ACE-WARP: A Cost-Effective Approach to Proactive and Non-disruptive Incident Response in Kubernetes Clusters

Sima Bagheri*, Hugo Kermabon-Bobinnec*, Mohammad Ekramul Kabir*, Suryadipta Majumdar*, Lingyu Wang*, Yosr Jarraya§, Boubakr Nour§, Makan Pourzandi§
*CIISE, Concordia University, Montreal, Canada. Emails: {sima.bagheri, hugo.kermabonbobinnec, mohammad.kabir, suryadipta.majumdar, lingyu.wang}@concordia.ca
§Ericsson Security Research, Canada. Emails: {yosr.jarraya, boubakr.nour, makan.pourzandi}@ericsson.com

***Abstract*—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. 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. In this paper, we propose ACE-WARP, a cost-effective proactive and non-disruptive incident response 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 approach is also non-disruptive since the mitigation 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 trade-off between security and cost (delay) as a multi-objective optimization problem. Our evaluation results show that ACE-WARP can successfully mitigate up to 81% of the attacks, and our optimization algorithm achieves up to 30% more threat reduction and 7% less delay while being 37 times faster compared to a standard optimization solution.**

***Index Terms*—Cloud-native, Incident Response, Proactive security, Containers, Kubernetes, Attack Mitigation, Optimization, Advanced Persistent Threats (APT)**

I. Introduction

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 [1]) makes it easy to deploy and manage a large-scale 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 [2]. Moreover, popular container images are shown to be buggy with vulnerabilities [3], and even the cluster orchestrator itself may contain vulnerabilities or misconfigurations (e.g., the Kubernetes privilege escalation vulnerability showcased at Black Hat USA 2022 [4]). 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.

There already exist attack prevention solutions for containers and Kubernetes, such as the Open Policy Agent (OPA) and Gatekeeper [5] combination for runtime security policy enforcement, as well as the Seccomp (SECure COMPuting) filter [6], which is for preventing containers from accessing certain host-level resources through blocking system calls. The Seccomp filters are also leveraged in existing works [7], [8], [9] 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. Therefore, attack detection solutions are still necessary. To that end, Falco [10] 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, these solutions do not provide a direct way to stop such attacks.

Unlike attack prevention and detection, which have both received much attention, automated mitigation of detected attacks in Kubernetes clusters has received less attention (a detailed review of related works will be given in Section I-A). 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 [10], and then take corresponding mitigation actions to stop or slow down the attack progress. However, such a traditional approach to attack mitigation may face several major challenges in the specific context of defending Kubernetes clusters (we will further illustrate those limitations through an example in section II-B).

- 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, 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 data 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 other factors through a cost-effective 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). It would be highly challenging for the administrator of a Kubernetes cluster to cater to those different requirements, all through manually adjusting his/her mitigation actions.

In this paper, we propose a novel approach named *ACE-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 level of security and its associated cost, as typically specified in their Service Level Agreements (SLAs).

The main contributions of this paper are as follows:

- We build the first large-scale Kubernetes attack dataset with 231k Falco alerts based on real-world APT attacks simulated in a controlled environment [11].
- 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 [12]. 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 [13] 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 ACE-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.

A preliminary version of this paper [14] introduces the basic idea of non-disruptive proactive attack mitigation in Kubernetes clusters, with the main limitation of lacking a systematic optimization approach. In this paper, we significantly extend our previous work to realize its full potential in achieving cost-effective and customizable attack mitigation. Specifically, our major extensions are as follows: (i) We formulate a new multi-objective optimization problem of Pods placement to formally model the trade-off between security and delay (Section IV-A). (ii) We design and implement a new heuristic algorithm to solve the optimization problem more efficiently (Section V-B1). (iii) We investigate and adopt new state-of-the-art migration mechanisms for better performance (Section V-B3). (iv) We propose a new method for leveraging network slicing to customize attack mitigation based on mixed tenant requirements (Section V-B2). (v) Finally, we provide a new guideline for adapting our solution to other platforms (Section VI-D) and perform new experiments to evaluate the effectiveness, performance, and customizability (Section VII).

