accounts have been given incorrect access. The dual-layered analysis was created to improve the accuracy of detecting potential security flaws and ensure a complete examination. The below Figure 3 depicts the flow of the RBAC algorithm.

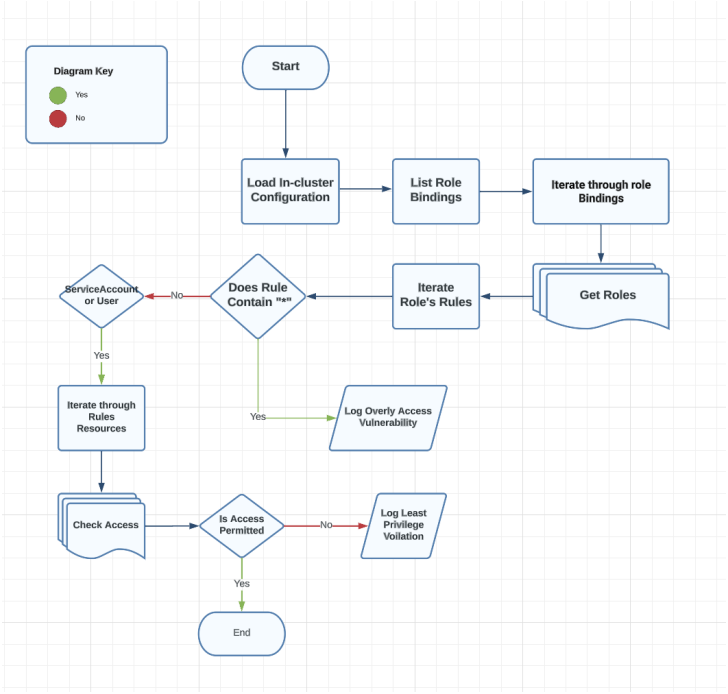

Figure 3: Algorithm structure Flow

However, "forbidden" errors occurred unexpectedly during the algorithm's execution within the Kubernetes environment. This result occurred despite the thorough design of the RBAC policies, demonstrating the unanticipated complexities and issues inherent with Kubernetes security. These findings emphasize the significance of a thorough grasp of RBAC principles, the Kubernetes authorization system, and its implementation intricacies. The technique is meant to generate a detailed report after successfully identifying RBAC misconfigurations. This study describes the RBAC misconfigurations and offers

remedial options to address the vulnerabilities. The unexpected "forbidden" faults that occurred throughout the execution highlight the need for further improvement to maintain tighter control over Kubernetes RBAC policies. The refinement phase would focus on improving the algorithm's robustness in a variety of Kubernetes scenarios, reducing the danger of unwanted access or privilege escalation.

4.2 System Requirements

The system requirements for implementing this study largely included an operating Kubernetes environment capable of simulating real-world Kubernetes usage scenarios. To allow for thorough testing of the RBAC Policy Misconfiguration Check method, this environment must have a variety of role-based access setups. Adequate computational resources were required to run the Kube-Hunter tool smoothly with the new method.

The ability to access Kubernetes manifest files was critical because they were used as input for the RBAC Policy Misconfiguration Check method. The system must also support the programming languages and libraries used in the development of Kube-Hunter and the algorithm to ensure seamless integration and operation. The "forbidden" problems discovered while running the algorithm in the Kubernetes environment indicate the need for a more sophisticated grasp of the system requirements. The experience necessitates further investigation into the complexities of Kubernetes RBAC implementation, operational permissions, and potential bottlenecks, indicating a promising field for future research and improvement.

5 Implementation

This section describes the process used to connect our Role-Based Access Control (RBAC) Policy Misconfiguration Check algorithm with Kube-Hunter. It includes a step-by-step description of the implementation process, the selection and use of required tools and languages, and the outcomes obtained throughout the implementation stage. While we avoid digging into minute code details, our goal is to provide a full understanding of the procedural stages, technological considerations, and nuances that defined our approach.

5.1 Implementation Process

The project's central component, the RBAC Policy Misconfiguration Check algorithm, was carefully developed with the intention of thoroughly inspecting Kubernetes RBAC objects and identifying any security flaws. This algorithm was created to improve Kube-Hunter's already-existing capabilities by adding the capacity to identify and fix probable RBAC policy misconfigurations. The deployment of Kube-Hunter, now using the revised algorithm, inside a working Kubernetes system served as a marker for the implementation phase. After that, scans were carried out using the tool on a variety of Kubernetes clusters. These clusters were chosen with care to represent a variety of popular Kubernetes use cases and security setups, enhancing the tool's general usefulness and resilience.

