KubeDeceive: Unveiling Deceptive Approaches to Protect Kubernetes Clusters

Abdelrahman Aly[*]
Ain Shams University, Cairo. Egypt.
Mahmoud Fayez[†]
Ain Shams University, Cairo. Egypt.
Mirvat Al-Qutt [‡]
Ain Shams University, Cairo. Egypt.
Ahmed M. Hamad [§]
Ain Shams University, Cairo. Egypt.

Abstract

The widespread adoption of containerization platforms, such as Kubernetes, has revolutionized application deployment and management. However, this evolution brings with it sophisticated security challenges. Deception-based strategies provide a powerful approach to address these challenges by misleading attackers with simulated resources. This paper presents KubeDeceive, a cutting-edge security framework specifically designed to enhance the security posture of Kubernetes environments through tailored deception techniques. KubeDeceive operates as a middleware, intercepting requests to the Kubernetes API server and guiding malicious users towards decoy components. Its effectiveness was evaluated in a Capture the Flag (CTF) competition designed to simulate real-world attacks. KubeDeceive proved highly effective, achieving a 100% success rate in preventing any participant from deploying a master node pod—the main target and final flag of the challenge—and trapping 89% of participants in deception decoys. Additionally, participants expended an average of 160 minutes in their unsuccessful attempts during dynamic scenarios, highlighting KubeDeceive's ability to prolong attacker engagement and decisively thwart their objectives.

Key Words: Cyber Deception, Kubernetes Security, Admission Controller, Malicious Actors, Threat Detection, Decoy Assets, Deceptive Environment, Cloud Security, Container Orchestration, Deceptive Tactics.

[*]Faculty of Computer and Information Science. Email: abdlrhmn.ali@cis.asu.edu.eg.
[†]Faculty of Computer and Information Science. Email: mahmoud.fayez@cis.asu.edu.eg.
[‡]Faculty of Computer and Information Science. Email: mmalqutt@cis.asu.edu.eg.
[§]Faculty of Computer and Information Science. Email: ahmed.hamad@cis.asu.edu.eg.

1 Introduction

Deception is pivotal in modern cybersecurity, addressing evolving threats and limitations of traditional defenses. It enables early threat detection by deploying traps and decoys that alert security teams to malicious activities. Deception tools provide deep insights into attackers' tactics, enhancing defensive strategies and adapting to dynamic threat landscapes effectively. This knowledge, as emphasized in the CSCI Conference [1], greatly enhances the comprehension and adaption to the evolving threat landscape. In this context, the rise of cloud computing and container orchestration platforms like Kubernetes represents a new frontier for applying deception strategies. The dynamic and distributed nature of cloud environments renders traditional security measures less effective. Adapting deception techniques to the cloud and Kubernetes offers a flexible defense strategy. By deploying deceptive assets within these environments, organizations can proactively detect and respond to potential threats, thereby protecting sensitive data and critical workloads.
One of the significant security risks highlighted in the OWASP Kubernetes Top 10 [2] is the potential for insecure Kubernetes configurations. Kubernetes' highly configurable nature can lead to misconfigurations that create security gaps, allowing attackers to access resources, compromise data, or disrupt services. Another critical vulnerability is container escapes, where vulnerabilities in container runtimes or the host OS enable attackers to break out from containers, gaining unauthorized access to the host system and potentially compromising the entire Kubernetes cluster. In the evolving landscape of Kubernetes security, several tools have become prominent for their ability to safeguard these environments against various threats. Tools like 'Clair[3],' 'Checkov[4],' 'Kubeaudit[5],' and the 'Open Policy Agent (OPA)[6]'. Each plays a unique role in fortifying Kubernetes deployments and will be illustrated in more detail in the related work section. Despite the existence of these robust tools, there remains a gap in the realm of deception-based security specifically tailored

for Kubernetes environments. The current deception solutions related to Kubernetes, such as HoneyKube[7], do not directly address Kubernetes-specific attacks but leverage Kubernetes to deploy deception-powered honeypots. This approach aims to prevent attacks on other systems—such as web servers, network devices, industrial systems, and IoT devices—by misleading attackers through Kubernetes-driven deception strategies.

In the broader realm of cybersecurity, deception frameworks play a crucial role across various layers of the deception stack, including system, network, endpoint, and data layers. At the network layer, frameworks like MTDCD (MTD Enhanced Cyber Deception Defense System) [8] and DESIR (Decoy-enhanced seamless IP randomization) [9] construct virtual network topologies to delay attackers by creating deceptive environments. However, their effectiveness in dynamic container networks like Kubernetes is limited due to the platform's transient object nature. Moving to the endpoint layer, tools such as Moonraker [10] and CHAOS (Chaos Tower Obfuscation System) [11], effective in traditional IT settings but less so in containerized environments like Kubernetes. In the software layer, techniques like those in SODA (A System for Cyber Deception Orchestration and Automation) [12] manipulate API calls and create deceptive documents to counter software-based threats. However, applying these strategies directly to Kubernetes is challenging due to its unique API interactions and resource management. At the data layer, web-based deception techniques [13] manipulate web content and session management, tailored for traditional web environments, which may not seamlessly integrate with Kubernetes' distributed data management models. Further exploration of these deception frameworks and their specific implementations will be detailed in the related work section.

To address these challenges, KubeDeceive has emerged as a robust deception framework specifically tailored for Kubernetes environments. It focuses on mitigating critical vulnerabilities highlighted in the Kubernetes OWASP Top 10 by intercepting malicious requests, mimicking genuine operations, and deploying sophisticated security measures. KubeDeceive achieves this by directing malicious workloads to decoy nodes, carefully adjusting pod configurations, and leveraging audit logs to thoroughly document activities within the cluster. This approach enables detailed analysis of malicious behavior, facilitates tracking of attacker movements, and supports continuous refinement of deception strategies to enhance Kubernetes security posture.

This paper is organized as follows: Section 1 provides an introduction to Kubernetes and discusses its security implications. Section 2 reviews literature on deception techniques in security contexts. Section 3 introduces the framework and its deployment approach. Section 4 presents a detailed CTF case study to assess efficacy. Section 5 concludes with reflections and future research directions.

2 Related Work

2.1 Traditional Security Solutions for Kubernetes

Kubernetes security has become increasingly complex and multi-faceted, with a variety of tools and techniques emerging to address its unique challenges. Prominent among these is 'Clair,' a container vulnerability scanning tool designed to identify security weaknesses in container images. 'Checkov' offers another layer of defense, auditing Kubernetes configurations to detect potential misconfigurations and ensure compliance with established best practices. 'Kubeaudit' plays a crucial role in auditing Kubernetes clusters for common security issues, providing actionable recommendations to enhance security. The OPA is integral for policy-based control, enabling fine-grained governance over Kubernetes clusters, and ensuring that activities and resources comply with corporate and regulatory policies. Container security platforms like 'Aqua Security'[14] provides comprehensive security solutions that encompass scanning container images for vulnerabilities, enforcing strict runtime security, and ensuring compliance. Network policies and segmentation tools, such as 'Calico,[15]' further bolster security by controlling pod-to-pod communication and limiting access based on need-to-know principles.

