A Cost-Effective Approach to Proactive and Non-disruptive Attack Mitigation in Kubernetes Clusters

Sima Bagheri

A Thesis

in

The Department

of

The Concordia Institute

for

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Information Systems Security) at

Concordia University

Montréal, Québec, Canada

April 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Sima Bagheri**

Entitled: **A Cost-Effective Approach to Proactive and Non-disruptive Attack Mitigation in Kubernetes Clusters**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Information Systems Security)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr.

_____ External Examiner
Dr.

_____ Examiner
Dr.

_____ Supervisor
Dr.

Approved by _____
Martin D. Pugh, Chair
Department of The Concordia Institute
for
Information Systems Engineering

_____ 2024

_____

Amir Asif, Dean
Faculty of Engineering and Computer Science

Abstract

A Cost-Effective Approach to Proactive and Non-disruptive Attack Mitigation in Kubernetes Clusters

Sima Bagheri

A large-scale cluster of containers managed with an orchestrator like Kubernetes are behind many cloud-native applications today. However, the weaker isolation provided by containers means attackers can potentially exploit a vulnerable container and then escape its isolation to cause more severe damages to the underlying infrastructure and its hosted applications. Besides, Kubernetes reportedly suffers from security vulnerabilities and misconfigurations which may lead to severe security threats.

Defending against such an attack using existing attack detection solutions can be challenging. Due to the well known high false positive rate of such solutions, taking aggressive actions upon every alert can lead to unacceptable service disruption. On the other hand, waiting for security administrators to perform in-depth analysis and validation could render the mitigation too late to prevent irreversible damages, e.g., denial of service. In this thesis, we propose WARP, a cost-effective approach to proactive and non-disruptive attack mitigation to address such security challenges for Kubernetes clusters. First, our approach is proactive in the sense that it performs mitigation based on predicted (instead of real) attacks, which prevents irreversible damages. Second, our mitigation approach is designed to be non-disruptive and it is achieved through live migration of containers, which causes

no service disruption even in the case of false positives. Finally, to realize the full potential of this approach in containers migration, we formulate the inherent tradeoff between security and cost (delay) as a multi-objective optimization problem and propose a heuristic algorithm to efficiently achieve a high level of threat reduction with minimal imposed delay.

Our evaluation results show that WARP can successfully mitigate up to 81% of the attacks, and our heuristic algorithm achieves up to 30% more threat reduction and 7% less delay while being 37 times faster compared to a standard optimization solution.

Acknowledgments

I would like to first thank my advisors, Dr. Lingyu Wang and Dr. Suryadipta Majumdar for their support and guidance.

I also want to thank Hugo and Mahmood for their friendship, camaraderie, kindness, support, compassion, and humor throughout the past years.

More importantly, I would like to share my love and gratitude for the support I've been given by my parents, my brother Sina, and my best friend Maryam. Thank you for indulging my decision to stay a student for one decade (!) and specially for the care you've all shown me in the last three years when I needed the most.

I dedicate this thesis to my people in Iran who are living their most courageous lives.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Problem Statement

Containerization is an increasingly popular choice for deploying large-scale cloud-native applications due to its inherent efficiency, agility, and flexibility. A container orchestrator such as Kubernetes (a widely adopted container orchestration platform [21]) makes it easy to deploy, manage, and maintain a large amount of containerized applications in a Kubernetes cluster. However, it is well known that, compared to full-fledged virtual machines, containers provide weaker isolation between the application and the underlying host [9]. Moreover, popular container images are shown to be buggy with vulnerabilities [8], and even the cluster orchestrator itself may contain vulnerabilities or misconfigurations (e.g., the Kubernetes privilege escalation vulnerability showcased at Black Hat USA 2022[40]). Such weaknesses may render Kubernetes clusters an attractive target to attackers, who can exploit a vulnerable container for initial accesses, and then escape the container to cause more severe damages to the underlying infrastructure and other containers as well as the applications they host. Additionally, security is typically an afterthought in the deployment of containerized applications and security breaches and attacks are usually detected after the fact, which could result in irreversible damages (e.g., denial of service or information

leakage).

There already exist attack prevention solutions for containers and Kubernetes, such as the Open Policy Agent (OPA) and Gatekeeper [25] combination for runtime security policy enforcement, as well as the Seccomp (SECure COMPuting) filter [31], which is for preventing containers from accessing certain host-level resources through blocking system calls. The Seccomp filters are also leveraged in existing works [46, 49, 58] to block system calls that are not normally used by the applications. Nonetheless, those preventive solutions can only reduce the general attack surface of containers to some extent, and cannot prevent all attacks. Moreover, these solutions leverage disruptive defenses (i.e., blocking) which is not desirable if a false positive attack detection occurs. Therefore, non-disruptive attack mitigation solutions are still necessary. To that end, Falco [17] provides a popular open source solution for detecting attacks on both the containers and the infrastructure, either with the default rules or by developing custom rules for specific attacks. Although such detection solutions can enable a security administrator to take notice and keep track of ongoing attacks, they do not provide a direct, automated, and non-disruptive way to stop such attacks.

Compared to prevention and detection, automated mitigation of detected attacks in Kubernetes clusters has received less attention (a detailed review of related works is given in Chapter 3). Attack mitigation in practice still largely depends on the intervention of security administrators, who will first investigate the alerts reported by a detection solution such as Falco [17], and then take corresponding mitigation actions to stop or slow down the attack progress. However, such a traditional mitigation approach may face several major challenges in the specific context of defending Kubernetes clusters (we will further illustrate those limitations through an example in Section).

In this thesis, we propose a novel approach named *WARP* to address the aforementioned limitations. First, we provide a fully *automated* solution for performing attack mitigation

in Kubernetes clusters. This helps avoid various limitations of manual efforts and makes attack mitigation scalable enough for large-scale applications. Second, upon a detected attack, we perform *proactive* mitigation actions to prevent the attacker from reaching other co-located or connected containers. Taking such early mitigation actions prior to attack propagation can limit the scope of attack damages and prevent irreversible losses. Third, we realize the proactive mitigation through a *non-disruptive* type of mitigation actions, i.e., live migration of Kubernetes Pods. Such mitigation actions can avoid service disruption in case the detected attack turns out to be a false alarm later on, since live migration is transparent to tenants, and is already being routinely performed in Kubernetes clusters for other purposes (e.g., load balancing or maintenance). Fourth, we provide *cost-effective* mitigation plans through multi-objective optimization. This allows us to maximize the amount of threat reduction achieved by our mitigation actions, while minimizing their potential delay and overhead. Finally, we provide *customized* mitigation to different groups of logically separated containers. This enables our solution to tailor the attack mitigation to the different requirements of tenants in terms of the security level and its associated cost, as typically specified in their Service Level Agreements (SLAs).

1.2 Research Gap

In summary, WARP mainly fills the below gaps derived from the state-of-the-art works as follows.

- The manual attack mitigation efforts from an admin can hardly be scalable enough for a large Kubernetes cluster. Such efforts are usually tedious for an admin, limited by his/her knowledge and skills, and prone to human errors. These can be exacerbated when managing a large Kubernetes cluster, since many alerts reported by a detection solution (e.g., Falco) can easily cause the admin to develop alert fatigue

and subsequently miss the actual attack events.

- Moreover, in making such a mitigation decision, an admin may likely face a dilemma. First, due to the well known high false positive rate of most attack detection solutions, the administrator may be aware that taking an aggressive action such as shutting down the victim container, or disconnecting its connected users could lead to unacceptable service disruption. On the other hand, the administrator knows the importance of a timely mitigation action, since performing in-depth analysis and validation of the reported alerts may take too long, and render the subsequent mitigation too late to prevent irreversible damages, such as potential large-scale information leakage or denial of service (DoS).

- Finally, knowing that any mitigation action may unavoidably incur a cost (e.g., delay to services), the admin must also strive to balance security against cost through an efficient strategy towards attack mitigation. Moreover, the multi-tenant nature of cloud and its diverse applications both imply that the tenants may have very different requirements in terms of security and costs (e.g., an autonomous vehicle application may regard negligible delay as its top priority, while a co-located smart parking application may value security more than delay). Therefore, it is highly challenging for the administrator of a Kubernetes cluster to cater to those different requirements, all through manually adjusting his/her mitigation actions.

WARP first provides an automated and proactive attack detection and mitigation to tackle the problem of manual attack investigation and further alerts fatigue. Second, we propose a non-disruptive mitigation solution (i.e., live migration) to address the challenge of false positive and guarantee the service continuity. Lastly, we provide an optimization for the mitigation selection to minimize the added cost of the mitigation while maximizing the benefit (i.e., security or threat reduction). To the best of our knowledge, WARP is the

first work offering an optimized and automated proactive attack mitigation solution for the containerized environments.

1.3 Thesis Contribution

In summary, the main contributions of this thesis are as follows:

- As per our knowledge, this is the first proactive attack mitigation approach that can also avoid the service disruption caused by potential false alarms.
- We build the first large-scale Kubernetes attack dataset with 231k Falco alerts based on real-world APT attacks simulated in a controlled environment [18].
- Using this Falco alerts dataset, we develop an attack prediction model to learn attackers' tactics and strategies in the form of MITRE ATT&CK framework [23]. This prediction model is learned offline and applied at runtime for attack prediction.
- We develop a series of techniques to predict the attacker's probable next moves after an initial attack is detected, identify the Pods (i.e., smallest deployable units of Kubernetes) that could become the next targets of attack propagation, evaluate the risk of those Pods to decide when mitigation should be triggered, and finally perform non-disruptive attack mitigation through migrating the Pods at risk according to an optimal migration plan.
- To derive such a cost-effective mitigation plan, we formulate the migration options and corresponding costs as a multi-objective optimization problem, prove its NP-hardness, and develop an efficient heuristic algorithm to find solutions that can maximize threat reduction with minimal cost.
- We also leverage network slicing [1] to apply different mitigation plans to different slices (groups of logically separated containers), such that our attack mitigation can

be customized for each tenant to satisfy its unique security requirement and cost constraint.

- We implement our approach based on a Kubernetes cluster deployed with Falco. We evaluate the effectiveness and performance of our solution through experiments. Our evaluation results show that WARP can mitigate up to 81% of the attacks, and our heuristic algorithm achieves up to 30% more threat reduction and 7% less delay while being 37 times faster compared to a standard optimization solution.

1.4 Related Publications

Conference Paper. Our work for warping the defence timeline: non-disruptive proactive attack mitigation for Kubernetes clusters has been published as an article in a peer-reviewed conference's proceedings:

> Warping the Defence Timeline: Non-disruptive Proactive Attack Mitigation for Kubernetes Clusters. Sima Bagheri, Hugo Kermabon-Bobinnec, Suryadipta Majumdar, Yosr Jarraya, Lingyu Wang, and Makan Pourzandi. *Proceedings of the IEEE International Conference on Communications (ICC), Rome, Italy, 28 May - 01 June, 2023. [Published]*

Journal Paper. Moreover, we extended the conference paper for ACE-WARP: a cost-effective approach to proactive and non-disruptive attack mitigation in Kubernetes clusters and submitted our manuscript to the IEEE Transactions of Information Forensics and Security as T-IFS-16834-2023.

> ACE-WARP: A Cost-Effective Approach to Proactive and Non-disruptive Attack Mitigation in Kubernetes Clusters. Sima Bagheri, Hugo Kermabon-Bobinnec,

Mohammad Ekramul Kabir, Suryadipta Majumdar, Lingyu Wang, Yosr Jarraya, Boubakr Nour, and Makan Pourzandi. *IEEE Transactions of Information Forensics and Security. [Submitted]*

1.5 Contribution of Co-authors/Collaborators

The student co-authors' contributions to the aforementioned publications are as follows: Sima Bagheri contributed to the motivation, approach and design, implementation as well as experiments. Hugo Kermabon-Bobinnec contributed to the testbed configuration, and part of the genetic algorithm implementation.

1.6 Outline

Change according to your outline The rest of this thesis is organized as follows: Chapter 2 provides the necessary background for this dissertation. Chapter 3 covers the literature related to this thesis. We present our solution for proactive and non-disruptive attack mitigation in Chapter 4. We present our conclusion, limitations and future work in Chapter 7.

Chapter 2

Background and Motivation

This chapter provides a background on containerization and security policy compliance, presents the motivation, and defines our threat model.

2.1 Background

Containerization and Kubernetes. The concept of containerization allows developers to create and deploy applications faster and more securely. In contrast to virtual machines (VMs), containers share the underlying operating system kernel and do not require the overhead of associating an operating system per application. Kubernetes [21] is one of the most widely adopted container orchestrators for cloud-native applications [36]. On the left side of Fig.2.1, the Kubernetes cluster includes one Master Node and two Worker Nodes hosting applications and services within *Pods* as the smallest deployable units of a Kubernetes cluster. In this case, Pods are hosting some network functions in a 5G Service-Based Architecture (SBA) [20]. According to the communication policies between Pods, some of the Pods have explicit connections shown as a direct line between them.

Kubernetes Runtime Security. The cluster shown in Fig. 2.1 is also configured with

Falco [17] is a cloud-native security tool, and a popular runtime security solution for Kubernetes which reports real-time security alerts on suspicious events in the cluster. Falco employs an agent on each Worker Node to monitor and detect malicious activities, which will then be reported in the form of alerts to the Master Node agent.

Network Slicing. Network slicing, as a network architecture method, allows multiple logical networks (slices) to co-exist on the same virtual network infrastructure (e.g., Kubernetes cluster) [1]. Each slice is logically isolated and can host services sharing similar requirements [65]. Therefore, this concept enables a Kubernetes cluster to host multiple services with different requirements of security and delay, e.g., a 5G application for autonomous vehicles may need close to zero delay and high security, whereas other applications (e.g., smart parking, and smart agricultural services) may value security over delay [1]. The left side of Fig. 2.1 depicts how those 5G network functions can be separated into multiple slices (illustrated in different colors). In WARP, we take advantage of network slicing to customize attack mitigation for different tenants.