Even with a careful planning process and adherence to the RBAC guidelines, the program ran into unforeseen "forbidden" faults. The underlying complexities and difficulties that are deeply integrated throughout the Kubernetes security landscape were highlighted by this event. These unforeseen obstacles illustrate the real-world difficulties that fre-

quently follow theoretical ideas, emphasizing how crucial it is to have a comprehensive grasp of the system for successful execution.

5.2 Tools and Languages Used

The main language utilized in this project was Python, which has won praise for its adaptability and the breadth of its libraries. Its libraries for machine learning and data analysis were especially helpful in developing and testing the method. It was heavily reliant on the Kube-Hunter program, a well-known open-source utility produced by Aqua Security. The program was containerized using Docker, a platform for developing, shipping, and running applications, guaranteeing operational consistency across various deployment scenarios.

The Kubernetes environment, which was the project's main emphasis, was crucial since it served as the testing and deployment basis for the upgraded Kube-Hunter tool. This setting provided a useful setting to assess the algorithm's efficacy and efficiency in practical situations.

5.3 Outputs

The Kube-Hunter tool was enhanced during our implementation process and now includes the new RBAC Policy Misconfiguration Check algorithm. This was the most significant result. After being successfully implemented, this instrument was supposed to produce thorough reports. These reports would include important details about the precise locations of RBAC policy misconfigurations as well as recommended corrective measures.

However, there were some hiccups in our installation procedure. The unexpected "forbidden" faults produced a distinct kind of output, providing a priceless window into the intricacies and difficulties of safeguarding Kubernetes settings. These difficulties highlighted the need for additional research and improved tactics by giving us a more nuanced view of the challenges that might be encountered during such attempts. These results highlight the need for further research funding to enable the creation of even more reliable and resilient security tools for Kubernetes. This would guarantee that these tools are capable of navigating the intricate complexity of the Kubernetes security landscape successfully, hence offering thorough and reliable security solutions.

6 Evaluation

The evaluation of the research is covered in this section, with a special emphasis on the improved Kube-Hunter tool and the RBAC Policy Misconfiguration Check algorithm. Due to the nature of the research, there are no precise graphs or charts; however, this does not prevent us from doing a thorough analysis of the findings. We will thoroughly examine the research findings, focusing on issues of code implementation and system performance, and discuss their consequences from both an academic and practical standpoint.

6.1 Data Presentation

The qualitative character of the research necessitates a different format for data presentation than what is typically used, such as graphs, charts, or plots. The base conclusions

on two important sets of data: the observed system reactions, specifically the unanticipated "forbidden" faults that occurred during the tool's execution, and the performance comparison between the improved Kube-Hunter tool and other security tools. The first piece of information was taken directly from the "forbidden" error answers that were captured in the system logs when the program was being used. This information gave us an in-depth understanding of the problems that may occur even with RBAC configurations that appear to be correct, as well as the practical difficulties of securing Kubernetes deployments.

The second data point, the comparison performance, revealed the relative efficacy and accuracy of the algorithm in finding RBAC misconfigurations through parallel execution of existing tools in identical scenarios.

6.2 Data Analysis

Having thoroughly examined the system reactions and contrasting performance of the improved Kube-Hunter tool as part of the data analysis. It was discovered after examining the system's replies that " prohibited" failures mostly happened during the scanning procedure. This suggested that the permission checks in the algorithm may have gone too far, or that method may not have taken into consideration the Kubernetes RBAC's underlying complexity. This provided a crucial insight into the need for more sophisticated management of permissions at the algorithmic level for Kubernetes security features. Medel et al. (2016) The comparative performance analysis showed that the tool was able to identify potential configuration errors that other tools were unable to. The unforeseen faults, though, limited the effectiveness of the tool. This comparative analysis highlighted the tool's advantages in dealing with RBAC misconfigurations and demonstrated its potential if the issues found are effectively resolved.

6.3 Implications

The findings have important ramifications for viewpoints from the academic and professional communities. Academically, the "forbidden" mistakes that were observed highlight the difficulty of Kubernetes security and the value of taking into account practical execution issues while developing security tools. This discovery creates opportunities for additional study to comprehend these intricacies and provide more reliable security solutions. These findings emphasize the need of good permission management in Kubernetes systems from the perspective of a practitioner. The conclusions drawn from the use of the tool highlight the necessity of routinely improving and testing such security solutions and offer a realistic grasp of potential problems.