2.2 Deception Solutions in Cybersecurity

While various Kubernetes security tools have emerged, they lack deception tactics. Deception in cybersecurity adds a potent layer of defense, facilitating early threat detection and strategically luring and delaying attackers. By creating decoy assets, deception tools waste attackers' time and resources, allowing security teams to intervene and gather valuable information on attacker tactics. To categorize the proposed systems for deception, we classify them into four main categories based on the general deception stack, as detailed in [16].

One notable example of such categorization is Moonraker, introduced by T. B. Shade et al., which serves as a system-based deception framework. Moonraker is specifically designed to mislead attackers within the system environment and protect critical assets. The study further evaluated the effectiveness of deceptive responses by comparing success rates between scenarios with and without deception, highlighting the practical benefits of employing deception strategies within the broader stack. The study evaluated the effectiveness of deceptive responses by comparing success rates between conditions with and without deception. Results demonstrated that deceptive responses significantly reduced successful attacks, highlighting their effectiveness in thwarting attacker objectives. Conversely, the control condition showed higher success rates in executing tactics, underscoring the disruptive impact of deceptive techniques on attackers. Moonraker's challenge lies in creating convincing system-level deceptions. Limiting participants' command usage during the study may influence attack behavior and impact the framework's real-world applicability.

Building on this, Gao, Wang et.al developed MTDCD (MTD Enhanced Cyber Deception Defense System) as an enhanced Network-Based cyber deception defense mechanism, focusing on using virtual network topologies (VNTs) to delay attackers in discovering vulnerable hosts. Their study demonstrated that deploying VNTs extended the time for attackers to discover vulnerable hosts by an average of seven times and increased the time to attack a vulnerable host by an average of eight times. Additionally, the study evaluated the impact of VNTs on network overhead, revealing increased network latency and flow table reinstallation frequency. These findings highlight the effectiveness of network-based deception in deterring attackers and introducing delay mechanisms. A noted limitation was the impact of VNTs on network performance, including increased latency and flow table reinstallation frequency, which could potentially affect overall network efficiency.

Similarly, Sajid, Wei, Abdeen et.al developed SODA (System for Cyber Deception Orchestration and Automation), a malware-based deception system aimed at thwarting malware attacks through deceptive tactics. SODA analyzes malware to extract Malicious Subgraphs (MSGs) representing API call sequences mapped to the MITRE ATT&CK framework. Evaluation across RATs, InfoStealers, Ransomware, and Spyware achieved 95% accuracy, with 224 out of 237 deception attempts successfully misleading malware. Successful deceptions included manipulating Command and Control (C2) interactions and tricking ransomware into generating ransom notes sans encryption. While effective against various malware types, SODA's focus on software-layer interactions may not fully address broader security needs in complex environments like Kubernetes, encompassing network, endpoint, and data layers.

In this context, Han, Kheir, & Balzarotti explored web-based deception techniques to thwart adversaries by manipulating web content, session management, and user interactions. Their experiments, including a CMS application and a CTF exercise [17], highlighted the effectiveness of deception in detecting web attacks. In the CMS experiment, honeytrap resources in the robots.txt file triggered alerts, while hidden deception elements remained undiscovered. During the CTF, participants encountered deception traps more frequently than real vulnerabilities, showcasing how such techniques can misdirect attackers. While effective in triggering alerts, web-based deception did not consistently uncover real vulnerabilities, underscoring the need for more comprehensive security strategies in diverse attack scenarios.

2.3 Bridging the Gap: The Unique Value Proposition of KubeDeceive

In contrast to existing frameworks, KubeDeceive is tailored specifically for Kubernetes, addressing its unique multi-layered security challenges. While SODA focuses on software-layer deception, KubeDeceive operates across network, endpoint, and data layers. It expands beyond Moonraker's system-level misleading by integrating deception directly into Kubernetes orchestration and containerization. Addressing MTDCD's limitations, KubeDeceive provides network deception with minimal overhead and extends web-based deception techniques to suit Kubernetes' dynamic nature. KubeDeceive bridges gaps between current deception frameworks and traditional Kubernetes tools by offering a holistic approach to deception. Its innovative strategy includes dynamic pod reconfiguration, manipulation of network traffic, and deceptive responses to API calls, ensuring robust defense against various attack vectors within Kubernetes clusters. Scientifically, KubeDeceive pioneers the use of Kubernetes-native mechanisms for deception, leveraging labels, annotations, and controllers to create indistinguishable deceptive artifacts integrated into Kubernetes' control plane. This dynamic environment adapts to cluster changes, maintaining effective security measures as new workloads are deployed. By studying how attackers engage with deceptive elements like fake pods, KubeDeceive enhances threat detection and mitigation, contributing valuable insights to cybersecurity practices in cloud-native environments.

3 The Proposed Deception Framework

The main goal of this research is to create a robust deception framework for Kubernetes environments and develop effective defense mechanisms against pod breakout attacks. To achieve this, we undertake a comprehensive problem identification and analysis phase, where we thoroughly investigate the security challenges and risks associated with pod breakout scenarios.

3.1 Problem Identification and Analysis

In this section, we identify and analyze critical security issues in Kubernetes environments, focusing on pod breakout scenarios and aligning them with OWASP Top 10 and MITRE (TTPs)[18]. The paper highlights two main flaws compromising node security:

(a) OWASP Top 10: K01 Insecure Workload Configurations (MITRE TTP: T1611-T1068-T1610)[19]: This flaw is related to insecure configurations of workloads within the Kubernetes environment. As part of the exploration of insecure workload configurations in the Kubernetes environment, we will closely examine the attributes of ”Privileged Containers,” ”HostPID,” ”HostPath Volumes,” and ”HostIPC.” These attributes have the potential to break the principle of least privilege and introduce security vulnerabilities if they are misconfigured or misused.