2.2 Motivatiing Example

Figure 2.1 depicts an attack scenario exploiting a real-world vulnerability [13] in a Kubernetes cluster hosting a 5G core [20].

Attack Scenario. The upper-left corner of Fig. 2.1 depicts a container escaping attack scenario (assuming security measures such as AppArmor are disabled and *SYS_ADMIN* capabilities are enabled) as follows. First, the attacker exploits the above-mentioned vulnerability in the rightmost container (which hosts the 5G Access and Mobility Management Function (AMF)) to escalate his/her privilege to root. Second, to escape the Pod's isolation and get into Worker Node 1, s/he creates a new control group (`cgroup`) by mounting a `cgroup` controller inside the container; s/he enables the `notify_on_release` option

and specifies a command to be executed on the host in the `release_agent` file; subsequently, once the `cgroup` process terminates, the command in the `release_agent` file is executed on the host, allowing the attacker to escape into the Node. Finally, the attacker now gains unauthorized accesses to other containers (e.g., Unified Data Management (UDM)) inside the co-located Pods.

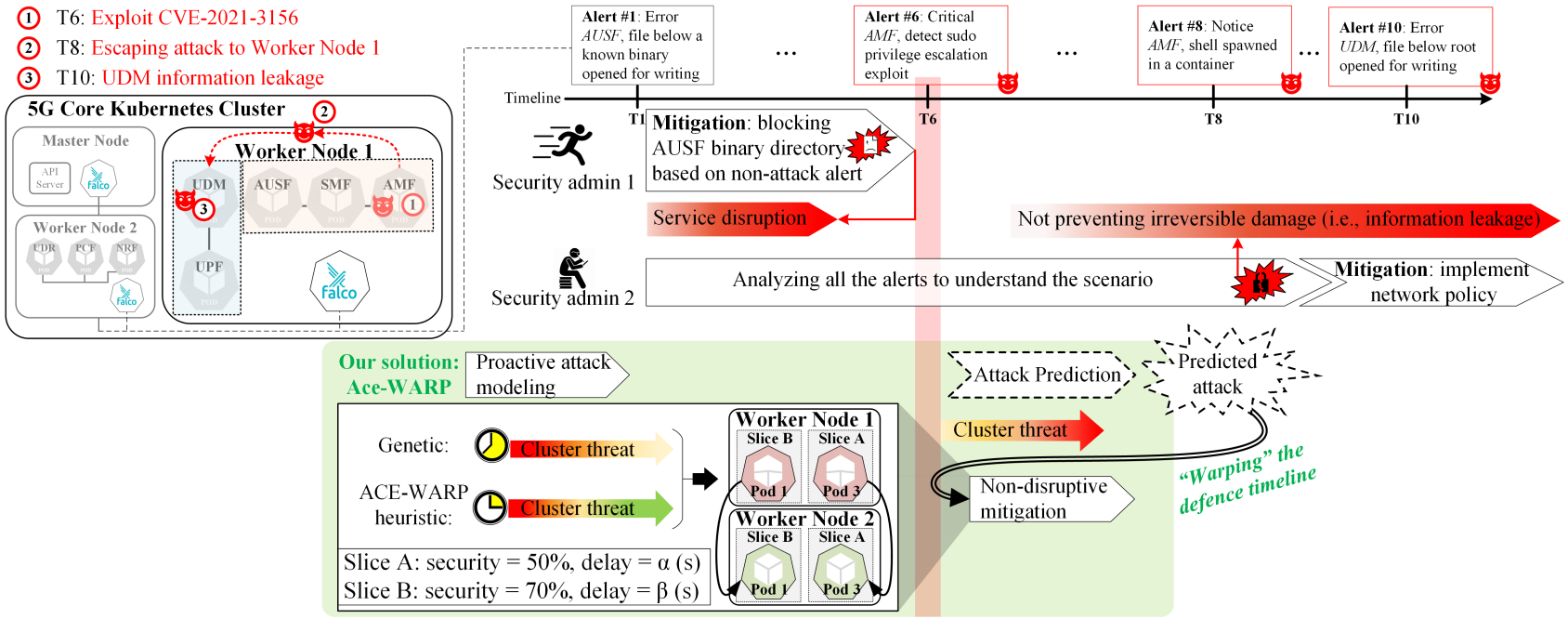


Figure 2.1: Motivating example

As shown in the upper-left corner of Fig. 2.1, those three attack steps are assumed to occur at time `T6`, `T8`, and `T10`, respectively. Falco raises an alert for each attack step, as shown above the timeline in the upper-right corner of Fig. 2.1 (`Alert #1` can be ignored for now and will be explained later). First, at time `T6`, `Alert #6` reports a detected privilege escalation in `AMF`. Second, at `T8`, `Alert #8` reports a shell being spawned inside a container. Finally, at `T10`, `Alert #10` reports a suspicious file access. To mitigate the attack based on those alerts, the right side of Fig. 2.1 also shows three potential approaches.

1. Early (but disruptive) mitigation. Assume Security admin #1 would like to minimize the security threat by taking an aggressive mitigation approach as follows. Upon the detection of `Alert #1` at time `T1`, which indicates a write operation in the binary directory of the Authentication Server Function (`AUSF`) Pod, the admin promptly takes action to block further access to its directory. However, after further investigation, s/he realizes that `Alert #1` is actually a false positive indicating no real threat. Nonetheless, his/her previous action has already caused unwanted disruption to the `AUSF` service, which is certainly not acceptable.

2. Non-disruptive (but late) mitigation. On the other hand, assume Security admin #2 takes a more cautious approach to avoid causing any service disruption. Therefore, the admin carefully analyzes each alert (till `Alert #10`), and eventually succeeds to identify `Alert #1` as a false positive and fully understand the attack scenario. However, by the time s/he starts to implement the mitigation action (of enforcing a new network policy) at time `T10`, it is already too late, since the confidential 5G user data managed by `UDM` is already leaked out to the attacker, which is a damage that cannot be reversed.

3. Our solution. As shown at the bottom of the figure, WARP can overcome the above limitations through a proactive and non-disruptive approach as follows. First, it proactively builds an attack prediction model offline (before `T1`). Next, upon receiving an alert at runtime, it evaluates the risk of attack propagation by predicting potential future attacks (using

the attack prediction model), and triggers a mitigation action once such a risk exceeds a predefined threshold. For instance, at time `T1`, it decides that the attack propagation risk of `Alert #1` is not yet sufficient to trigger a mitigation action. However, at time `T6`, by applying the prediction model to `Alert #6`, it predicts that a future escaping attack is probable (which would actually happen at `T8` if not prevented), and hence decides the risk is high enough (illustrated as the yellow to red arrow) to trigger the mitigation. Therefore, at `T6`, it performs live migration of Pods following an optimal migration plan, e.g., by migrating the Pods containing `UDM`, and `UPF` a to another Node. As a result, even though the attacker may still escape to the Node at `T8`, the threat to the cluster is decreased as s/he would no longer have access to `UDM`, and `UPF`. Note the live migration would not cause any service disruption even if `Alert #6` later turns out to be a false positive. Finally, as demonstrated in the magnified area, our heuristic optimization algorithm can find an optimal migration plan more efficiently, with a higher level of threat reduction and less cost (delay), compared to a standard (genetic) optimization algorithm. In addition, our solution provides customizable mitigation to satisfy the tenants different requirements via network slicing.

2.3 Threat Model

We mainly focus on mitigating the potential damages caused by attack propagation inside a Kubernetes cluster. Therefore, the in-scope threats may include any attacks that aim at compromising multiple Pods, whose initial step(s) can be detected by an existing detection or monitoring tool such as Falco. Such attacks may be launched by external attackers, a malicious tenant, or insiders whose goal is to cause large-scale damages through exploiting misconfigurations or vulnerabilities in a Kubernetes cluster.

As WARP is designed to mitigate the damages caused by detected attacks, both attacks that can be effectively prevented using existing solutions, and zero day attacks that can

completely evade existing detection tools are out of scope. Also, as we focus on mitigating attack propagation, attacks whose damages can be realized in a single step or in a single Pod are out of scope. Finally, attacks targeting lower-level infrastructures than containers, attacks that can temper with the integrity of WARP, Falco, and Kubernetes, and attacks that can breach the isolation of network slicing are out of the scope of this thesis.

Chapter 3

Related Work

WARP lies at the intersection of attack detection, attack investigation and provenance, attack mitigation, proactive security approaches, and resource placement optimization. Therefore, this section reviews and compares WARP to related works in those areas. First, Table 3.1 presents a comparison of WARP with existing solutions in terms of their methods, environments, tenant adjustability, and different features (proactiveness, model learning, leveraging the MITRE ATT&CK framework, attack coverage, mitigation, and being non-disruptive). The comparison shows WARP covers all the aforementioned features, operates in the Kubernetes environment, and is adjustable with the tenants' requirements.

Attack Detection and Mitigation. Most of the existing attack detection and mitigation solutions are limited to detecting attacks or security policy violations in a reactive manner, which cannot prevent irreversible attack damages such as information disclosure or denial of service. For instance, Sysdig [33], Falco [17], and OPA/Gatekeeper [25] are runtime attack detection tools designed for containerized environments. The authors in [70] presented *KubAnomaly*, a learning-based anomaly detection approach for security monitoring in Kubernetes. Several solutions [48, 72] have been proposed to analyze the performance data of Kubernetes containers and Pods, and detect anomalies. Unlike those reactive solutions, WARP provides a proactive attack mitigation solution by predicting and mitigating

Table 3.1: Comparing existing solutions with WARP.

Ref.	Method	Features						Environment	Tenant Adjustability
		Proactive	Model Learning	MITRE ATT&CK	Attack Coverage	Mitigation	Non-disruptive		
PROLEMus [68]	MAC protocol	●	●	-	◑	-	N/A	Cognitive Radio Network (CRN)	N/A
ProSAS [64]	Custom Algorithm	●	●	-	●	●	-	Cloud	-
Ma et al. [60]	Custom Algorithm	●	-	-	◑	●	●	Kubernetes	-
ProSPEC [57]	Custom Algorithm	●	●	-	●	●	-	Kubernetes	-
Liu et al. [59]	Watermark	●	-	-	◑	-	N/A	Cyber-physical system	N/A
ATLAS [37]	Custom Algorithm	-	●	-	●	-	N/A	Windows	N/A
NoDoze [52]	Custom Algorithm	-	-	-	●	-	N/A	Windows	N/A
UNICORN [51]	Custom Algorithm	●	●	-	◑	-	N/A	Windows/Linux	N/A
WARP	Custom Algorithm	●	●	●	●*	●	●	Kubernetes	●

The symbols (●), (◑), (-) and N/A mean fully supported, partially supported, not supported and not applicable, respectively.
*WARP attack coverage relies on underlying detection tool (**??**).

the attacker's probable next move after an initial attack is detected.

Proactive Attack Detection and Mitigation. There exist efforts on security policy compliance for container-based (e.g., Kubernetes [57]) and traditional cloud environments([35, 62, 61, 64]). The authors in [62, 63, 64] propose proactive security policy verification where the mitigation is performed at runtime. ProSPEC [57] extends such proactive security auditing to Kubernetes. However, as these approaches only start the mitigation after critical events have occurred, they may still be too late to prevent irreversible damages. Several efforts focus on proactive detection of specific attacks. The authors in [59] propose a watermark-based approach to proactively defend against man-in-the-middle attacks. PROLEMus [68] is a learning-based approach for addressing denial-of-service (DoS) attacks. Unlike those existing proactive approaches, WARP is not limited to specific attacks, and it employs a non-disruptive mitigation approach (migration of Pods), which can be

launched well before the attack events actually occur yet without the risk of causing service disruption.

Provenance Analysis. Provenance analysis solutions aim to investigate the root cause of detected attacks and do not provide attack mitigation [54, 53, 71]. UNICORN [51] proposes a graph-based technique to investigate contextual information of stealthy APT attack steps without predefined attack signatures. NoDoze [52] focuses on attack triaging using provenance graphs to identify anomalous paths for further manual response. Holmes [66] correlates suspicious events with the MITRE ATT&CK framework to trigger a detection signal and provide the analyst with a high-level graph for further actions. The authors in [37] design a learning-based approach to build a sequence-based model out of a provenance graph to extract the attack story. WARP can leverage analyst's feedback after provenance analysis to improve its attack detection accuracy and risk formulation, while it can complement provenance-based solutions with its mitigation capability.

Resource Placement Optimization. The problem of optimal resource placement and scheduling has been studied [45, 41, 47, 38, 69, 56]. Since this problem is generally intractable, heuristics, meta-heuristics, and machine learning algorithms are proposed to solve the problem efficiently. The authors in [45, 43] propose a near-optimal heuristic for virtual network functions (VNF) placement and scheduling in cloud environments with the objective of minimizing energy consumption and resource utilization. VNF placement is also studied in the 5G C-RAN context in [38]. CODO [41] presents a heuristic solution to the problem of firewall rule ordering in the cloud to optimize network traffic, QoS/throughput, delay, and security. In Kubernetes, the authors in [47] propose a genetic algorithm to solve the non-linear problem of microservices placement and maximize throughput. Resource allocation to containers based on hyper-heuristic algorithms is studied in [69]. The authors in [56] study the problem of scheduling mobile charging infrastructure for vehicles. The placement optimization to container network functions is studied in [39]. Similarly,

the reduction of cluster threat and delay are the main objectives for the Pods migration optimization problem formulated in this work.

Chapter 4

WARP: A Cost-Effective Approach to Proactive and Non-disruptive Attack Mitigation in Kubernetes Clusters

4.1 Overview

Figure 4.1 shows an overview of WARP, including the overview of its integration to Kubernetes cluster (on the left) and overview of its major steps (on the right).