The rest of this paper is organized as follows. We provide related work in Section I-A. Section II provides preliminaries. Section III gives an overview of ACE-WARP. Sections IV and V detail the offline and runtime phases, respectively. Sections VI and VII present the implementation and our evaluation results, respectively. Finally, conclusion, discussion, and future work are discussed in Section VIII.

A. Related Work

ACE-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 ACE-WARP to related works in those areas. First, Table I presents a comparison of ACE-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 ACE-WARP covers all the aforementioned features, operates in the Kubernetes environment, and is adjustable with the tenants' requirements.

TABLE I: Comparing ACE-WARP with existing solutions.

Ref.	Method	Features						Environment	Tenant Adjustability
		Proactive	Model Learning	MITRE ATT&CK	Attack Coverage	Mitigation	Non-disruptive		
PROLEMus [15]	MAC protocol	●	●	–	◑	–	N/A	Cognitive Radio Network (CRN)	N/A
ProSAS [16]	Custom Algorithm	●	●	–	●	●	–	Cloud	–
Ma et al. [17]	Custom Algorithm	●	–	–	◑	●	●	Kubernetes	–
Miranda et al. [18]	Custom Algorithm	●	●	–	●	–	●	Wireless Sensor Networks (WSNs)	*N/A*
ProSPEC [19]	Custom Algorithm	●	●	–	●	●	–	Kubernetes	–
Liu et al. [20]	Watermark	●	–	–	◑	–	N/A	Cyber-physical system	N/A
ATLAS [21]	Custom Algorithm	–	●	–	●	–	N/A	Windows	N/A
NoDoze [22]	Custom Algorithm	–	–	–	●	–	N/A	Windows	N/A
UNICORN [23]	Custom Algorithm	●	●	–	◑	–	N/A	Windows/ Linux	N/A
ACE-WARP	Custom Algorithm	●	●	●	●*	●	●	Kubernetes	●

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

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 [24], Falco [10], and OPA/Gatekeeper [5] are runtime attack detection tools designed for containerized environments. The authors in [25] presented *KubAnomaly*, a learning-based anomaly detection approach for security monitoring in Kubernetes. Several solutions [26], [27] have been proposed to analyze the performance data of Kubernetes containers and Pods, and detect anomalies. Unlike those reactive solutions, ACE-WARP provides a proactive attack mitigation solution by predicting and mitigating 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 [19]) and traditional cloud environments ([16], [28], [29], [30]). The authors in [16], [29], [31] propose proactive security policy verification where the mitigation is performed at runtime. ProSPEC [19] 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 [20] propose a watermark-based approach to proactively defend against man-in-the-middle attacks. PROLEMus [15] is a learning-based approach for addressing denial-of-service (DoS) attacks. Authors in [18] integrate intrusion prevention and anomaly detection for wireless sensor networks (WSNs), which can be a complementary work with ACE-WARP. Unlike those existing proactive approaches, ACE-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. ACE-WARP can address any multi-step attacks involving attack propagation inside the cluster (detailed in Section II-C).

Provenance Analysis. Provenance analysis solutions aim to investigate the root cause of detected attacks and do not provide attack mitigation [32], [33], [34]. UNICORN [23] proposes a graph-based technique to investigate contextual information of stealthy APT attack steps without predefined attack signatures. NoDoze [22] focuses on attack triaging using provenance graphs to identify anomalous paths for further manual response. Holmes [35] 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 [21] design a learning-based approach to build a sequence-based model out of a provenance graph to extract the attack story. ACE-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 [36], [37], [38], [39], [40], [41]. Since this problem is generally intractable, heuristics, meta-heuristics, and machine learning algorithms are proposed to solve the problem efficiently. The authors in [36], [42] 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 [39]. CODO [37] 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 [38] 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 [40]. The authors in [41] study the problem of scheduling mobile charging infrastructure for vehicles. The placement optimization to container network functions is studied in [43]. Similarly, the reduction of cluster threat and delay are the main objectives for the Pods migration optimization problem which is formulated in this work in Section IV.