(b) OWASP Top 10: K05 - Inadequate Logging and Monitoring (MITRE TTP: T1070)[20]: This vulnerability arises from the absence of proper logging and monitoring mechanisms in the Kubernetes environment . Attackers can exploit this weakness to carry out stealthy attacks and evade detection. To address this critical concern, the deception framework incorporates robust logging and monitoring features that capture and analyze crucial

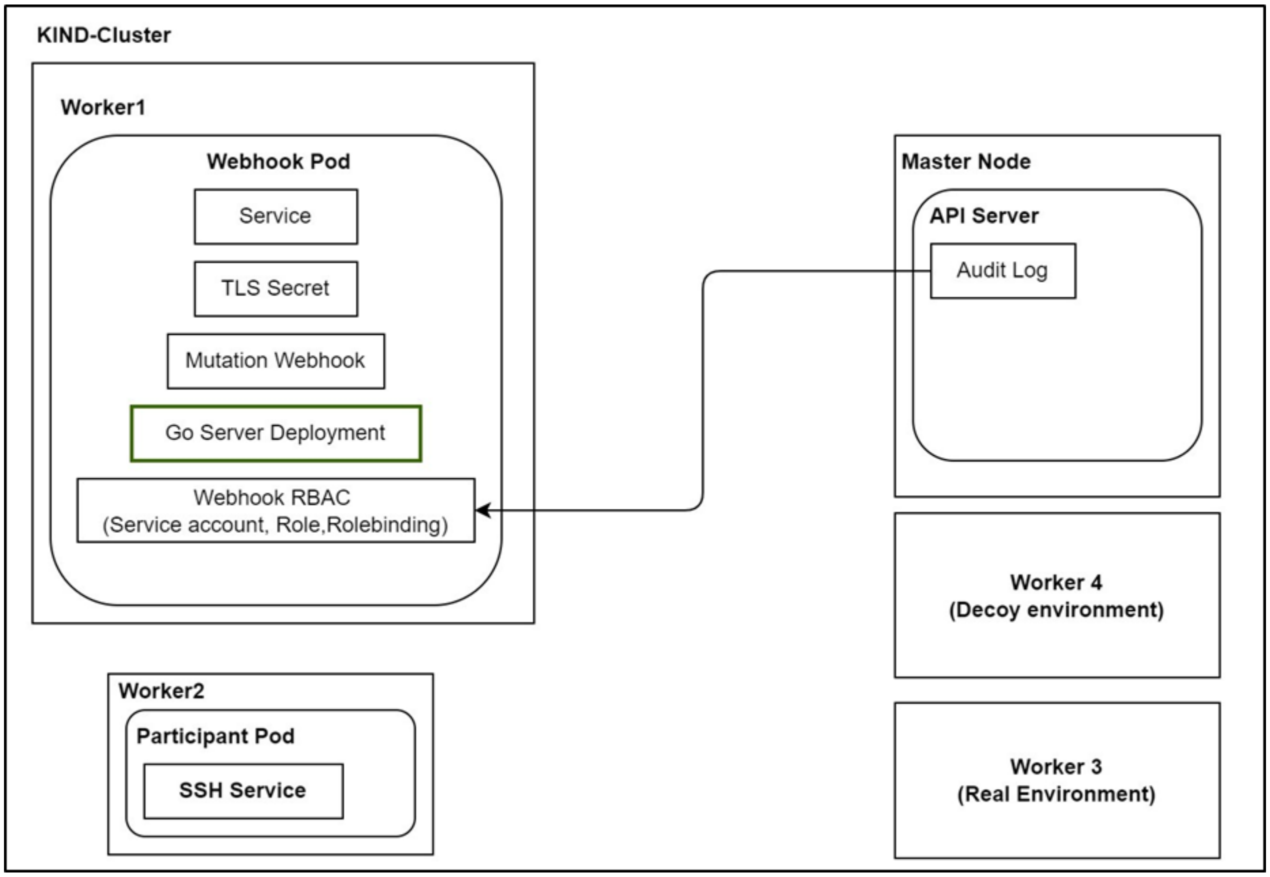


Figure 1: System architecture of KubeDeceive,
integrating Kubernetes components to deceive and monitor attackers.

activities within the Kubernetes cluster. By generating comprehensive audit logs for all actions, the framework enhances visibility and provides valuable insights into potential security incidents.

3.2 Deception System Architecture

The described deception framework integrates multiple key components to establish a structured environment aimed at deceiving potential attackers effectively. Illustrated in Figure 1, the system architecture includes elements like the Cluster, Pod with Mutation Webhook, Participant/Attacker Pod, Audit Log, and Overlay script. Together, these components intercept and mislead attackers within the KIND [21] cluster, enhancing overall system security. The choice of KIND (Kubernetes in Docker) over Minikube [22] was driven by scalability and compatibility needs. While Minikube is suitable for local Kubernetes development in single-node configurations, KIND's multi-node capability better suited the project's requirements, especially in VMware environments, ensuring seamless integration, portability, and performance. In the following subsections, we will explore each component, providing a detailed description of its purpose, functionalities, and interactions within the framework.

3.2.1 Kind Cluster

The cluster serves as the foundation of the system, consisting of a minimum of three nodes. It includes a master node responsible for managing th— e cluster, a worker1 node that acts as a normal environment serving the business, and a worker2 node that acts as a honeypod environment. The cluster provides the necessary infrastructure for running various components, ensuring high availability and scalability.

3.2.2 Webhook Pod

The Webhook Pod serves as the core component responsible for hosting the mutation webhook logic written in GO based on the clientgo library [23].Client-go is a Go client library by Kubernetes for programmatically interacting with Kubernetes clusters and APIs, enabling developers to build custom controllers, operators, and applications. The combination of the following components within the Webhook Pod allows for effective interception and modification of requests, ensuring controlled access, secure communication, and seamless integration within the Kubernetes cluster.
RBAC (Role-Based Access Control) Security Mechanism: Ensures controlled access by defining privileges and

permissions for the Webhook Pod within the Kubernetes cluster. It governs access policies to mitigate security risks like unauthorized access or misuse.
Webhook Object: Defines specific requests and actions that the webhook intercepts and modifies. Administrators customize its behavior based on criteria, specifying triggers for request interception.on specific criteria. This includes defining the triggers for intercepting requests.
Service Object: Facilitates communication between the Webhook Pod and other cluster components. It provides a stable endpoint for reliable routing of requests, ensuring seamless integration and efficient request processing.
Secret Object: Manages encryption and decryption of traffic between the webhook and other components using TLS certificates. Ensures data confidentiality and integrity, safeguarding against unauthorized access or tampering.
Deployment Object: Manages execution and scaling of the Webhook Pod's GO code. It handles pod creation, updates, and scaling based on defined replica counts, monitoring pod health to maintain operational state.

3.2.3 GO Webserver

This server, developed in Go, plays a vital role in managing intercepted requests. Listing 1 elaborates on the pseudo-code for the GO server logic. The server initiates by parsing intercepted requests and deserializing them into REST request objects (Lines 1-5). It then comprehensively examines these objects, including their configurations, privileges, and resource usage (Lines 8-10). This analysis forms the basis for determining appropriate deception actions, such as rejecting requests, introducing delays, or modifying actions (Lines 12-19). To ensure legitimate user interactions are not disrupted, the server incorporates mechanisms to whitelist user IPs, accounts, and custom keys (Lines 22-27). This approach balances robust security measures with the need to maintain operational integrity.