Integration Overview. As shown in left side of Fig. 4.1, WARP is integrated with the Kubernetes cluster. Specifically, there is an WARP agent deployed in each Node (both Worker and Master) of cluster. The WARP agent in Master Node is connected to Falco to receive alerts collected from those Nodes. Additionally, the tenants' service-level agreement (SLA) policies, such as security and delay requirements, are also the input to WARP.

Approach Overview. WARP approach is performed in two major phases: (i) *Offline Modeling*, and (ii) *Runtime Detection and Mitigation*. During the offline phase, WARP first formulates an optimization model to derive cost-effective migration plan for Pods while

reducing the overall cluster threat. It also builds an attack prediction model to predict the attacker's next steps. During the runtime phase, WARP proactively predicts attack steps from a received Falco alert using the prediction model and calculates the associated risk for each Pod. If the risk is higher than a predefined threshold (derived from tenants' inputs and discussed later), WARP initiates non-disruptive mitigation where it first derives Pods migration plan using the optimization model to minimize the cluster threat with minimum imposed delay. Thus, WARP proactively prevents the attacker from proceeding with the attack. In the following, we further illustrate WARP's approach using an example.

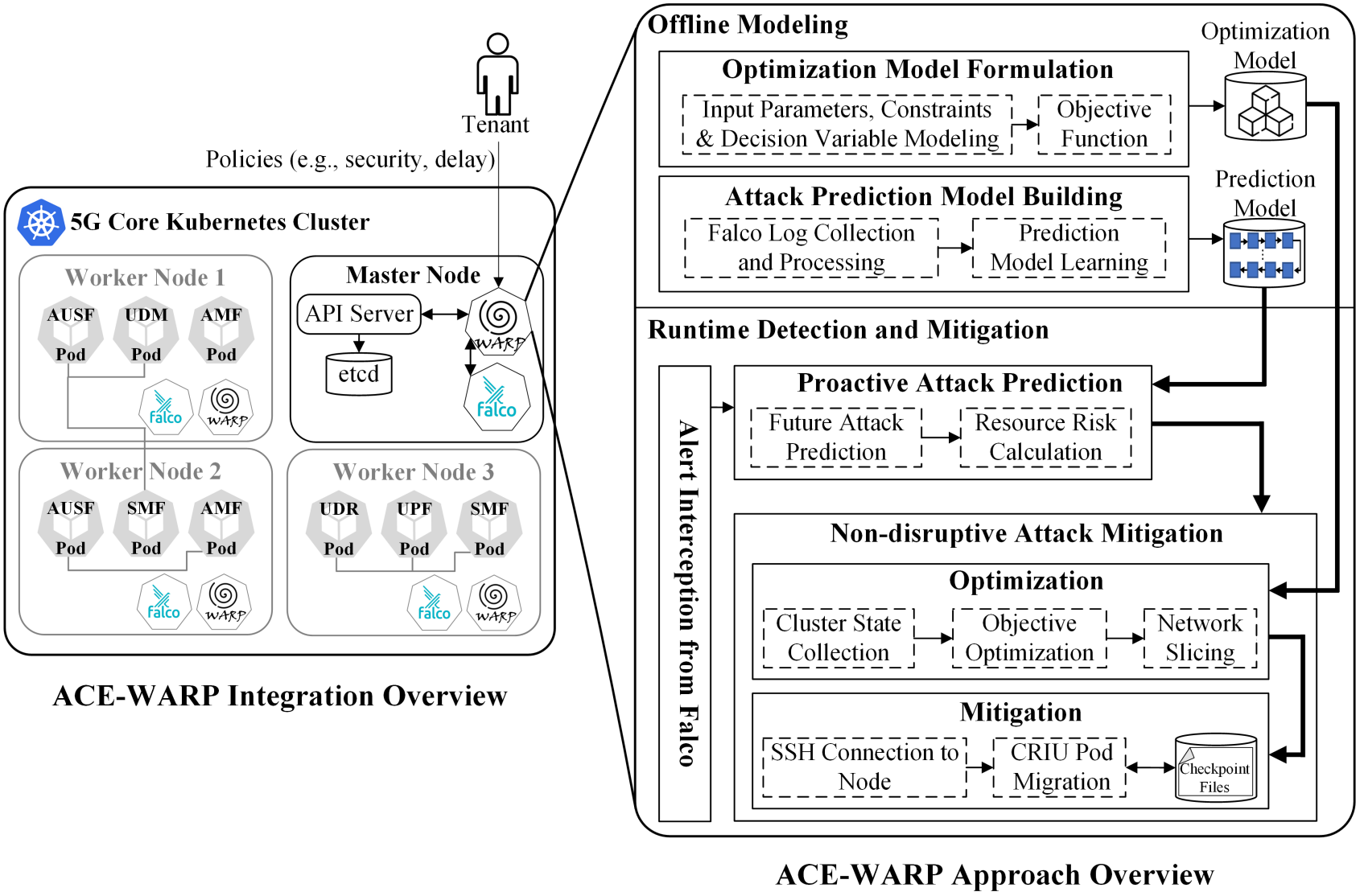


Figure 4.1: An overview of WARP.

Example 1. Figure 4.2 illustrates a toy example to show major steps of WARP for a cluster of two Worker Nodes (`Node 1` and `Node 2`) where Pods are distributed over two slices (`Slice A` and `Slice B`). First, WARP builds an attack prediction model (based on historical alerts) with three attack vertices where the probability for an attacker to move from `Privilege Escalation` to `Execution` is 0.31 (indicated as edge label). Second, at runtime, WARP receives `Alert 1` related to the same attack scenario described in our motivating example (exploiting `CVE-2021-3156`) with the alert tag `privilege_escalation`. Third, WARP finds the corresponding vertex (`Privilege Escalation`) in the prediction model for the current attack and then predicts potential next moves with the highest probability (`Execution`). Fourth, it calculates the estimated risks incurred by all the Pods (`Pod 1` - `Pod 6`) in the cluster using the risk formula (explained later in Section 4.2), and finds the Pods with a risk higher than their slice threshold. Fifth, while meeting the constraints and achieving the objective of minimizing the $Threat$ and $Delay$, it finds two migration options: (i) `Pod 3` to `Node 2` in `Slice A`, and (ii) `Pod 5` to `Node 1` in `Slice B`. Finally, WARP performs those two migrations and thus, the cluster threat is reduced.

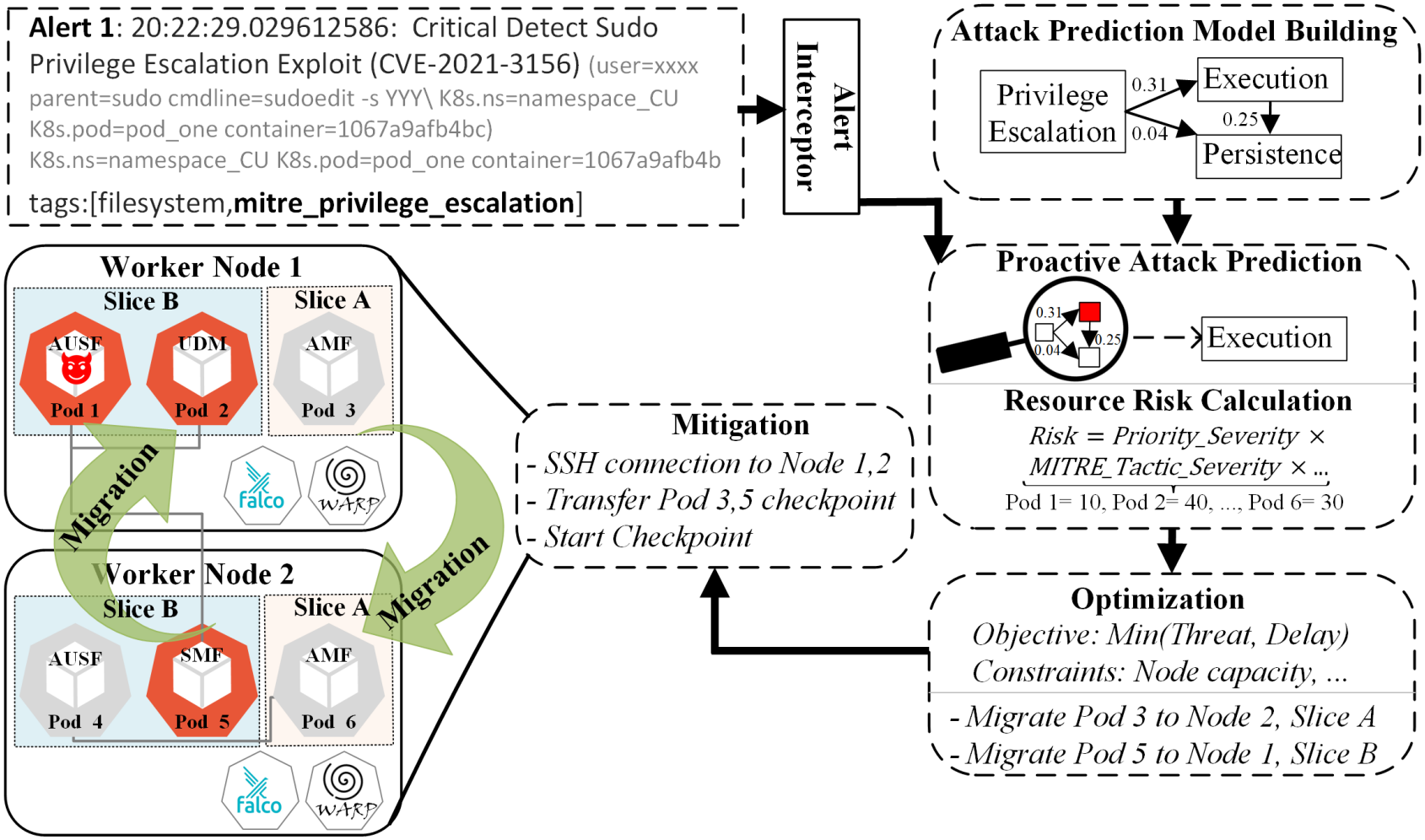


Figure 4.2: Illustrative example of WARP approach overview.

Table 4.1: List of input parameters.

Input Symbol	Description
P	Set of all $Pods\ p$ of the cluster
N	Set of all $Nodes\ n$ of the cluster
x_p^n	$x_p^n = 1$ if $Pod\ p$ is located at $Node\ n$; Otherwise 0
MD_p	Migration delay of $Pod\ p$
AV_p	Asset value of $Pod\ p$
$Size_p$	Size of $Pod\ p$ (in MB)
$Capacity_n$	Maximum capacity of $Node\ n$ (in MB)
p_{att}	The Pod under attack
$Con_{pp'}$	$Con_{pp'} = 1$ if $Pod\ p$ has a network connection with $Pod\ p'$; Otherwise 0

4.2 Offline Modeling

4.2.1 Optimization Model Formulation

The problem of optimal Pod placement in the cluster has a trade-off between the potential security threats to a cluster (i.e., cluster threat) and the delay involved with Pods' migration from one Node to another in a cluster (i.e., cluster delay). Therefore, in this step, WARP aims at formulating the Pods placement as an optimization problem. Table 4.1 lists the parameters used in our problem formulation. We formulate a mathematical model after defining the input parameters, constraints, decision variables, and optimization objectives.

Decision Variable. We define the decision variable y_p^n to represent if $Pod\ p$ is migrated to $Node\ n$ or not:

$$y_p^n = \begin{cases} 1, & \text{if } Pod\ p \text{ migrated to } Node\ n \\ 0, & \text{otherwise} \end{cases} \qquad \forall p \in P \tag{1}$$

Constraints. At a particular time, a Pod $p \in P$ can be located at only one Node $n \in N$:

$$\sum_{n} y_p^n = 1, \quad \forall p \in P \tag{2}$$

On the other hand, each Node $n \in N$ has a maximum capacity, $Capacity_n$, and hence, we cannot migrate more Pods to Node n beyond its capacity as shown in Equation (3).

$$\sum_{p} (y_p^n \times Size_p) \leq Capacity_n, \quad \forall n \in N \tag{3}$$

Here, $Size_p$ is the size of Pod p.

Objective. Our multi-objective optimization model simultaneously intends to: (1) minimize the cluster threat; and (2) minimize the migration delay for critical Pods as follows.

(1) Cluster Threat. According to our threat model, any Pod that can be reached from the attacked Pod after a finite number of lateral movements is under threat. Therefore, the threat of attack propagation in the cluster can be modeled as the summation of the asset values of all such Pods. More formally, we define a binary relation R over the set of Pods P to represent the collection of pairs of Pods that are reachable using direct connection or co-location:

$$R = \{(p_i, p_j) \mid p_i, p_j \in P, (Con_{p_i p_j} = 1) \vee (\exists n \in N, y_{p_i}^n \times y_{p_j}^n = 1)\} \tag{4}$$

Let R^* be the transitive closure [67] of R (i.e., the collection of pairs of Pods that are reachable via a finite number of applications of R). Given any attacked Pod p_{att}, we can then define the cluster threat, $\mathbb{T}$, using R^*. Specifically, as Equation (5) shows that the cluster threat is modeled as the total asset value that could potentially be affected by attack propagation from the attacked Pod to all other Pods which are either directly connected or co-located with the attacked Pod.

$$\mathbb{T} = \sum_{p:\ (p_{att}, p) \in R^*} AV_p \tag{5}$$

2. Delay. The total migration delay $\mathbb{D}$ is expressed as:

$$\mathbb{D} = \sum_{p}(1 - \sum_{n} x_p^n y_p^n) \times MD_{pn} \times AV_p \tag{6}$$

MD_{pn} is the delay of migrating $Pod\ p$ to Node n, and when Pod p is not migrated (i.e., $x_p^n = y_p^n$), $(1 - \sum_n x_p^n y_p^n) = 0$, and hence, according to Equation (6), Pod p does not add any delay. Moreover, we also weigh the delay of each Pod with its asset value to express that the migration delay has more impact on Pods with higher asset values (likely critical) than on Pods with lower asset values (less critical).