6.4 Discussion

The discussion section will go deeper into the significance of the findings, set them in the perspective of prior research, and point out any potential enhancements or changes that could be made to future studies. The discussion will critically assess the architecture of the instrument and give a more in-depth analysis of the findings in contrast to the implications section.

In order to improve the Kube-Hunter tool's ability to identify and fix any RBAC policy misconfigurations in Kubernetes settings, the study focused on the development

of the RBAC Policy Misconfiguration Check algorithm within the tool. The emergence of unexpected "forbidden" faults throughout the tool's execution revealed an essential aspect—the complicated complexity of Kubernetes environments—despite the careful design and ostensibly accurate configurations.

This finding fits with the widely held belief that Kubernetes is a complicated system that needs a thorough understanding for efficient security management, according to previous studies. The research, however, highlights the complexity and unpredictability of real-world execution even more, highlighting the need for more sophisticated and resilient security mechanisms.

This gets to the design of the tool which must be critically analyzed. The RBAC Policy Misconfiguration Check algorithm was created to thoroughly investigate role bindings and cluster role bindings, however, the existence of "forbidden" mistakes suggests that there may have been a mistake in how the program foresaw and dealt with such exceptions. Due to this, the architecture was not sufficiently robust to handle all real-world cases, despite the algorithm's potential for spotting incorrect permissions.

In light of this, we suggest two main changes for future research. First, the algorithm should be improved to incorporate more complex error handling and exception-catching techniques, guaranteeing that the tool is capable of managing unforeseen circumstances with grace. Second, more attention needs to be paid to comprehending and accommodating the many permissions and responsibilities prevalent in Kubernetes setups so that the algorithm can successfully navigate through these complexities. Additionally, it is clear that the "forbidden" mistakes need to be fixed for the tool to fully realize its potential for discovering and fixing RBAC misconfigurations when compared to the performance of the tool versus other security tools. As a result, future development should include constant improvement, thorough testing, and consistent tool update.

7 Conclusion and Future Work

The research was started with the specific goal of improving the Kube-Hunter tool's abilities to quickly spot and fix potential RBAC policy misconfigurations in Kubernetes settings. The Kube-Hunter tool was updated to include the painstakingly created RBAC Policy Misconfiguration Check algorithm in an effort to improve security. Nevertheless, despite a seemingly good implementation, unforeseen "forbidden" failures during the tool's use hampered the research. While first perceived as a problem, these failures quickly changed into a useful discovery that offered a clear view into the nuances of actual Kubernetes security. This finding fits with the major goals of the study, providing important details about the intricate configurations of Kubernetes RBAC policies and highlighting the need for advanced tools to manage them. Given the aforementioned problems, the research also showed that the improved Kube-Hunter tool showed promise in offering a more thorough analysis of RBAC items than certain other tools. Even with its limitations, this comparative performance highlights the benefits of the strategy and the tool's potential to address Kubernetes security concerns after the problems found are fixed.

The study has created a number of possibilities for further investigation. The unanticipated "forbidden" failures that arose throughout the algorithm's execution point to possible areas for improvement. In order to make the tool more capable of navigating through challenging real-world settings, we plan to improve the algorithm to include more

reliable error handling and exception-catching capabilities. Additionally, the algorithm might be expanded to accommodate a wider variety of Kubernetes security capabilities, such as Admission Controllers, Pod Security Policies, and Namespace Restrictions, among others. A more thorough investigation of these topics might provide the algorithm with a more sophisticated comprehension of Kubernetes environments, enabling it to anticipate and manage a larger range of security problems. A subsequent study effort might examine the tool's user experience in addition to these technical improvements. While the research concentrated on the technical capabilities of the instrument, examining its usability from the user's point of view could provide insightful information. Future studies might examine how users engage with the tool, the problems it run into, and the features they value the most. This might result in the creation of a user-friendly interface, improving the tool's usability and accessibility.

There is significant potential for commercialization in a larger setting. The need for strong and easy-to-use security tools is anticipated to increase as Kubernetes continues to gain widespread adoption. The Kube-Hunter tool, which uses the RBAC Policy Misconfiguration Check algorithm, has the potential to be a market-ready solution to the critical security issues in Kubernetes systems with further development and improvements.