3.2.4 Participant/Attacker Pod

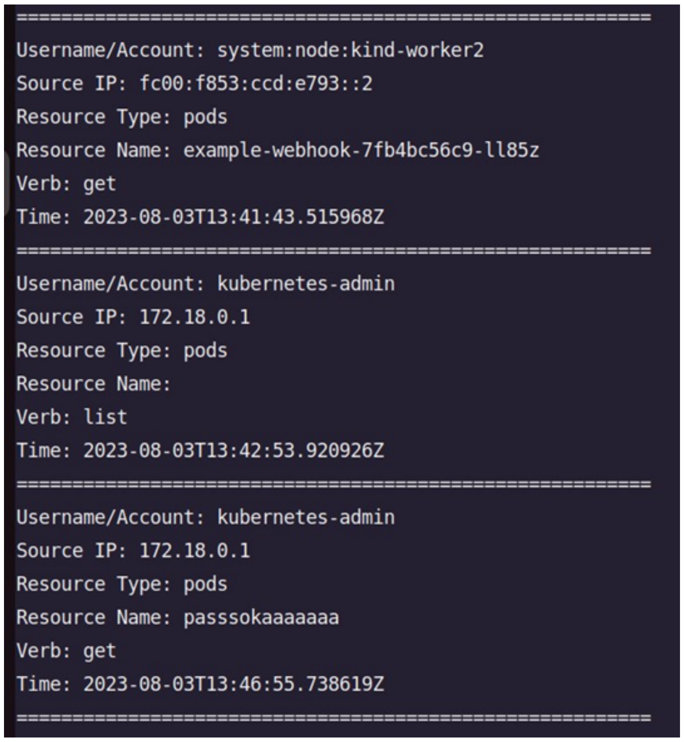
The participant pod is a critical component in the system, acting as a simulated adversary. It hosts an attacker who, once inside the cluster, mimics harmful actions. These actions can result from system vulnerabilities or, in some cases, an insider with unauthorized access. This setup allows participants to interact with the Kubernetes API server, potentially executing harmful attacks. Participants use SSH [24] connection which serves as the entry point for such attacks. The connection is closely monitored and controlled through the GO logic on a web server, with admission controllers ensuring thorough scrutiny and security management of potentially harmful activities within the cluster.

Listing 1: Pseudocode for Webhook Pod logic in Go

```
1
2  parseAndAnalyzeRequests () {
3    for each intercepted request {
4      parsedRequest := deserializeRequest(request)
5      goFormatObject := bindToGOFormat(parsedRequest)
6      analyzeObject(goFormatObject)
7    }
8  }
9  analyzeObject(goObject) {
10     deceptionAction:=
           determineDeceptionAction(goObject)
11     performDeceptionAction (deceptionAction, goObject)
12 }
13 performDeceptionAction (deceptionAction, goObject) {
14   switch deceptionAction {
15   case "reject":
16     rejectRequest(goObject)
17   case "delay":
18     delayRequest(goObject)
19   case "change":
20     changeAction(goObject)
21   }
22 }
23 determineDeceptionAction(goObject) {
24   if isWhitelistedUser(goObject) {
25     return "none" // No deception action
26   } else {
27     deceptionAction := randomlyChooseAction()
28     return deceptionAction
29   }
30 }
31 // Main execution
32 parseAndAnalyzeRequests()
```

3.2.5 Audit Log

KubeDeceive capitalizes on Kubernetes' native audit log functionality [25] to monitor cluster activities. Audit logs play a crucial role in Kubernetes' security architecture by providing a chronological record of system-affecting events. In our Kubernetes cluster configuration, we specify the audit log path and policy file within the API server arguments. This setup logs all relevant API calls, creating a detailed trail for security analysis. The main webhook includes an admission control endpoint to parse and analyze audit logs in real-time, enabling proactive security measures and continuous monitoring.
For example, Figure 2 illustrates a sample audit log entry captured by KubeDeceive. This entry denotes an attempt to access a pod resource. The suspicious nature of the pod name, resembling a potential decoy, triggers KubeDeceive's analysis protocol. The admission control hook processes this request, and by correlating it with the defined security policies and the context of the user's activity, it determines whether the action is legitimate or potentially malicious.



```
========================================================
Username/Account: system:node:kind-worker2
Source IP: fc00:f853:ccd:e793::2
Resource Type: pods
Resource Name: example-webhook-7fb4bc56c9-ll85z
Verb: get
Time: 2023-08-03T13:41:43.515968Z
========================================================
Username/Account: kubernetes-admin
Source IP: 172.18.0.1
Resource Type: pods
Resource Name:
Verb: list
Time: 2023-08-03T13:42:53.920926Z
========================================================
Username/Account: kubernetes-admin
Source IP: 172.18.0.1
Resource Type: pods
Resource Name: passsokaaaaaaa
Verb: get
Time: 2023-08-03T13:46:55.738619Z
========================================================
```

Figure 2: Sample audit log showing intercepted API request analysis by KubeDeceive

3.2.6 Overlay Script

The overlay script is a critical component of the system, developed as a Python script to automate and streamline the environment creation process. Its primary purpose is to empower users by enabling them to specify essential options and configurations during setup. This approach significantly improves user experience by eliminating the need for manual intervention and complex configurations. Furthermore, the overlay script is designed for high flexibility, capable of seamlessly adapting to future upgrades or framework changes.

3.3 Deception Framework Deployment Strategy

The deployment strategy for our deception framework in Kubernetes is designed to intercept and mitigate malicious actions effectively. Illustrated in Figure 3, the process begins with requests from users and potential attackers passing through a webhook pod. This pod intercepts requests aimed at the API server and applies deception actions based on predefined logic implemented in GO. If needed, deception measures are executed before forwarding modified requests to the API server. In the deployment process of KubeDeceive, several critical steps are undertaken to ensure its effective integration and operation within a Kubernetes environment. The process begins with the configuration of essential security components. Following this, a Python script is executed to set up the cluster and admission control functionalities.

(a) **TLS Certificate and Secret Configuration:** The initial step involves configuring the Transport Layer Security (TLS) certificate and secrets. These are crucial for secure communication between users and the Kubernetes API server. This setup ensures that all interactions, especially those related to the admission control webhook, are encrypted and protected from unauthorized access or tampering.

(b) **Execution of Overlay Python Script:** Subsequently, an overlay Python script is executed. It runs a predefined cluster configuration YAML file, which outlines the number of nodes required for the cluster and the configurations necessary for audit logging. This script ensures that the Kubernetes cluster is set up with the appropriate settings to support the advanced monitoring and logging capabilities needed for effective deception.