Therefore, we can formulate a multi-objective optimization problem to minimize both the cluster threat ($\mathbb{T}$) and delay ($\mathbb{D}$). Alternatively, we can also combine the two objectives through a weighted sum, with α and β as the weighting factors that are used to adjust the relative importance of each objective (with $\alpha + \beta = 1$). Therefore, if minimizing the threat is more important than maintaining low latency, then α should be larger than β, and vice versa. In practice, α and β can be adjusted as per each tenant's requirements. Hence, our optimization problem can be formally defined as follows.

Definition 1. *Given a set of Nodes N, a set of Pods P, and a Pod $p_{att} \in P$ under attack, minimize $(\mathbb{T}, \mathbb{D})$ or minimize $(\alpha\mathbb{T} + \beta\mathbb{D})$ under given constraints.*

Theorem 1. *Our problem in Definition 1 is NP-hard.*

Proof. We reduce the well known NP-hard Minimum Dominating Set problem [50] (i.e., finding a subset of nodes in a graph such that every node in the graph is either in the subset or adjacent to another node in the subset) to our problem. Consider any instance $G = (V, E)$ of the Minimum Dominating Set problem, where V is the set of vertices and E the set of edges. We construct an instance of our problem as follows. For each $v \in V$, we create a Node that contains two Pods, one with asset value 0, and the other with asset value 1. For each $e \in E$, we assign a migration delay of 0 for migrating a Pod with asset value 0 in either direction of the edge, and a migration delay of 1 for all other migrations.

We also assume the attacked Pod p_{att} is not co-located with, but connected to, all the Pods with asset value 0.

Let $\alpha = 1$ and $\beta = \infty$. To minimize $(\alpha\mathbb{T}+\beta\mathbb{D})$, we must have $\mathbb{D} = 0$, while minimizing $\mathbb{T}$. First, to achieve $\mathbb{D} = 0$, we can only migrate the Pods with asset value 0 along each edge in either direction (since all other migrations have a delay of 1). Second, to minimize $\mathbb{T}$, we need to minimize the total number of Nodes that contain Pods with asset value 0 after migration, since each such Node will incur a threat of 1.

To minimize the total number of Nodes containing Pods with asset value 0 after migration, we need to find a minimum subset of Nodes, such that every Node n we create is either already in this subset (no migration is needed), or adjacent to another Node n' in the subset (the Pod with asset value 0 in n' will be migrated to n, with a delay of 0), which is equivalent to the dominating set. Therefore, finding a solution to this instance of our problem yields a solution to the given instance of the Minimum Dominating Set problem in polynomial time. Since the latter is known to be NP-hard, this concludes the proof. □

4.2.2 Attack Prediction Model Building

This step is to build an attack prediction model based on the historical Falco alerts to predict the attacker's next move. Specifically, WARP first collects and processes Falco alert logs of Pods in a cluster by parsing the alert log entries from different Pods and extracting their `mitre <tactic name>` tag and forms a sequence. Second, WARP learns the predictive model from the sequences of tactics by leveraging Bayesian network [55] for this model where Nodes indicate MITRE ATT&CK tactics, edges indicate their transitions and are labeled with probabilities of transitions.

Example 2. Figure 4.3 shows an example of this model built out of the MITRE tactic parameter in the alerts. Our attack scenario tactics (highlighted in red) start from the *Privilege Escalation*, lead to *Execution* and then use *Persistence*, with a probability of 31% and 25%,

respectively.

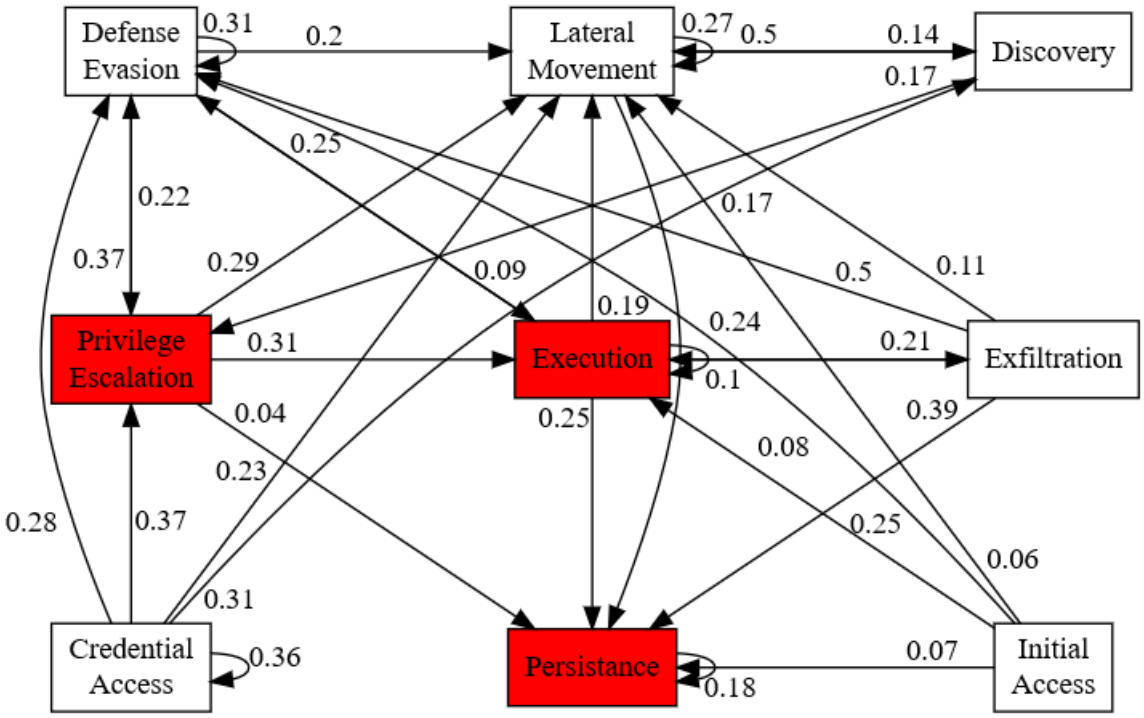


Figure 4.3: Attack prediction model example.

4.3 Runtime Detection and Mitigation

This section presents the steps during our runtime phase.

4.3.1 Proactive Attack Prediction

This step is to predict future attacks based on alerts raised by Falco at runtime. To that end, WARP first applies the previously built attack prediction model to find the attacker's potential next step. Then, it calculates the risk for Pods by examining the alert parameters where it assigns each parameter a value according to the definition given below. If the calculated risk exceeds a predefined threshold, it performs the Pods migration based on optimization model (as in Section 4.3.2).

Example 3. Figure 4.4 shows how the risk associated with `pod_one` is calculated using our risk formula (Equation (7)). The MITRE ATT&CK tactic for the observed alert is `Privilege escalation`. Therefore, according to Figure 4.3, the next likely tactic is `Execution`. Other variables of the risk formula are extracted from alert as shown in Fig. 4.4. Finally, the overall risk for `pod_one` is calculated using the following formula.

$$\begin{aligned} Risk = Priority\_Severity \times MITRE\_Tactic\_Severity \times \\ Context\_Severity \times Next\_MITRE\_Tactic\_Probability \times \\ 2 \times max(NEXT\_MITRE\_Tactic\_Severity) \times Asset\_Value \end{aligned} \tag{7}$$

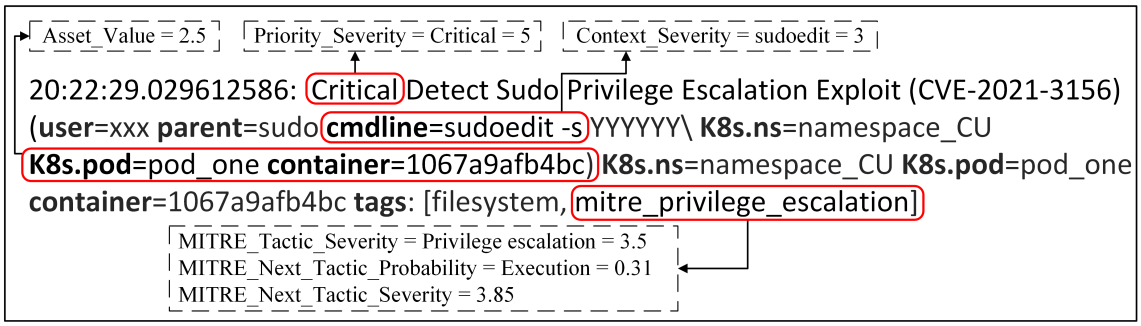


Figure 4.4: Parameters value extracted from a received Falco alert.

In the following, we describe the Equation (7) parameters.

- **Priority Severity** $[1-5]$**:** is enumerated from one to five based on the priority parameter in a Falco alert (i.e., Debug, Notice, Warning, Error, and Critical), respectively.

- **MITRE Tactic Severity** $[1-5]$**:** is the average priority severity of all alerts for one MITRE ATT&CK tactic.

- **Context Severity** $[1-5]$**:** is assigned to alert parameters depending on their predefined malicious level.

- **Next MITRE Tactic Probability** $[0-1]$**:** is the probability of the attacker's next tactic from the prediction model.

- **Asset Value** $[1-5]$**:** is assigned by security admin to each Pod based on the relative importance of hosted services and information.
- **Next MITRE Tactic Severity** $[1-5]$**:** is the *MITRE Tactic Severity* for the next predicted tactic.

4.3.2 Non-disruptive Attack Mitigation

This step is to perform the mitigation when the associated risk for at least one Pod is higher than a certain threshold. WARP optimally mitigates by reducing both the migration delay and overall threat so that with the growing number of migrations, it does not incur prohibitive delay to the users. Specifically, WARP first obtains the current placement information of the entire cluster. Second, it finds the optimal migration plan that leads to a safer cluster state using the previously formulated optimization model and our heuristic algorithm, ***P***roactive ***0P***timization (P0P) as in Algorithms 1 and 2, discussed later. Third, WARP utilizes CRIU [11] to migrate the selected Pods to the selected Nodes, and finally, the cluster transits into a state with reduced threat. In the following, we elaborate on our heuristic optimization algorithm.

4.3.2.1 Objective Optimization with P0P

As the optimization problem formulated in Section 4.2.1 is non-linear in nature, we first explore the potential of using standard heuristics such as genetic algorithms to find near-optimal solutions for the problem. However, our evaluation in Chapter 6 will show, such a standard optimization algorithm not only finds sub-optimal solutions, but also is computationally more expensive. In WARP, the *Optimization* sub-step first applies our optimization model (see Section 4.2.1) and seeks to minimize the threat while minimizing the migration delay using our proposed heuristic algorithms (Algorithms 1 and 2). Next, the *Network Slicing* sub-step ensures the Pods are placed in their corresponding slices. Finally, using

the migrations discovered with Algorithm 1, WARP performs the actual Pod migration from source to destination Node.

The optimal Pods placement problem can be divided into three sub-problems to prevent recomputing the steps (a dynamic programming-based approach [44]) as follows.

1. Threat: isolating the Pods that are either directly connected or co-located with the attacked Pod (p_{att}).

2. Delay: having the maximum delay tolerance in the each tenant's SLA and each Pod migration delay (MD_p), find the maximum possible number of Pods allowed to be migrated.

3. Capacity: having each Node size ($Capacity_n$), and Pod size ($Size_p$), find the maximum possible number of Pods to be placed in each Node.

Algorithm 1 solves the *Threat*, and *Delay* sub-problems. Using a breadth-first search algorithm (BFS), it finds all the Pods connected to the attacked Pod(s) (both direct connection, and co-location), defined as $HotPods$ (Line 1). For the *Delay* sub-problem, it finds the maximum number of migrations (Line 2). In Lines 3 and 4, it finds the $OptimalNode$ for migration destination. This Node has the minimum asset value of $Citizens$ (i.e., Pods that do not belong to $HotPods$). The $OptimalNode$ is the best destination as it hosts less important $Citizens$ in case a $HotPod$ migrates there. In Lines 5-7, as long as we do not exceed the delay constraint, $HotPods$ are migrated to the $OptimalNode$, and placed in their slice. Eventually, if we are left with $Migrations$, we attempt to evict $Citizens$ to another Node and thus achieve more isolation for the $HotPods$. The complexity of BFS is $O(Pods + Connections)$, and accordingly P0P complexity is linear.

Algorithm 2 is called right before the migration to solve the *Capacity* sub-problem. It stores the result of a maximum number of Pods per Node in $L_{NodeCapacity}$ (Lines 1 and 2). The preservation of the result in this list prevents the recursive computation, and reduces

the algorithm complexity. Lines 5 and 6 are to meet the $MaxCapacity$ and $MaxDelay$ constraints. Lines 7 and 8 perform the migration to the $Destination$ Node using CRIU, and return the delay. In case the delay exceeds the delay constraint, the algorithm stops, and in case a capacity constraint is exceeded, the next available Node from $L_{NodeCapacity}$ is selected as destination.

Algorithm 1 $P0P$

1: **Input:** G: Cluster graph, $MaxDelay$, L_{Nodes}: Nodes list, $L_{AttackedPod}$: Attacked Pods list
2: **Output:** G': Optimized cluster graph, Delay
3: $HotPods = BFS(G, L_{AttackedPod})$
4: $Migrations = Max\_Num\_Migration(MaxDelay)$
5: $OptimalNode = Find\_Optimal\_Node(G)$
6: $Citizens = Find\_NonHot\_Pod(OptimalNode)$ ▷ List of Pods to be potentially compromised sorted by their asset value
7: **while** $(Migrations > 0$ **and** $Len(HotPods) > 0)$ **do**
8: $Delay \mathrel{+}= Migrate(G, HotPods.pop(0), OptimalNode)$
9: $Adjust\_Network\_Slice()$
10: **end while**
11: ▷ Number of HotPods is less than delay tolerance ▷ Citizen Pods eviction to other Nodes due to threat
12: **if** $Migrations > 0$ **then**
13: **while** $Migrations > 0$ **and** $Len(Citizens) > 0$ **do**
14: $Delay \mathrel{+}= Migrate(G, Citizens.pop(0), L_{Nodes} - OptimalNode)$
15: $Adjust\_Network\_Slice()$
16: **end while**
17: **end if**
18: **Return G', Delay**

4.3.2.2 WARP with Network Slicing

The main goal of network slicing is to provide tenants with a proactive security solution that can be customized based on their requirements (i.e., security and delay). Network slicing can be leveraged to have many logical networks (slices) over the cluster through three different methods as follows.