Moving Target Defence (MTD). Moving Target Defense (MTD) consists of applying system reconfiguration (e.g., VM migration, IP shuffling, or redundancy) to dynamically change the attack surface and deceive attackers [44]. The migration-based MTD deployment is an easy-to-apply defense in virtualized environments and it is the core technique of many MTD solutions for cloud computing. Migrations aim to move the attacker's VM to prevent compromising the co-resident VMs. Besides, there are existing works such as [45], [46], [47] that propose to optimally choose the MTD strategy based on the attack scenario. However, as ACE-WARP core defense is Pod migration, we focus on migration-based MTD solutions.

There are three methods of *reactive*, *proactive*, and *hybrid* strategies when it is the time of deploying migration-based MTD solutions. The reactive method is based on a security alert. The proactive method triggers migration based on a fixed or random time intervals. Proactive methods ensure regular migrations even in absence of any attack. The hybrid strategy is a combination of both reactive and proactive methods, where the time interval to perform migration is based on security alerts while the interval is also integrated to prevent potential, undetected security threats (e.g., zero-day attacks) [44]. The major difference of ACE-WARP with proactive MTD approaches is that we predict the attack and proactively mitigate its propagation. However, proactive MTD is just applying migration in advance with no attack indicator or prediction. Therefore, it is fair to compare ACE-WARP with existing proactive MTD approaches in Section VII-G.

II. Preliminaries

This section provides preliminaries.

A. Background

Docker Container. A Docker container is a lightweight, standalone, and executable package that includes everything needed to run a piece of software, including the code, libraries, and settings. Docker Engine is the core technology that allows containers to run on a host system [48].

Kubernetes. Kubernetes [1] is one of the most widely adopted container orchestrators for cloud-native applications [49]. On the left side of Figure 1, the Kubernetes cluster includes one Master Node and two Worker Nodes hosting applications and services within Pods.

Pod. A pod is the smallest deployable unit in Kubernetes. It represents a single instance of a running process in the cluster.A pod can contain one or more containers and it encapsulate an application's container(s) that share resources such as storage volumes, IP addresses, and environment variables. Pods are designed to be ephemeral and If a pod fails or needs to be scaled, Kubernetes easily terminate it and replace it to ensure high availability and reliability of applications running on the cluster. In Figure 1 Pods are hosting some network functions in a 5G Service-Based Architecture (SBA) [50]. Some Pods have explicit connections shown as a direct line between them.

Falco. Our cluster in Figure 1 is also configured with Falco [10], a popular runtime security solution which reports security alerts on suspicious events in the cluster. Specifically, 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 Falco agent on the Master Node.

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) [13]. Each slice is logically isolated and can host services sharing similar requirements [51]. The left side of Figure 1 depicts how those 5G network functions can be separated into multiple slices (illustrated in different colors). In ACE-WARP, we take advantage of network slicing to customize attack mitigation for different tenants.

B. Motivating Example

Figure 1 depicts an attack scenario exploiting a real-world vulnerability [52] in a Kubernetes cluster hosting a 5G core [50].

Attack Scenario. The upper-left corner of Figure 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.

As shown in the upper-left corner of Figure 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 Figure 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 Figure 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