- **Application of RBAC and Webhook Configurations**. As part of the script execution, several key YAML files are applied to the cluster:
  - **Rbac.yaml:** This file defines the necessary permissions for the webhook's pod, ensuring that it has the appropriate access rights within the Kubernetes environment.
  - **Webhook.yaml:** This configuration file sets up the Mutation Admission Webhook object. The webhook acts as a gatekeeper, modifying or rejecting requests to the API server based on predefined rules and logic.
- **Whitelisting Mechanism:** An important feature of this deployment is the ability to whitelist certain usernames and IP addresses. Users and systems with these credentials are allowed to bypass the restrictions imposed by the admission control. This mechanism is critical for maintaining normal operation within the environment while selectively targeting and stopping only malicious actors.
- **Building the Deception Deployment:** Finally, the script proceeds to build the deployment that encapsulates the main logic of the admission controller and the various deception techniques employed by KubeDeceive. This deployment includes the creation of a Kubernetes object that integrates closely with the Kubernetes API server and uses a container image containing the server logic written in Go.

The setup is designed to be dynamic, allowing for real-time adjustments and updates to the deception tactics based on

3.4 Challenges and Limitations

As we developed a deception framework for Kubernetes, we faced challenges in intercepting traffic, prompting a thorough exploration of various approaches. This subsection details the encountered challenges and diverse methods considered for

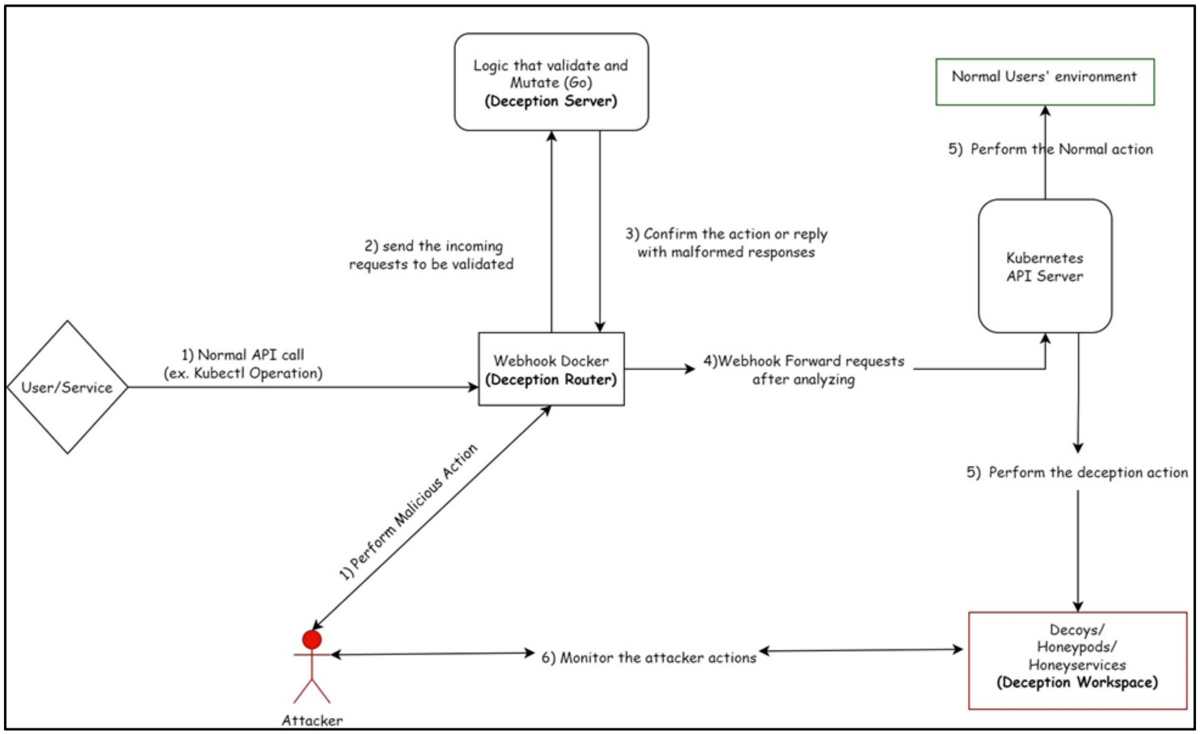

Logic that validate and Mutate (Go) (Deception Server)
Normal Users' environment
5) Perform the Normal action
2) send the incoming requests to be validated
3) Confirm the action or reply with malformed responses
Kubernetes API Server
User/Service
1) Normal API call (ex. Kubectl Operation)
Webhook Docker (Deception Router)
4)Webhook Forward requests after analyzing
1) Perform Malicious Action
5) Perform the deception action
6) Monitor the attacker actions
Decoys/ Honeypods/ Honeyservices (Deception Workspace)
Attacker

Figure 3: Data flow diagram showing request interception and deception by the webhook

effective traffic interception within the Kubernetes ecosystem. The analysis provides insights into the complexities involved, shaping the foundation of our deception framework.

(a) **Kubernetes Code Modification:** We initially considered modifying the Kubernetes codebase to incorporate interception logic using source graph extension and Visual Studio tools. However, the complexity of Kubernetes internals and the risk of disrupting the environment deterred us from this approach. Integrating custom solutions into the Kubernetes codebase required a thorough understanding of the system's intricacies and could introduce unintended complications.

(b) **GRPC Calls for Traffic Redirection[26]:** We explored using GRPC calls for redirecting traffic to address interception requirements. The goal was to find a seamless way to control the redirection process. However, we faced challenges in finding libraries or APIs for calls from outside the API server. Additionally, using a Python plugin led to errors and compatibility issues, hindering progress.

(c) **Mitmproxy Implementation[27]:** We considered mitmproxy for intercepting and redirecting requests between components and the Kubernetes API server. Mitmproxy offered features for intercepting and manipulating network communications, making it a promising solution. However, configuring system components to work with an external proxy, particularly with SSL settings, posed significant challenges. Managing SSL certificates and establishing trust required substantial effort to ensure secure and functional communication between components and the proxy.

While KubeDeceive effectively addresses many threats to Kubernetes clusters, it does have some key limitations:

(a) **Limited Scope of Attack Detection:** KubeDeceive is designed to handle specific types of attacks, such as API request-based exploits and common Kubernetes misconfigurations. However, it is not yet equipped to detect obfuscated or behavior-based attacks. Addressing such threats would require integrating AI-powered models [28] capable of identifying subtle anomalies and patterns indicative of advanced attacks.

(b) **Dependence on Decoy Environment Setup:** The effectiveness of the framework heavily relies on how well the decoy environment aligns with the organization's specific business use case. Deploying a decoy environment tailored to the unique configurations and operations of each business can be resource-intensive and requires a deep understanding of the business's Kubernetes setup. These limitations underscore the need for future work to expand the framework's capabilities, particularly through AI-powered threat detection and automation of decoy environment setup to align with diverse business contexts.