No Network Slicing. In the first method, all services are deployed in the Kubernetes cluster

Algorithm 2 $Migrate(G, Pod, Destination)$

for $Node\ in\ L_{Nodes}$ **do**
    $L_{NodeCapacity} = Max\_Num\_Pods(Node)$
end for
while $Len(L_{NodeCapacity}) > 0$ **do**
    if $Check\_Capacity(Destination)$ **then**
        if $Check\_Delay(Pod)$ **then**
            $MigDelay = CRIU(Pod, Destination)$
            $Adjust\_Network\_Slice()$
        else
            $Return:\ 0$ ▷ Beyond $MaxDelay$
        end if
    else
        $Destination = L_{NodeCapacity}.pop(0)$ ▷ Next available Node
    end if
end while
Return MigDelay

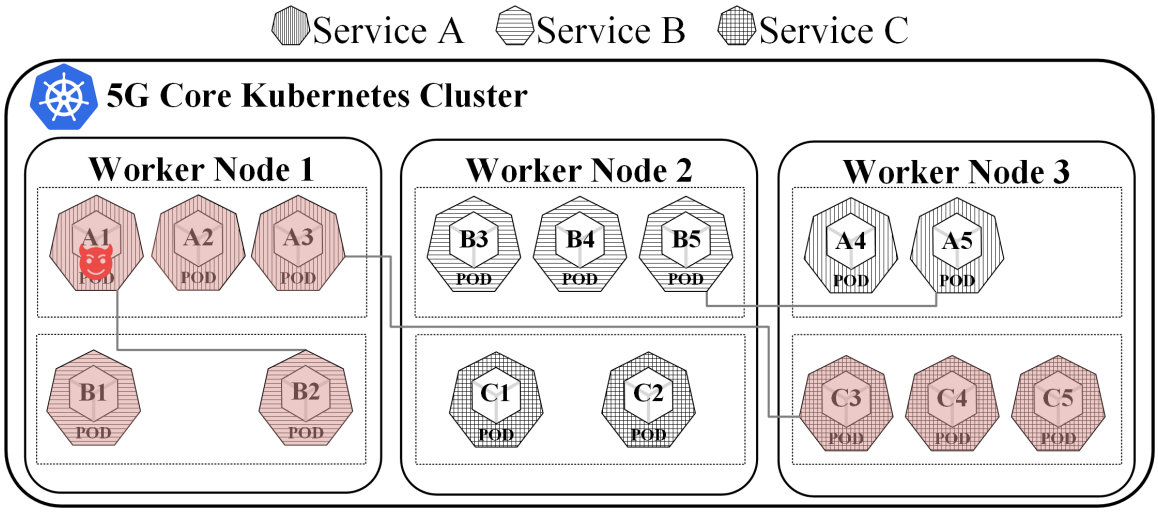
without being logically separated (i.e., no network slicing). Although there is much less network complexity, it is not possible to differentiate between different tenants services. As a result, WARP is obliged to set one global threshold for the whole cluster, which eventually sacrifices some services requirements over the others.

Network Slicing - Type one. In the second method, there is one slice per Node hosting services with similar requirements. Although this approach can take advantage of WARP in terms of different thresholds per slice, it has several disadvantages. First, in the case of service scaling, a Node is fully devoted per slice, which is not efficient in terms of resource utilization. Second, deploying Pods with similar services in one Node, exposes them to a single point of failure (SPOF) risk. Third, this method might interfere with our optimization objectives. For instance, if WARP decides to migrate $Pod\ p$ to $Node\ n$, and $Node\ n$ does not host its slice, the migration plan must be modified.

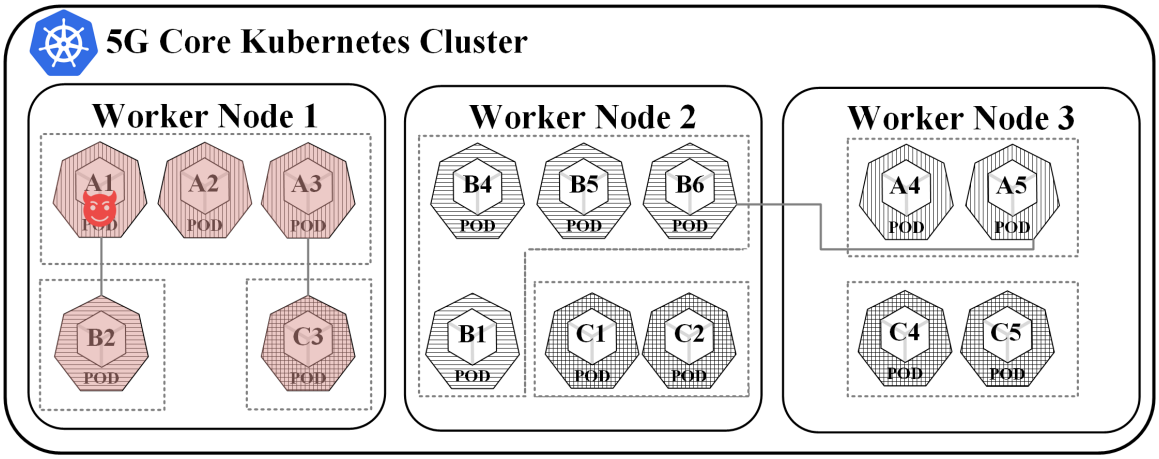
Network Slicing - Type two. In the third method, all slices are present in each Node, to overcome the previous method's disadvantages. Therefore, not only it does not interfere with P0P's optimization objectives, but also it distributes the Pods over multiple Nodes,

using the resources efficiently, and resulting in less risk of SPOF. Example 4 depicts the P0P logic.

Example 4. Figure 4.5 depicts an example of P0P logic and illustrates its adjustability to the tenants' requirements with three services (`A, B, C`) deployed in their slices in the cluster. As shown in Figure 4.5a, `A1` is identified as the `AttackedPod` during the *Proactive Attack Prediction* step. The `HotPods` are either co-located or directly connected with `A1`. Therefore, services `B` and `C` are included in the cluster threat as well. Based on Algorithm 1, `HotPods` are: `A2`, and `A3` due to co-location and `B2`, and `C3` due to direct connection with `A1`. The `OptimalNode` is `Node 1`, because it has less asset value compared to other Nodes. `B1` Pod is a `Citizen` as a non-hot Pod. The goal is to isolate the `AttackedPod` and the `HotPods` in the `OptimalNode`, and evict the `Citizens`, as long as there are available migration moves. Therefore, `C3` is migrated to Node 1, and `Citizen B1` is migrated to an available Node (e.g., Node 2) in its own slice.



(a) Before: attacker initial step.



(b) After: WARP mitigation.

Figure 4.5: WARP and network slicing.

In the second part of this example, we elaborate on the advantage of network slicing for tenants' requirements adjustability. Table 4.2 compares the three slicing methods for the three aforementioned services. Suppose services `A, B, and C` require 80, 70, and 60 (%) security, and accept up to 2, 1, and 0.5 (seconds) delay, respectively, according to the tenants' SLA specifications. As a result, in case of not utilizing network slicing, either

security or delay is sacrificed for some services over others because of selecting a global threshold (e.g., 40%). However, both network slicing types (Type one and Type two) can adjust with each service requirement.

Table 4.2: WARP in 5G - example with three services.

Approaches	**WARP Threshold per Service**			**Resource Utilization (# of Nodes)**
	A	**B**	**C**	
No Network Slicing	30%	30%	30%	3
Network Slicing- type one	27%	42%	53%	>3
Network Slicing- type two	27%	42%	53%	3

4.3.2.3 Mitigation

This step is to perform mitigation by using the optimization result as a migration plan (i.e., list of Pod(s) to be migrated to the selected destination(s)). There are several ways for Pods migration, and we detail three of them as follows.

Kubernetes Rescheduling and Docker. Utilizing Kubernetes rescheduling for Pod migration poses the major limitation due to the lack of control over the destination Node selection, which is in contrast to P0P objective optimization [10]. Although, Pods' anti-affinity can be used to modify the destination Node, it requires `yaml` file modification at runtime (e.g., specifying Nodes for the Pods through labels). Since, any changes to the `yaml` file requires Pod redeployment, this method is not efficient for our purpose. Docker has a similar but more efficient mechanism as Kubernetes, by stopping the Pod, pushing into a local repository and recreating it in a new Node. However, its incurred delay might not be desirable to the end user.

CRIU. CRIU [11] is a checkpoint and restore tool that can be used to migrate containers

by saving the latest checkpoint of the container's state in the disk and restoring them in a new Pod at the destination Node. CRIU keeps the network connection states inside the containers to meet the non-disruptive migration principle. In WARP, CRIU checkpoints the latest state of the container prior to the Falco alert and ensures that the state is prior to the attacker's presence. After saving checkpoint, the Pod is immediately deleted (to minimize the attacker time window), and restored with its checkpoint in the destination Node. During the restoring process (i.e., migration), a delay is experienced by the end user which is discussed in Section 6.1. We overlook the checkpoint transfer time to the disk, as several transfer methods (e.g., SSH/SCP, FTP, rsync [30]) exist with various performances.

Chapter 5

Implementation

5.1 Implementing and Integrating WARP with Kubernetes

WARP is implemented and integrated with Kubernetes following the architecture shown in Figure 5.1. Specifically, WARP is implemented in Python 3.8 and integrated with Kubernetes v1.20.2. The physical infrastructure is composed of one physical rack-mounted server with 2x Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz and 128GB of DDR4-2933 running Debian 10. Our Kubernetes cluster is hosted over 11 VMs (one Master Node and ten Worker Nodes) running Lubuntu 20.04, with VirtualBox 6.1 as the hypervisor. For alert collection, we deploy Falco in our Kubernetes cluster using its official Helm deployment [19]. The prediction model (Bayesian network) is implemented using the *pgmpy* library. At runtime, we leverage CRIU v0.27.0 for mitigation (i.e., Pods migration). For network slicing, the *netaddr* Python library is used.

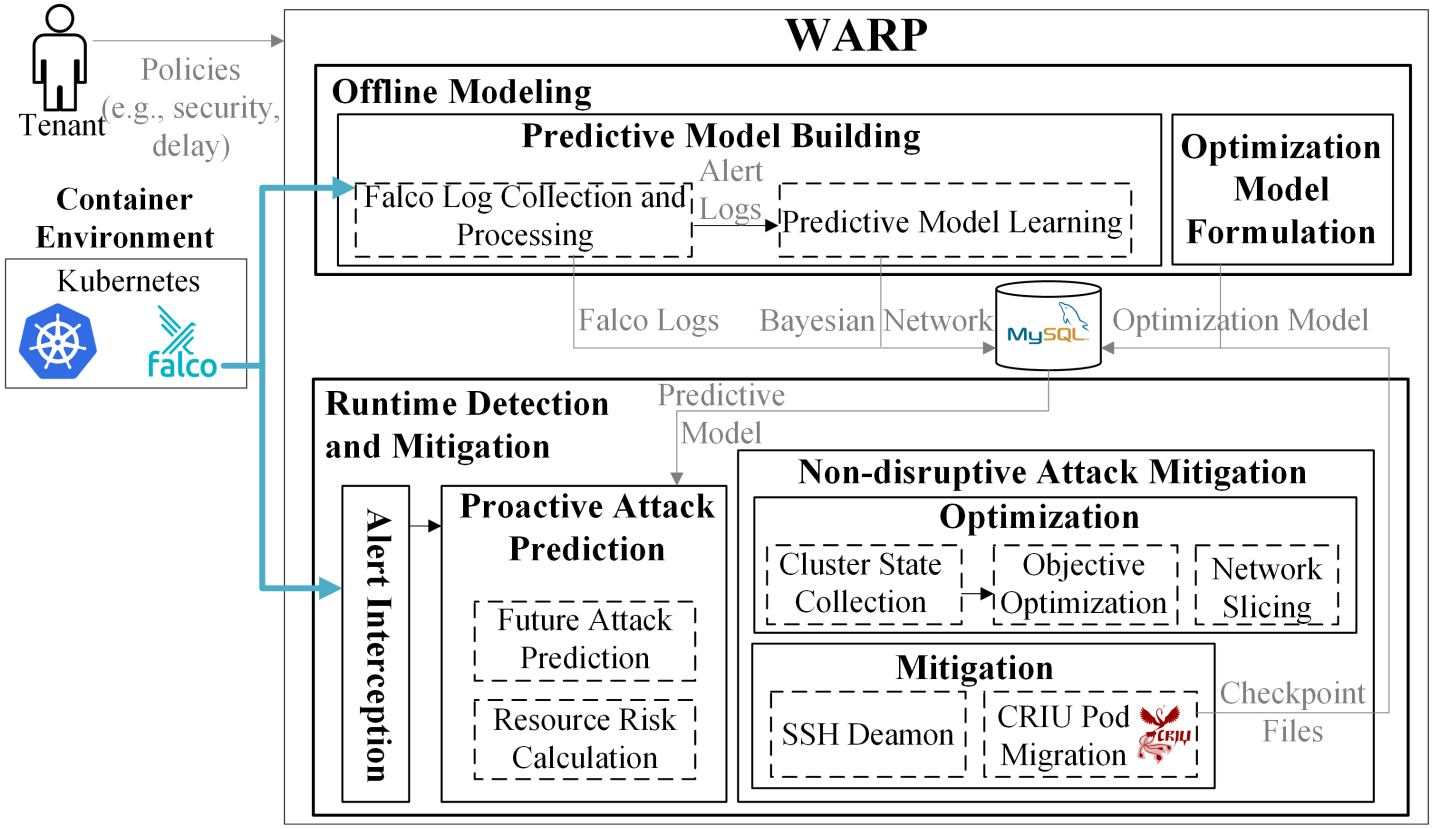


Figure 5.1: WARP architecture.