References

Binnie, C. and McCune, R. (2021). Kubernetes vulnerabilities.

Bose, D. B., Rahman, A. and Shamim, S. I. (2021). 'under-reported' security defects in kubernetes manifests, *2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*, pp. 9–12.

Burns, B., Beda, J., Hightower, K. and Evenson, L. (2022). Kubernetes: up and running.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J. (2016). Borg, omega, and kubernetes: Lessons learned from three container-management systems over a decade, *Queue* **14**(1): 70–93.

Calderón-Gómez, H., Mendoza-Pittí, L., Vargas-Lombardo, M., Gómez-Pulido, J. M., Rodríguez-Puyol, D., Sención, G. and Polo-Luque, M.-L. (2021). Evaluating service-oriented and microservice architecture patterns to deploy ehealth applications in cloud computing environment, *Applied Sciences* **11**(10): 4350.

Carrión, C. (2022). Kubernetes scheduling: Taxonomy, ongoing issues and challenges, *ACM Computing Surveys* **55**(7): 1–37.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R. and Safina, L. (2017). Microservices: yesterday, today, and tomorrow, *Present and ulterior software engineering* pp. 195–216.

Ibryam, B. and Huß, R. (2022). *Kubernetes Patterns*, " O'Reilly Media, Inc.".

Kumara, I., Garriga, M., Romeu, A. U., Di Nucci, D., Palomba, F., Tamburri, D. A. and van den Heuvel, W.-J. (2021). The do's and don'ts of infrastructure code: A systematic gray literature review, *Information and Software Technology* **137**: 106593.

Martin, P. and Martin, P. (2021). Kubernetes resources, *Kubernetes: Preparing for the CKA and CKAD Certifications* pp. 19–22.

Medel, V., Rana, O., Bañares, J. Á. and Arronategui, U. (2016). Modelling performance & resource management in kubernetes, *Proceedings of the 9th International Conference on Utility and Cloud Computing*, pp. 257–262.

Mendonça, N. C., Jamshidi, P., Garlan, D. and Pahl, C. (2019). Developing self-adaptive microservice systems: Challenges and directions, *IEEE Software* **38**(2): 70–79.

Minna, F., Blaise, A., Rebecchi, F., Chandrasekaran, B. and Massacci, F. (2021). Understanding the security implications of kubernetes networking, *IEEE Security Privacy* **19**(5): 46–56.

Panagiotis, M. (2020). *Attack methods and defenses on Kubernetes*, PhD thesis, University of Piraeus (Greece).

Rahman, A., Shamim, S. I., Bose, D. B. and Pandita, R. (2023). Security misconfigurations in open source kubernetes manifests: An empirical study, *ACM Transactions on Software Engineering and Methodology* **32**(4): 1–36.

Revuelta Martinez, Á. (2023). Study of security issues in kubernetes (k8s) architectures; tradeoffs and opportunities.

Roeckle, H., Schimpf, G. and Weidinger, R. (2000). Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization, *Proceedings of the fifth ACM workshop on Role-based access control*, pp. 103–110.

Rossi, F., Cardellini, V., Presti, F. L. and Nardelli, M. (2020). Geo-distributed efficient deployment of containers with kubernetes, *Computer Communications* **159**: 161–174.

Sandhu, R. S. (1998). Role-based access control, *Advances in computers*, Vol. 46, Elsevier, pp. 237–286.

Sayfan, G. (2017). *Mastering kubernetes*, Packt Publishing Ltd.

Shamim, M. S. I., Bhuiyan, F. A. and Rahman, A. (2020). Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices, *2020 IEEE Secure Development (SecDev)* pp. 58–64.

Vayghan, L. A., Saied, M. A., Toeroe, M. and Khendek, F. (2018). Deploying microservice based applications with kubernetes: Experiments and lessons learned, *2018 IEEE 11th international conference on cloud computing (CLOUD)*, IEEE, pp. 970–973.

Viktorsson, W., Klein, C. and Tordsson, J. (2020). Security-performance trade-offs of kubernetes container runtimes, *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, IEEE, pp. 1–4.

Xu, J., Chen, P., Yang, L., Meng, F. and Wang, P. (2017). Logdc: Problem diagnosis for declartively-deployed cloud applications with log, *2017 IEEE 14th International Conference on e-Business Engineering (ICEBE)*, IEEE, pp. 282–287.