4 Evaluation

To assess the effectiveness of the deception framework, a CTF competition [29] was organized, involving 20 participants with varying experience levels in penetration testing, ranging from 2 to 5 years of experience. The objective of the competition was for participants to successfully deploy a malicious pod on the master node of a simulated Kubernetes cluster. Participants were presented with a scenario where they could exploit exposed and vulnerable services within the cluster to gain unauthorized access. Once inside the cluster, participants had direct interaction with the Kubernetes API server, enabling them to execute potentially harmful commands.

This realistic challenge aimed to assess participants' skills and creativity in exploiting Kubernetes vulnerabilities and orchestrating damaging actions. To facilitate the competition, we implemented a streamlined approach for participants. Each participant received access to a specific pod hosted on a worker node via a secure SSH (Secure Shell) connection.

This SSH connection allowed remote access and control of the designated pod within the Kubernetes cluster. Additionally, a service account was set up specifically for this pod, configured with a predefined role that granted specific permissions and capabilities within the cluster environment. In our CTF competition scenario, participants were tasked with deploying a malicious pod on the Kubernetes Master Node, navigating through simulated vulnerabilities and controls. They utilized a service account role allowing pod creation within the cluster. Figure 4 outlines their procedural flow: accessing a controlled pod via SSH credentials, configuring environment variables, and employing kubectl commands, all managed under KubeDeceive's protective controls. Despite enforced rules to prevent unauthorized actions, Figure 5 revealed some participants bypassing restrictions using the 'exec' verb on master pods. In response, we refined controls, restricting actions to specific service accounts tailored for pod creation and predefined commands. These measures effectively secured the CTF environment, highlighting the necessity of dynamic security protocols in Kubernetes deployments.

This setup demonstrates KubeDeceive's architecture, integrating comprehensive security measures with Kubernetes' native controls to safeguard against evolving threats and unauthorized accesses.

Approximately 7 participants immediately pursued the token path, encountering strategically placed fake tokens that tested their decision-making skills. Meanwhile, others explored locally and fell into traps set by decoys representing fake tokens in various locations. After acquiring the token, participants proceeded to create the kubeconfig file necessary for pod creation. Participants crafted various malicious YAML files, and their interactions were closely monitored and analyzed using audit logs to evaluate their strategies. Interestingly, only two participants thoroughly examined the created pod YAMLs, using 'describe' actions to identify special notations and differences.

As demonstrated in the evaluation results, Table 1 indicates varying trap counts for different actions taken by participants to achieve their goals. Initially, three static secrets were distributed

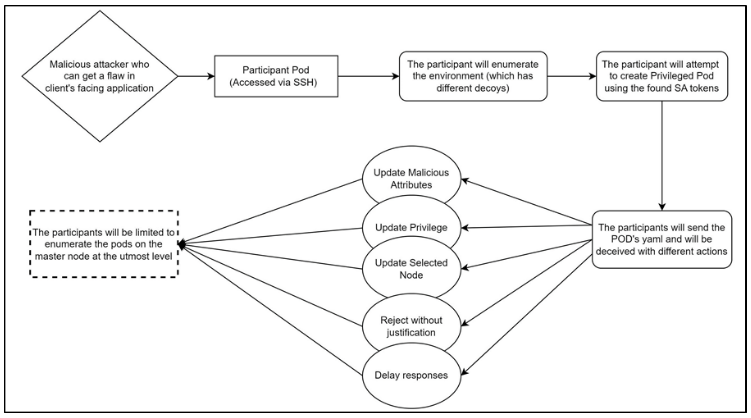

Malicious attacker who can get a flaw in client's facing application
Participant Pod (Accessed via SSH)
The participant will enumerate the environment (which has different decoys)
The participant will attempt to create Privileged Pod using the found SA tokens
The participants will send the POD's yaml and will be deceived with different actions
Update Malicious Attributes
Update Privilege
Update Selected Node
Reject without justification
Delay responses
The participants will be limited to enumerate the pods on the master node at the utmost level

Figure 4: Attack Path Taken by Participants with Deception Solutions in Place

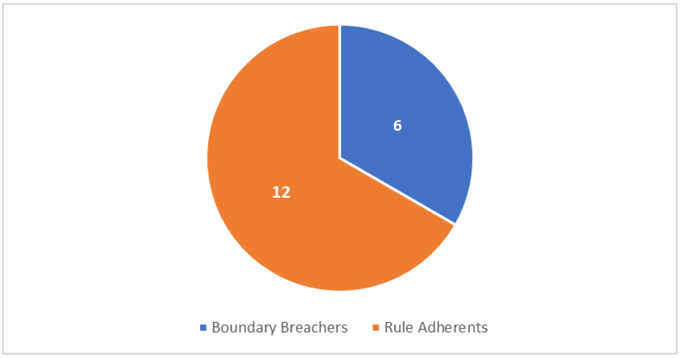


Figure 5: Quantitative Analysis of Participant Compliance in Kubernetes CTF Challenge

within their Pod, serving as a fixed layer of deception. This initial layer aimed to test participants' ability to navigate through deceptive elements strategically placed within their environment.

Another flag was found as a label in the pod specifications, serving as a key to authorize pod creation. In contrast, other participants relied on hints to locate the secret label needed to complete the challenges successfully. Upon successful communication with the API server and retrieval of Pod information, participants were required to extract labels and annotations from the created pods. These labels and annotations, which were part of the whitelisting techniques mentioned earlier, were then inserted into their malicious YAML files, which would later be used to create the malicious pod. Additionally, another IP whitelisting technique was applied, which prevented participants from creating a pod on the master node. While the secret label allowed them to create pods, it alone was insufficient to complete the target. This combination of techniques led participants to believe they had achieved progress, only to be ultimately deceived and thwarted in their attempts. This step was critical in assessing the effectiveness of our dynamic deception techniques, as it directly engaged participants in a more complex interaction with the deceptive environment, challenging them to distinguish between legitimate and deceptive information.

Table 2 presents the number of successful participants as determined by the flags collected at each step, providing a quantitative measure of the deception framework's effectiveness. The progression from identifying the correct secret token to obtaining the appropriate whitelisting label and attempting the creation of a privileged pod illustrates a sequential engagement with the deployed deceptive mechanisms.

Notably, the absence of participants successfully creating a pod on the master node highlights the robustness of the deception mechanisms in thwarting unauthorized access to critical resources. This layered approach to deception, starting with static secrets and escalating to more complex deceptions involving pod label manipulation, exemplifies the framework's capability to adaptively challenge and mislead potential attackers. The dynamic nature of these deceptions not only delayed participants but also significantly reduced the likelihood of successful attacks on critical cluster resources.