5.2 Auto-scaling WARP

Kubernetes supports automatic horizontal scaling, which can dynamically adjust the cluster to the tenants' service requirements by increasing or decreasing the number of Pods/Nodes in the cluster. Similarly, we implement WARP in such a way that it can also scale horizontally using Kubernetes DaemonSet [14]. Specifically, new WARP agents will be automatically created upon Nodes scaling up, and those agents will communicate with the main WARP agent (deployed in the Master Node) to obtain its configuration regarding network slices (i.e., threshold values, as detailed below). In case of Node termination due to services scaling down, Kubernetes will take care of gracefully terminating the Pods containing WARP agents.

5.3 Portability to Other Cloud Platforms

WARP can potentially be adapted to other major cloud platforms, such as Amazon Elastic Container Service (ECS) [2] and Microsoft Azure [22]. The components of WARP that may depend on the platform include the runtime monitoring tool, the mapping between alerts and MITRE tactics, and the migration. First, different monitoring tools (e.g., Falco [17], Sysdig Secure [34], and Prometheus [28]) may be leveraged for different cloud platforms. Also, there are different dedicated plugins to handle the MITRE ATT&CK framework in different cloud platforms (e.g., in *Azure Platform Logs* [24] and *AWS* [5]). Second, although the optimization step is platform-independent, the migration step of WARP will depend on specific migration techniques used in the cloud platform. Finally, WARP is independent of the container orchestrator and can thus work for other orchestration systems such as Docker Swarm [15] or OpenShift [26].

Table 5.1: Overview of simulated APT attacks and exploits for WARP dataset.

Attack ID	**Attack Campaign**	**CVE Number**	**Attack Features**[a]					**MITRE ATT&CK Tactic Sequence**[b]
			PL	PA	INJ	IG	BD	
1	APT 3 [4]	2015-3113	✓	✓	✓	✓	✓	Exe, DE, Dis, DE, LM
2	Spam campaign [7]	2017-11882		✓	✓	✓	✓	Dis, Per, Exe, DE, DE, LM, Exf
3	APT 29 [3]	2021-36934	✓	✓	✓	✓	✓	Per, Exe, DE, PE, DE, Dis, LM, IA, Per, PE, DE
4	Escape attack [16]	2021-3156				✓		PE, Exe, Per
5	Simulated cryptominer spread [12]	2017-10271	✓		✓	✓	✓	Dis, Exe, Per, DE, LM
6	Root data theft [29]	2020-14386			✓	✓	✓	Dis, Per, PE, Exf, Per, LM
7	SWC [32]	2015-5122	✓		✓	✓	✓	Dis, Exe, DE, Per
8	Targeted gov phishing [27]	2015-5119	✓		✓	✓	✓	Dis, Per, LM, Exf

[a]PL: Phishing email link. PA: Phishing email attachment. INJ: Injection. IG: Information gathering. BD: Backdoor.
[b]Exe: Execution, DE: Defense Evasion, Dis: Discovery, LM: Lateral Movement, Per: Persistence, PE: Privilege Escalation, IA: Initial Access, Exf: Exfiltration

5.4 Building Dataset

To facilitate learning our prediction model (see Section 4.2.2) and to support future research, we build a relatively large dataset of Falco alerts for Kubernetes, which is publicly available on GitHub [18]. Our dataset includes the alert samples of both normal activities and (APT) attacks. For normal activities, we rely on the fact that Falco generates normal daily routine alerts even in the absence of any attack, therefore we label these samples as "normal". For the attack alerts, we leverage CALDERA [6], an adversary emulation platform developed by MITRE, to mimic attacks in a Kubernetes cluster. Our dataset contains 231k alerts (including 2,314 attack alerts and 228,686 normal alerts). Table 5.1 provides more details of those attacks including the attack feature(s) they follow and the MITRE ATT&CK tactic sequences collected and extracted from the alerts.

Challenges. During building this dataset, we encountered several challenges as follows. First, Falco alerts may be reported at a high rate from many unrelated resources in a Kubernetes cluster. To identify correct alert sequences and reconstruct the attack steps, we wrote scripts to automatically aggregate the alert by resources (e.g., using the container IDs) and then extract the MITRE ATT&CK tactics' property from the sequence of alerts on each container. Second, the original dataset we obtained is imbalanced with a significantly higher number of normal alerts than attack ones, as Falco tends to generate a considerable number of alerts for normal system events. To obtain a realistically balanced dataset [37] for our experiments, we undersample the normal alerts by filtering out normal alerts that share more than 80% similarities, and oversample the attack alerts by duplicating attack alerts (since different attacks may share similar tactics regardless of the exploited vulnerability or the executed payload [37]).

Chapter 6

Evaluation

This section evaluates the effectiveness of WARP. In particular, we investigate the following research questions (RQ):

RQ1. What is the cost of Pod migration?

RQ2. How effective is our optimization algorithm (P0P)?

RQ3. How effective is WARP for mitigating attacks?

RQ4. How much overhead does WARP incur?

RQ5. What is the impact of network slicing on WARP?

6.1 Migration Cost

To answer RQ1, we measure the cost of migration.

Migration Approaches Comparison. As discussed in Section 4.3.2.3, migrating a Pod is performed by migrating its container(s). In this set of experiments, we measure the delay caused by three popular migration methods (i.e., CRIU, Kubernetes rescheduling, and Docker) for containers of different sizes. Among the three methods shown in Figure 6.1a,

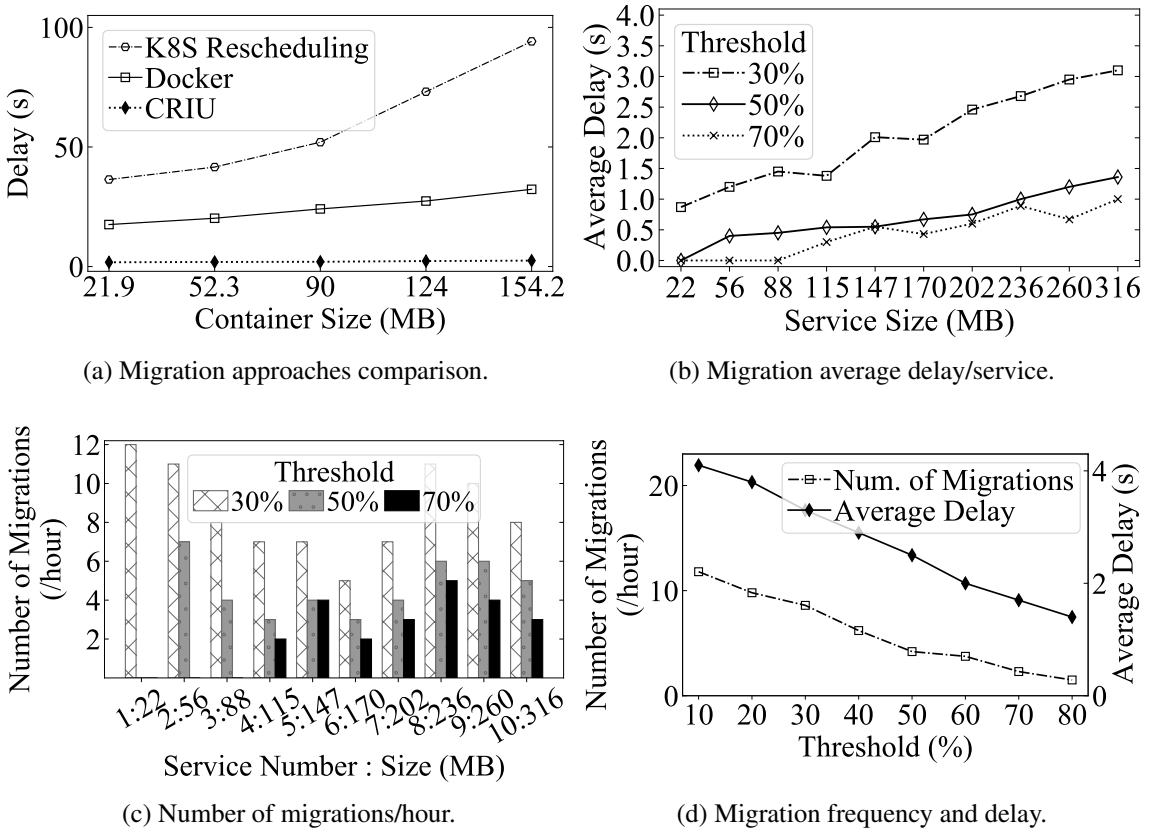
we find that CRIU has the best performance (i.e., the lowest delay) as it stores the checkpoints directly in memory. Moreover, unlike other methods, it does not need to redeploy the Pods from `yaml` files (cf. Kubernetes rescheduling), or fetch the Pods image from repositories (cf. Docker) [11].

Migration Delay on Services. Figure 6.1b depicts the impact of service size over the migration delay for ten services (i.e., *Services 1-10*) with different sizes of Pods (from 22 MB to 316 MB). Even under a threshold of 30% (a smaller threshold means more frequent migration), the average delay is no more than 3.1 (seconds). In addition, Figure 6.1c measures the number of migrations/hour for those ten different services under different thresholds. As expected, a higher threshold value triggers less frequent migrations and hence less delay to the services (e.g., *Services 1*, *2*, and 3 under 70% threshold). Finally, Figure 6.1d shows both the number of migrations per hour and the average delay under various threshold values, which both decrease under larger values of thresholds. Although there is an inherent trade-off between mitigation effectiveness and migration delay, the impact on services is generally negligible and, more importantly, non-disruptive.

6.2 Optimization Effectiveness

To answer RQ2, we compare our P0P heuristic algorithm with a standard optimization solution (genetic algorithm (GA) [42]).

To evaluate the effectiveness of those two algorithms, we measure the cluster threat reduction (Figure 6.2a) and the number of migrations (Figure 6.2b) for 0.5% and 1% attack data under different cluster sizes. As shown in those results, P0P achieves a significant reduction in cluster threat, with an average decrease of 95% and 90.3% for 0.5% and 1% attack data, respectively. On the other hand, GA exhibits a reduction in cluster threat by 76.3% and 70.8%, respectively.



(a) Migration approaches comparison.

(b) Migration average delay/service.

(c) Number of migrations/hour.

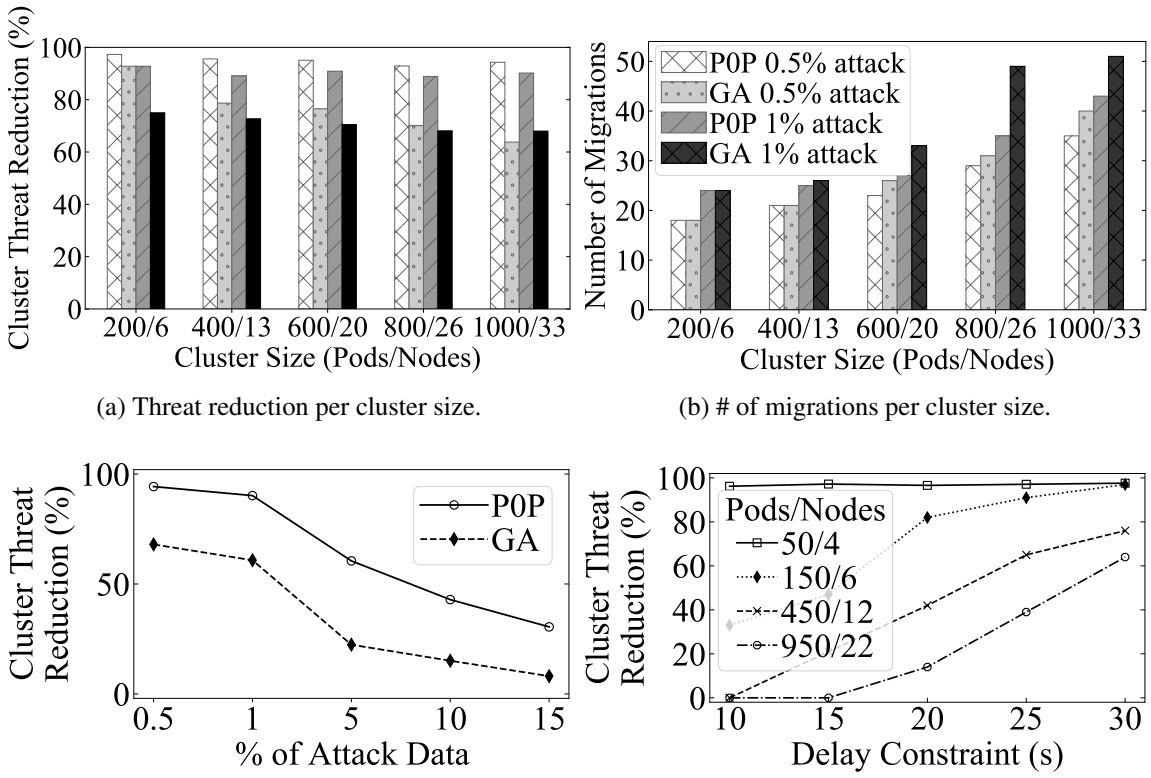
(d) Migration frequency and delay.

Figure 6.1: Migration approaches comparison and cost analysis.

We also compare the number of migrations required by P0P and GA for reducing the cluster threat (Figure 6.2b). P0P achieves more threat reduction while migrating less Pods under all sizes of clusters. Note the total number of migrations is reasonable considering the fact that the average migration delay per Pod is only about 0.87 (seconds) using CRIU (Figure 6.1a).

For further evaluation, we perform a stress test on both P0P and GA to evaluate their effectiveness in reducing the cluster threat under different percentages of attack data for a large cluster of 1,000 Pods in 33 Nodes (Figure 6.2c) Overall, P0P performs significantly better than GA (i.e., 20% more threat reduction on average). Under larger percentages of attack data, both algorithms struggle to minimize threat as more Pods are exposed to attacks; however, P0P still outperforms GA.

In addition, we consider the case where tenants specify a constraint on the total delay (i.e., the delay accumulated over all migrations, see Section 4.2.1). Figure 6.2d shows the results of employing P0P to reduce the cluster threat while satisfying delay constraints. The results demonstrate that, for smaller clusters, a stricter delay constraint does not make a significant difference in threat reduction. However, when the size of the cluster grows, more relaxed delay constraints become necessary, as more Pods might need to be migrated in a larger cluster (note the delay per Pod, which is what tenants would experience, remains negligible, e.g., 0.87 (seconds) using CRIU).



(a) Threat reduction per cluster size.

(b) # of migrations per cluster size.

(c) Attack data impact (1000 Pods/33 Nodes).

(d) The delay/threat trade-off.

Figure 6.2: Cluster threat reduction and delay comparison ((a), (b), (c): no delay constraint).

6.3 WARP Effectiveness

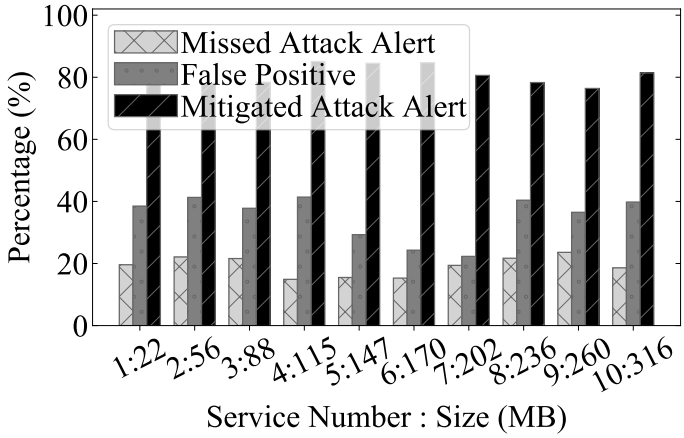
To answer RQ3, we evaluate the effectiveness of WARP in attack mitigation in terms of mitigating attack alerts (i.e., true positives), non-attack alerts (i.e., false positives), and missed attack alerts (i.e., false negatives). Table 6.1 shows the numerical results for six of the simulated attacks.

Table 6.1: WARP effectiveness per attack and dataset.

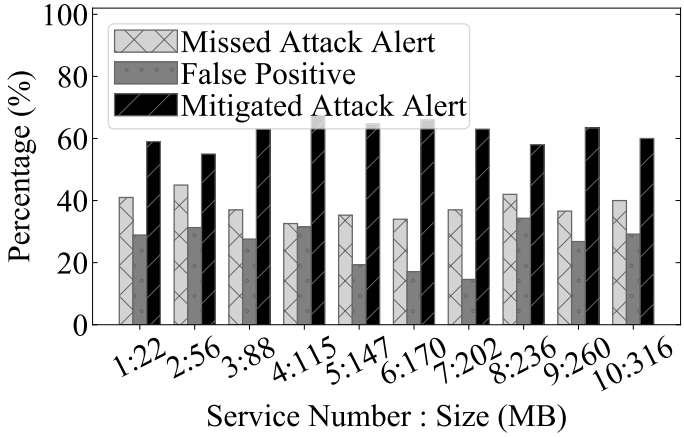
Threshold		**Attack ID**						**Average per Attack (%)**	**Total Dataset (%)**
		1	**2**	**3**	**4**	**5**	**6**		
30%	**Mitigated**	60	91	68	91	95	89	81	81
50%	**Attack**	0	85	40	77	84	79	61	61.95
70%	**Alerts (%)**	0	68	30	44	67	65	45.6	42.54
30%	**False**	32	39	39	42	77	38	39	35.1
50%	**Positive(%)**	8	34	29	34	35	33	28	26
70%		8	24	18	23	31	28	22	18.29

For a lower threshold value (e.g., 30%), WARP is more aggressive and hence results in a higher mitigated attack alert rate (a lower missed attack alert rate) and a higher false positive rate (Fig. 6.3). Therefore, adopting such a lower threshold is generally preferable, especially considering that the false positives have less impact under WARP due to the non-disruptive nature of migration (i.e., they only result in a slightly increased delay but no disruption to the services).

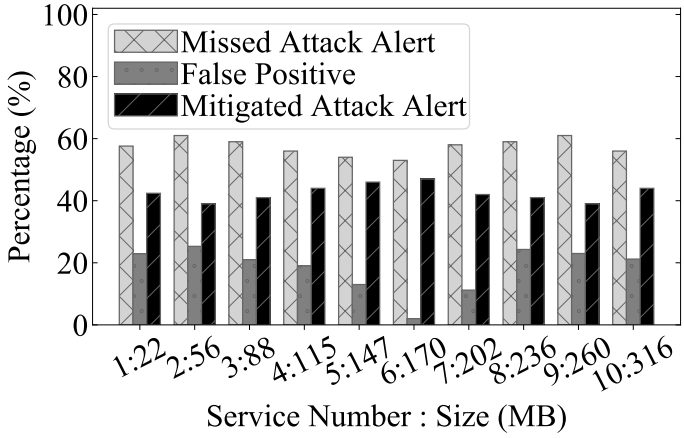
Attack Progress. We conducted another set of experiments (as shown in Fig. 6.4) to evaluates the effectiveness of WARP for mitigating three sample attacks (i.e., SWC, APT 3, and APT 29) under different threshold values (reflected by the number of received Falco alerts). The risk values are calculated using the method described in Fig. 4.4 upon receiving each Falco alert. For instance, APT 29 is completed with 10 alerts received, and the associated risk value stays in the range of *very low* during the first three alerts and reaches to *high* after



(a) Threshold = 30%



(b) Threshold = 50%



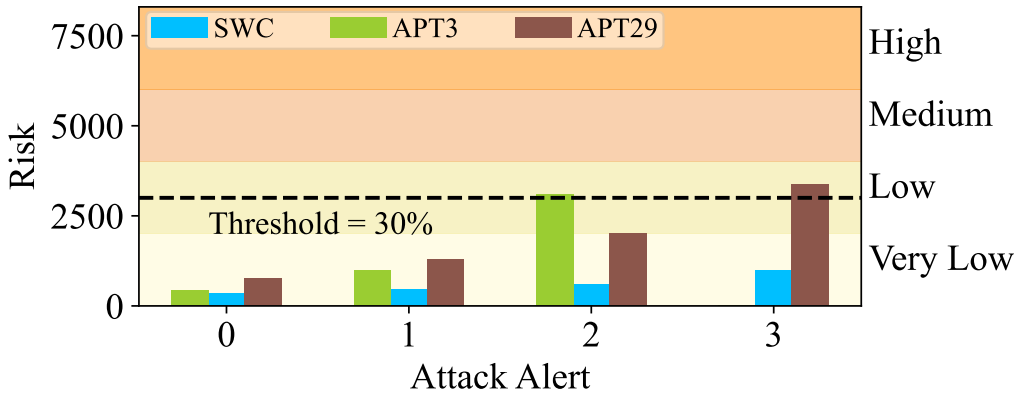
(c) Threshold = 70%

Figure 6.3: WARP effectiveness.

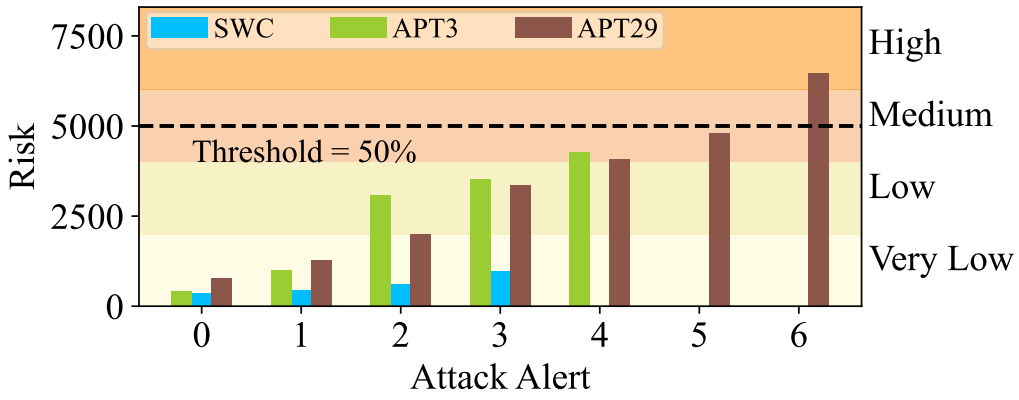
Alert 7. As the figure shows, for different thresholds (30%, 50%, and 70%), indicated by horizontal dashed line, WARP will mitigate the attacks at different risk levels. A threshold of 30% (Fig. 6.4a) can mitigate all the attacks and prevent potential damages to the cluster. A higher threshold of 50% (shown in Fig. 6.4b) will mitigate APT 29, while allowing APT 3 and SWC to successfully complete without being mitigated. Finally, a threshold of 70% (shown in Fig. 6.4c) will only mitigate APT 29 at the *high* risk level, and the other attacks will complete successfully. The implication of those results is that a security admin could generally choose a lower threshold to mitigate attacks more effectively (although it also implies a slightly higher migration delay, as shown in the following experiments).

6.4 Performance

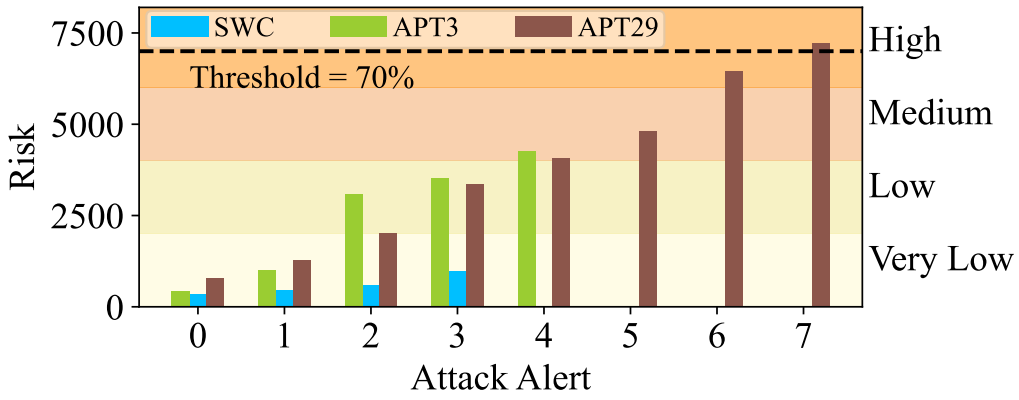
To answer RQ4, we perform two experiments to evaluate the execution time and CPU consumption. Figure 6.5a shows the execution time for various cluster sizes, considering 0.5% and 1% attack data. The results demonstrate that P0P surpasses GA in terms of performance. Specifically, P0P makes its decision in less than a second even for a relatively large cluster. Figure 6.5b shows the CPU consumption of WARP running under three different thresholds. A lower threshold triggers more frequent migrations resulting in higher CPU usage on average (spikes in the graph). However, even under a lower threshold (e.g., 30%), the CPU consumption increases only by 20% compared to the cluster's normal CPU usage.



(a) Threshold = 30%

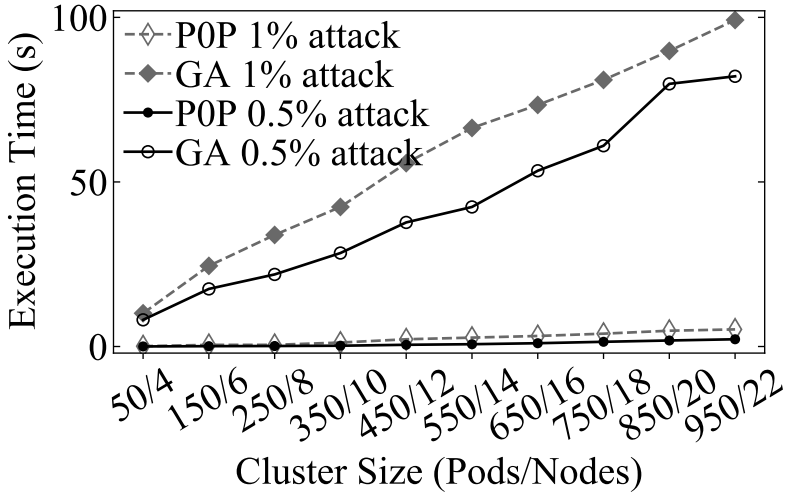


(b) Threshold = 50%

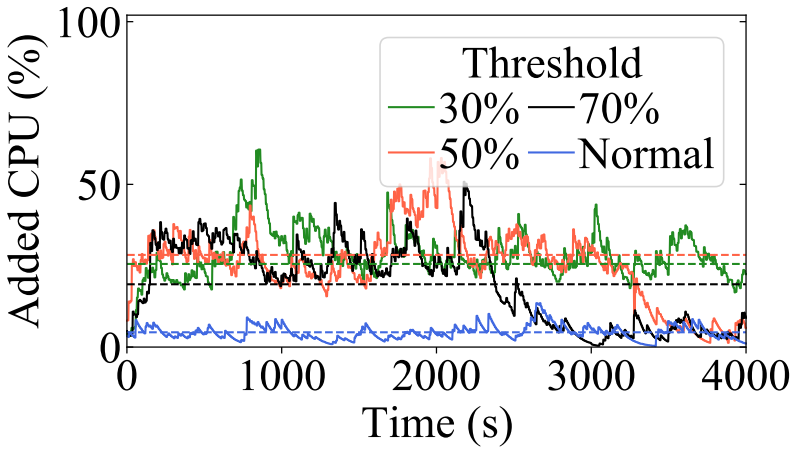


(c) Threshold = 70%

Figure 6.4: Attack progress.