Trap Name	Trap Count	Detected Participants	Avg N. of Attempts
Fake Secrets (Different Locations)	3	11	17
Update Privilege (Privileged/runasuser)	2	8	10
Update Malicious Attributes (HostIPC/HostNetwork/HostPID/HostPath)	4	7	7
Update Selected Node (MasterNode)	1	10	13

Table 1: Analysis of Traps and Participants Actions

Flags	N. of Participants
Retrieve the correct Token	12
Get the Secret Label	8
Create a privileged pod	6
Create a pod on the master node	0

Table 2: Summary of Flags and Participant Engagement

Additionally, we introduced a 'static deception' scenario, where the absence of active deception solutions (mechanisms implemented by KubeDeceive) required participants to navigate only static obstacles. These included crafting the kubeconfig file, identifying correct labels and annotations from misleading pod specifications, and dealing with fake secrets strategically placed within the participant's pod.

Figure 6 compares the time taken by participants to perform malicious actions across three distinct environments: the full deception environment, the static deception environment, and the baseline environment. The full deception environment, powered by KubeDeceive, dynamically intercepts and manipulates API requests, deploys misleading pod configurations, and uses adaptive deceptive tactics to actively delay and mislead attackers. In contrast, the static deception environment features pre-configured obstacles such as fake secrets, misleading labels, and static pod specifications, providing a fixed and non-interactive deceptive layer. Lastly, the baseline environment serves as a control, offering no deception measures, allowing attackers unrestricted access to exploit vulnerabilities directly.

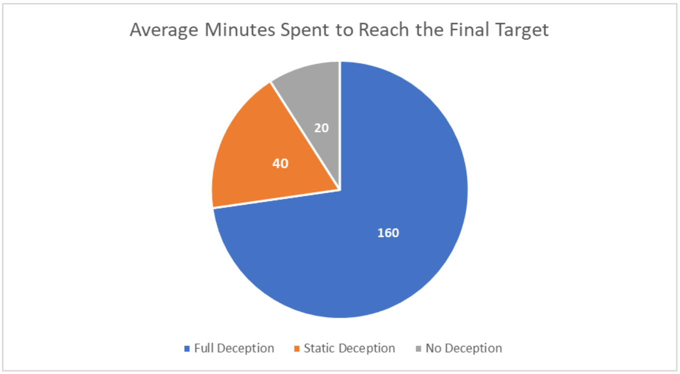


Figure 6: Time Consumed to Reach the Target

As a result, no participants were able to bypass the deception mechanisms or identify deceptive behaviors within the allocated time frame of 160 minutes. KubeDeceive's deployment within a Kubernetes environment proved highly effective, as shown in the evaluation results. Approximately 89% of participants fell for at least one trap, with the 'Fake Secrets' and 'Update Selected Node' traps being the most effective. Additionally, KubeDeceive was 100% successful in preventing participants from creating a pod on the master node, the ultimate challenge of the simulation, demonstrating its robust capability to delay and halt potential threats within the Kubernetes ecosystem.

5 Conclusion and Future Work

In conclusion, this paper introduces KubeDeceive, a pioneering cybersecurity framework designed specifically for Kubernetes environments. KubeDeceive stands out by integrating innovative deception strategies across multiple layers of Kubernetes, effectively mitigating inherent platform vulnerabilities. Its effectiveness is evident in its exceptional ability to safeguard the master node, achieving a 100% success rate in thwarting unauthorized pod creations. This underscores its capacity to strengthen critical components of Kubernetes against sophisticated attacks, such as insecure workload configurations, inadequate logging and monitoring, and privilege escalation, which pose significant threats to Kubernetes environments as discussed earlier. Compared to existing solutions, KubeDeceive offers a comprehensive and adaptable security posture capable of dynamically responding to evolving threats within containerized infrastructures.

Future work aims to build upon its foundation by incorporating advanced features and expanding its scope. One key area of focus is integrating anomaly detection systems powered by machine learning (ML) and artificial intelligence (AI)[30]. By analyzing audit logs and identifying unusual patterns in pod behavior or network traffic, these models can provide real-time detection and response to potential threats, further strengthening the framework's capabilities. Extending KubeDeceive to other cloud-native platforms, such as OpenShift, Docker Swarm, and Amazon EKS, represents another promising direction. Additionally, automating the generation of deceptive Kubernetes resources, such as fake pods, services, and secrets, is a priority. Automation would enable dynamic updates to decoys based on observed attacker strategies and ensure seamless integration with CI/CD pipelines, promoting continuous deployment of updated deception tactics.

6 Acknowledgments

I am deeply thankful to my wife and family for their love, understanding, and encouragement, which have been my greatest source of strength. I extend my heartfelt gratitude to my supervisors for their invaluable guidance, expertise, and unwavering support, which have been pivotal to the progress of this research. I also wish to express my sincere appreciation to Loay Abdelrazek for his constant technical support and mentorship, as well as to my colleagues for their collaborative spirit and contributions to my professional growth. Thank you all for being an integral part of this journey.

References

[1] *CSCI 2022 International Conference on Computational Science and Computational Intelligence: proceedings: 14-16 December 2022, Las Vegas*. The Institute of Electrical and Electronics Engineers, 2022.

[2] OWASP Kubernetes Top Ten — OWASP Foundation. URL `https://owasp.org/www-project-kubernetes-top-ten/`. Accessed: Jul. 23, 2023.

[3] GitHub - quay/clair: Vulnerability Static Analysis for Containers. URL `https://github.com/quay/clair`. Accessed: Nov. 28, 2023.

[4] Kubernetes - checkov. URL `https://www.checkov.io/4.Integrations/Kubernetes.html`. Accessed: Nov. 28, 2023.

[5] GitHub - Shopify/kubeaudit: kubeaudit helps you audit your Kubernetes clusters against common security controls. URL `https://github.com/Shopify/kubeaudit`. Accessed: Nov. 28, 2023.

[6] Open Policy Agent. URL `https://www.openpolicyagent.org/`. Accessed: Nov. 28, 2023.

[7] Honeykube – Unveiling The Who, how and what of your cyber enemies. URL `https://honeykube.ch/`. Accessed: Nov. 28, 2023.

[8] C. Gao, Y. Wang, X. Xiong, and W. Zhao. MTDCD: An MTD Enhanced Cyber Deception Defense System. In *IMCEC 2021 - IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference*, pages 1412–1417. Institute of Electrical and Electronics Engineers Inc., 2021. doi: 10.1109/IMCEC51613.2021.9482133.

[9] J. Sun and K. Sun. DESIR: Decoy-enhanced seamless IP randomization. In *Proceedings - IEEE INFOCOM*, volume 2016-July, 2016. doi: 10.1109/INFOCOM.2016.7524602.

[10] T. B. Shade, A. V. Rogers, K. J. Ferguson-Walter, S. B. Elson, D. K. Fayette, and K. E. Heckman. The MoonRaker study: An experimental evaluation of host-based deception. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, pages 1875–1884. IEEE Computer Society, 2020. doi: 10.24251/hicss.2020.231.

[11] Y. Shi et al. CHAOS: An SDN-Based Moving Target Defense System. *Security and Communication Networks*, 2017, 2017. doi: 10.1155/2017/3659167.

[12] M. S. I. Sajid et al. SODA: A System for Cyber Deception Orchestration and Automation. In *ACM International Conference Proceeding Series*, pages 675–689. Association for Computing Machinery, 2021. doi: 10.1145/3485832.3485918.

[13] X. Han, N. Kheir, and D. Balzarotti. Evaluation of deception-based web attacks detection. In *MTD 2017 - Proceedings of the 2017 Workshop on Moving Target Defense, co-located with CCS 2017*, volume 2017-January, pages 65–73, 2017. doi: 10.1145/3140549.3140555.

[14] Aqua Cloud Native Security, Container Security Serverless Security. URL `https://www.aquasec.com/`. Accessed: Nov. 28, 2023.

[15] Calico - IBM Documentation. URL `https://www.ibm.com/docs/en/cloud-private/3.2.x?topic=ins-calico`. Accessed: Nov. 28, 2023.

[16] A. Aly, M. Fayez, M. M. Al-Qutt, and A. Hamad. Navigating the deception stack: In-depth analysis and application of comprehensive cyber defense solutions. *International Journal of Intelligent Computing and Information Sciences*, 23(4):50–65, Dec. 2023. doi: 10.21608/IJICIS.2023.247380.1306.

[17] K. Ferguson-Walter, M. Major, D. Van Bruggen, S. Fugate, and R. Gutzwiller. The world (of CTF) is not enough data: Lessons learned from a cyber deception experiment. In *Proceedings - 2019 IEEE 5th International Conference on Collaboration and Internet Computing, CIC 2019*, pages 346–353, 2019. doi: 10.1109/CIC48465.2019.00048.

[18] Matrix - Enterprise — MITRE ATTCK. URL `https://attack.mitre.org/matrices/enterprise/containers/`. Accessed: Jul. 23, 2023.

[19] K01: Insecure Workload Configurations — OWASP Foundation. URL `https://owasp.org/www-project-kubernetes-top-ten/2022/en/src/K01-insecure-workload-configurations`. Accessed: Jul. 23, 2023.

[20] K05: Inadequate Logging — OWASP Foundation. URL `https://owasp.org/www-project-kubernetes-top-ten/2022/en/src/K05-inadequate-logging`. Accessed: Jul. 23, 2023.

[21] kind. URL `https://kind.sigs.k8s.io/`. Accessed: Aug. 07, 2023.

[22] minikube start — minikube. URL `https://minikube.sigs.k8s.io/docs/start/`. Accessed: Aug. 07, 2023.

[23] clientgo package - k8s.io/client-go - Go Packages. URL `https://pkg.go.dev/k8s.io/client-go`. Accessed: Jul. 23, 2023.

[24] SSH server, sshd, SSH daemon - How to get one, how it works, how to configure. URL `https://www.ssh.com/academy/ssh/server`. Accessed: Jul. 23, 2023.

[25] Auditing — Kubernetes. URL `https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/`. Accessed: Jul. 23, 2023.

[26] Kasun Indrasiri and Danesh Kuruppu. *gRPC: Up and Running: Building Cloud Native Applications with Go and Java*. Google Books. URL `https://books.google.com.eg/books?hl=en&lr=&id=883LDwAAQBAJ&oi=fnd&pg=PR2&dq=+GRPC+Calls+kubernetes+&ots=juxSbM6yBy&sig=4rkW79XYSdxjTLR2dhm7rT90Q1k&redir_esc=y#v=onepage&q=GRPC%20Calls%20kubernetes&f=false`. Accessed: Jul. 23, 2023.

[27] B. Pingle, A. Mairaj, and A. Y. Javaid. Real-World Man-in-the-Middle (MITM) Attack Implementation Using Open Source Tools for Instructional Use. In *IEEE International Conference on Electro Information Technology*, volume 2018-May, pages 192–197, 2018. doi: 10.1109/EIT.2018.8500082.

[28] Seema Kumari. Ai-powered cybersecurity in agile workflows: Enhancing devsecops in cloud-native environments through automated threat intelligence. *Journal of Science & Technology*, 1(1):809–828, Dec 2020. URL `https://thesciencebrigade.com/jst/article/view/425`.

[29] B. Ksiezopolski, K. Mazur, M. Miskiewicz, and D. Rusinek. Teaching a Hands-On CTF-Based Web Application Security Course. *Electronics*, 11(21):3517, 2022. doi: 10.3390/ELECTRONICS11213517.

[30] A. Aly, M. Fayez, M. M. Al-Qutt, and A. M. Hamad. Multi-class threat detection using neural network and machine learning approaches in kubernetes environments. pages 103–108, 2024. doi: 10.1109/ICCI.2024.1234567.

Authors

Abdelrahman Aly earned his Bachelor's degree in computer science from Ain Shams University, where he is also pursuing his Master's degree. His primary field of study is security, emphasizing expertise in DevSecOps practices and the integration of robust security solutions. In addition, he is an experienced security engineer, skilled in combining offensive and defensive techniques to enhance the resilience of modern infrastructures. Abdelrahman is a Teaching Assistant at Ain Shams University in Cairo, Egypt. He has co-authored ”Navigating the Deception Stack: In-Depth Analysis and Application of Comprehensive Cyber Defense Solutions” (Int. J. Intell. Comput. Inf. Sci., 2023) and ”Multi-Class Threat Detection Using Neural Network and Machine Learning Approaches in Kubernetes Environments” (2024 6th Int. Conf. Comput. Informatics).

Mahmoud Fayez holds a PhD from Ain Shams University, where he is a researcher in the Computer Systems Department. His expertise includes high-performance computing, GPU acceleration, and elastic optical networks.

Mirvat Al-Qutt holds a PhD from Ain Shams University, specializes in high-performance computing and computational optimization. Her work includes neural network-based hardware configuration prediction and big data solutions for intensive computations.

Ahmed M. Hamad is a Professor of Computer Systems at Ain Shams University, Cairo, Egypt. He earned his Ph.D. in Electrical Engineering from the University of Orsay, France, in 1981, specializing in electronic systems analysis. His academic journey includes roles as Demonstrator, Assistant Professor, and eventually Professor at the Faculty of Computer and Information Sciences, Ain Shams University, starting in 1998. Prof. Hamad has authored over 70 papers in various fields of information technology, contributing significantly to both international and local conferences and journals.