(a) P0P Execution time.



(b) WARP CPU consumption.

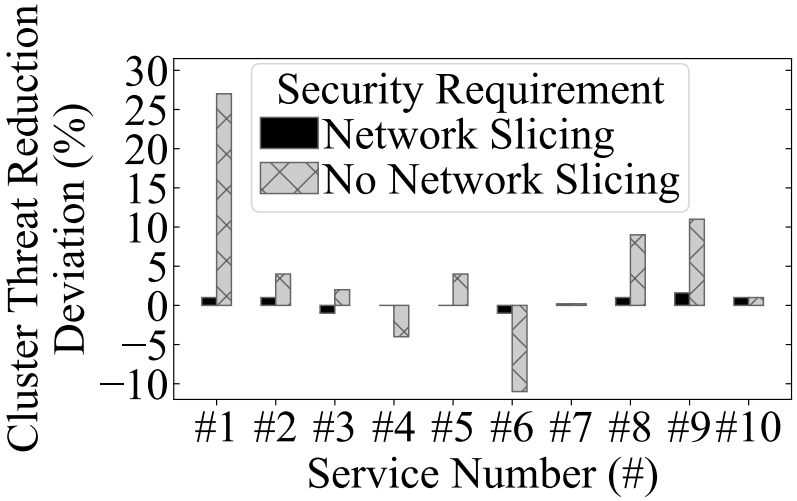
Figure 6.5: Performance overhead.

6.5 Adjustability to Tenants' Requirements

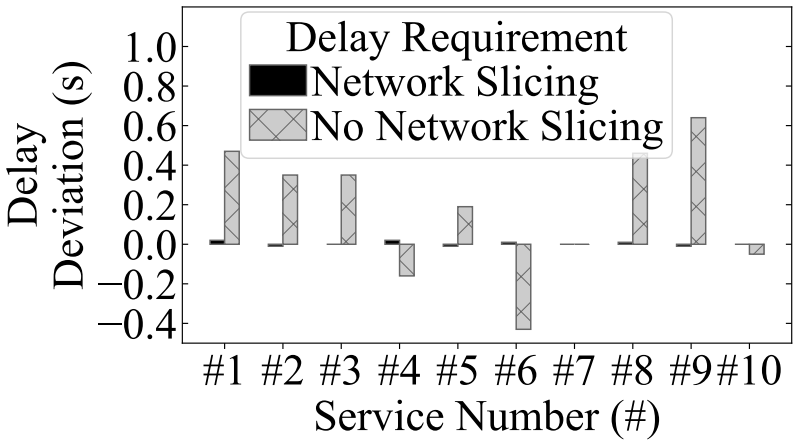
To answer RQ5, this experiment studies how network slicing can meet different services' requirements, and how WARP can be leveraged as an adjustable security solution.

Figure 6.6 shows both delay and threat reduction deviation from the actual requirements for each service with and without the network slicing. When network slicing is not

considered, one global threshold is set for all services, and therefore WARP may sacrifices requirements (i.e., providing more or less security/delay than what each service require). On the other hand, when network slicing is considered, each slice has its own threshold value instead of a global one. Hence, each WARP agent (residing in a Node) performs the migration for threat reduction based on its own threshold value. Subsequently, WARP is able to fully meet the different requirements of services, as demonstrated by a close-to-zero deviation from the tenant requirement. adjustable security solution.



(a) Threat reduction.



(b) Imposed delay.

Figure 6.6: Effect of network slicing.

Chapter 7

Conclusion

We presented WARP, a cost-effective approach for proactive non-disruptive attack mitigation in Kubernetes clusters. We proposed a prediction model built out of MITRE ATT&CK tactics, and utilized it to identify resources under the risk of attack propagation. We then designed a non-disruptive migration approach to optimally reduce the cluster threat with minimum cost. To derive the most cost-effective migration plan, we proposed an efficient heuristic optimization algorithm. Finally, we leveraged network slicing to support different tenant requirements within the same cluster.

7.1 Limitations and Future Work

First, we utilize Falco as a rule-based threat detection tool, which lacks the capabilities to detect zero-day attacks or attacks that mimic normal behavior. A future work is to leverage other attack detection methods to cover more attacks. Second, our prediction model is learned from a list of manually simulated attack alerts, and therefore expanding the scope of our model is another future direction. Third, WARP only employs MITRE ATT&CK tactics for attack prediction, and a future work is to explore alert text processing along with MITRE ATT&CK techniques to derive novel attack indicators. Finally, we also plan

to explore other non-disruptive mitigation methods to complement migration for better coverage.

Bibliography

[1] 5G network slicing. https://www.ericsson.com/en/network-slicing. [Accessed 10-3-2023].

[2] Amazon ECS. Amazon Elastic Container Service (ECS) https://aws.amazon.com/ecs/. [Accessed 25-4-2023].

[3] APT 29. https://attack.mitre.org/groups/G0016/. [Accessed 30-3-2022].

[4] APT 3. https://attack.mitre.org/groups/G0022/. [Accessed 30-3-2022].

[5] AWS Security Stack Mappings. AWS Security Stack Mappings. https://center-for-threat-informed-defense.github.io/security-stack-mappings/AWS/README.html. [Accessed 30-3-2022].

[6] CALDERA. https://caldera.mitre.org/. [Accessed 30-3-2022].

[7] Cedrick Ramos. Spam campaigns with malware exploiting CVE-2017-11882 spread in Australia and Japan. https://www.trendmicro.com/vinfo/us/threat-encyclopedia/spam/3655/spam-campaigns-with-malware-exploiting-cve201711882-spread-in-australia-and-japan/. [Accessed 30-3-2022].

[8] Container images vul. https://sysdig.com/learn-cloud-native/container-security/docker-vulnerability-scanning. [Accessed 4-7-2023].

[9] Containers vs. VM. https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm. [Accessed 4-7-2023].

[10] CORDON/UNCORDON. https://kubernetes.io/docs/tasks/administer-cluster/safely-drain-node/. [Accessed 10-3-2023].

[11] CRIU. https://criu.org. [Accessed 30-3-2022].

[12] Crypto miner delivery. Exploiting CVE-2017-10271. https://www.mandiant.com/resources/cve-2017-10271-used-deliver-cryptominers-overview-techniques-used-post-exploitation-and/. [Accessed 30-3-2022].

[13] CVE-2021-3156. https://nvd.nist.gov/vuln/detail/CVE-2021-3156/. [Accessed 30-3-2022].

[14] DaemonSet | Kuberetes. Daemonset | Kubernetes. https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/. [Accessed 30-3-2022].

[15] Docker Swarm. Docker Swarm https://docs.docker.com/engine/swarm/. [Accessed 25-4-2023].

[16] Escape attack. Exploiting CVE-2021-3156. https://www.helpnetsecurity.com/2021/01/27/cve-2021-3156/. [Accessed 30-3-2022].

[17] Falco. https://falco.org/. [Accessed 30-3-2022].

[18] Falco alert dataset with APT attacks. https://github.com/simabagheri1/Falco-Alerts-Dataset-with-APT-attacks.

[19] Falco Installation Tools. https://falco.org/docs/getting-started/third-party/install-tools/#helm. [Accessed 10-3-2023].

[20] Free5GC. https://www.free5gc.org/. [Accessed 30-3-2022].

[21] Kubernetes. https://kubernetes.io/. [Accessed 30-3-2022].

[22] Microsoft Azure. Microsoft Azure https://azure.microsoft.com/. [Accessed 25-4-2023].

[23] MITRE Att&CK. https://attack.mitre.org/. [Accessed 30-3-2022].

[24] MITRE ATT&CK Azure. https://www.microsoft.com/en-us/security/blog/2021/06/29/mitre-attck-mappings-released-for-built-in-azure-security-controls/. [Accessed 25-4-2023].

[25] Open Policy Agent/Gatekeeper. Open Policy Agent/Gatekeeper https://open-policy-agent.github.io/gatekeeper/. [Accessed 10-3-2023].

[26] OpenShift. OpenShift. https://docs.openshift.com/. [Accessed 30-3-2022].

[27] Pierluigi Paganini. Phishing campaigns target US government agencies exploiting hacking team flaw CVE-2015- 5119. . [Accessed 30-3-2022].

[28] Prometheus. Prometheus https://prometheus.io/. [Accessed 25-4-2023].

[29] Root data theft. Kernel vulnerability exploiting CVE-2020-14386. https://nvd.nist.gov/vuln/detail/CVE-2020-14386/. [Accessed 30-3-2022].

[30] Rsync. https://linux.die.net/man/1/rsync. [Accessed 10-3-2023].

[31] Seccomp security profiles forDocker. https://github.com/docker/docker/blob/master/docs/security/seccomp.md. [Accessed 4-7-2023].

[32] Strategic web compromise. https://www.fireeye.com/blog/threat-research/2015/07/second_adobe_flashz0.html/. [Accessed 30-3-2022].

[33] Sysdig. Sysdig https://sysdig.com/. [Accessed 10-3-2023].

[34] Sysdig SECURE. Sysdig SECURE https://sysdig.com/products/secure/. [Accessed 25-4-2023].

[35] OpenStack Congress, 2015.

[36] CNCF Survey Report. www.cncf.io, 2020. [Accessed 10-10-2022].

[37] A. Alsaheel et al. ATLAS: A sequence-based learning approach for attack investigation. In *USENIX Security*, 2021.

[38] S. Arzo et al. Study of virtual network function placement in 5g cloud radio access network. *IEEE Trans. Netw. Serv.*, 2020.

[39] W. Attaoui et al. Vnf and cnf placement in 5g: Recent advances and future trends. *IEEE Trans. Netw. Serv.*, 2023.

[40] Y. Avrahami and S. Ben Hai. Kubernetes privilege escalation: Container escape == cluster admin? In *Black Hat USA*, 2022.

[41] S. Bagheri et al. Dynamic firewall decomposition and composition in the cloud. *TIFS*, 2020.

[42] W. Banzhaf et al. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., 1998.

[43] F. Bari et al. Orchestrating virtualized network functions. *IEEE Trans. Netw. Serv.*, 2016.

[44] R. E. Bellman. *Dynamic programming*. Princeton university press, 2010.

[45] A. De Domenico et al. Optimal virtual network function deployment for 5g network slicing in a hybrid cloud infrastructure. *TWC*, 2020.

[46] N. DeMarinis et al. Sysfilter: Automated system call filtering for commodity software. In *RAID*, 2020.

[47] Z. Ding et al. Kubernetes-oriented microservice placement with dynamic resource allocation. *IEEE Trans. on Cloud Comput.*, 2022.

[48] Q. Du et al. Anomaly detection and diagnosis for container-based microservices with performance monitoring. In *ICA3PP*, 2018.

[49] S. Ghavamnia et al. Confine: Automated system call policy generation for container attack surface reduction. In *RAID*, 2020.

[50] F. Grandoni. A note on the complexity of minimum dominating set. *J. Discrete Algorithms*, 2006.

[51] X. Han et al. Unicorn: Runtime provenance-based detector for advanced persistent threats. In *NDSS*, 2020.

[52] W. Hassan et al. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *NDSS*, 2019.

[53] W. Hassan et al. OmegaLog: High-fidelity attack investigation via transparent multi-layer log analysis. In *NDSS*, 2020.

[54] W. Hassan et al. Tactical provenance analysis for endpoint detection and response systems. In *IEEE SP*, 2020.

[55] D. Heckerman. A tutorial on learning with bayesian networks, 2021.

[56] M. Kabir et al. Joint routing and scheduling of mobile charging infrastructure for v2v energy transfer. *IEEE Trans. Intell. Veh.*, 2021.

[57] H. Kermabon-Bobinnec et al. Prospec: Proactive security policy enforcement for containers. In *ACM CODASPY*, 2022.

[58] L. Lei et al. Speaker: Split-phase execution of application containers. In *DIMVA*. Springer, 2017.

[59] H. Liu et al. Watermark-based proactive defense strategy design for cyber-physical systems with unknown-but-bounded noises. *IEEE Trans. Autom. Control.*, 2022.

[60] T. Ma et al. A mutation-enabled proactive defense against service-oriented man-in-the-middle attack in kubernetes. *IEEE Trans. Comput.*, 2023.

[61] S. Majumdar et al. Proactive verification of security compliance for clouds through pre-computation: Application to openstack. In *ESORICS*, 2016.

[62] S. Majumdar et al. LeaPS: Learning-based proactive security auditing for clouds. In *ESORICS*. Springer, 2017.

[63] S. Majumdar et al. Learning probabilistic dependencies among events for proactive security auditing in clouds. In *J. Comput. Secur.*, 2019.

[64] S. Majumdar et al. Prosas: Proactive security auditing system for clouds. *TDSC*, 2021.

[65] B. Martini et al. Intent-based network slicing for sdn vertical services with assurance: Context, design and preliminary experiments. *FGCS*, 2023.

[66] S. Milajerdi et al. Holmes: real-time APT detection through correlation of suspicious information flows. In *IEEE SP*, 2019.

[67] I. Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1971.

[68] M. Patnaik et al. Prolemus: A proactive learning-based mac protocol against puea and ssdf attacks in energy constrained cognitive radio networks. *TCCN*, 2019.

[69] B. Tan et al. A cooperative coevolution genetic programming hyper-heuristics approach for on-line resource allocation in container-based clouds. *IEEE Trans. on Cloud Comput.*, 2020.

[70] C.-W. Tien et al. Kubanomaly: anomaly detection for the docker orchestration platform with neural network approaches. *Engineering reports*, 2019.

[71] R. Yang et al. UIScope: Accurate, instrumentation-free, and visible attack investigation for GUI applications. In *NDSS*, 2020.

[72] Z. Zou et al. A docker container anomaly monitoring system based on optimized isolation forest. *IEEE Trans. on Cloud Comput.*, 2019.