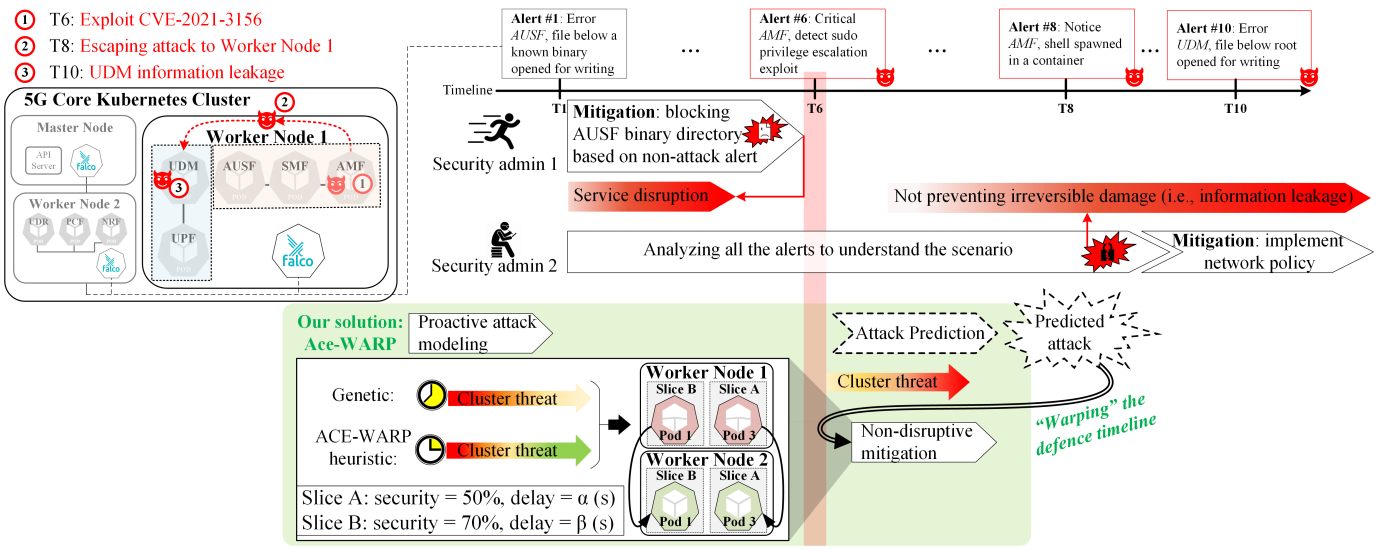


Fig. 1: Motivating example showing a container escaping attack (left) and three potential solutions (right).

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, ACE-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.

C. 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 start with compromising a container (which can be an easy target since container images are known to be buggy), and then escape the container to compromise/take control of node, and subsequently other containers/nodes in the same cluster, causing large-scale damages. Such attacks may be launched by external attackers, a malicious tenant, or insiders through exploiting misconfigurations or vulnerabilities in a Kubernetes cluster. Examples of such attacks include Advanced Persistent Threats (APTs) attacks, which typically involve multiple steps and lateral movements between multiple Pods. We assume the initial step(s) of attacks can be detected by an existing detection or monitoring tool such as Falco.

As ACE-WARP is designed to mitigate the damages caused by detected attacks, we summarize the out-of-scope attacks in the following:

- Preventable Attacks: Attacks that can be effectively prevented using existing prevention solutions, e.g., credentials brute-forcing attacks blocked by password policies, will not require ACE-WARP for mitigation.
- Single-Step/Single-Pod Attacks: Attacks whose damage can be realized in a single step or in a single Pod, e.g.,

DoS/DDoS attacks launched from outside the cluster, and attacks targeting a single service or a single Pod, are out-of-scope since there is no room for ACE-WARP to react and mitigate their damages.
- Undetected Attacks: Attacks that can completely evade existing detection tools, e.g., social engineering attacks and zero-day attacks, will not trigger ACE-WARP. In fact, we rely on existing attack detection tools to provide attack alters so that ACE-WARP can prepare proactive mitigation plan.
- Scripted/Fast Attacks: Multi-step attacks that are scripted and can be performed faster than our mitigation approach can perform the migrations are out-of-scope.
- Other Attacks: Attacks targeting lower-level infrastructures than containers, attacks that can tamper with the integrity of ACE-WARP, Falco, and Kubernetes, and attacks that can breach the isolation of network slicing are all out of the scope of this paper. Similar assumption is made in [19], [25]

We assume the initial step(s) of attacks can be detected by an existing detection or monitoring tool such as Falco. We also assume the containers may be escaped due to existing vulnerabilities or weaknesses. Note that, compared to VMs, containers offer minimal separation from the host OS and other containers on the same machine, and consequently the security boundary is less robust. Although there exist approaches to provide isolation for containers, such as using Linux-based features including cgroups, AppArmor and namespace, these are not bullet-proof and can be circumvented. For instance, CVE-2022-0492 is a privilege escalation vulnerability in the kernel, that affects cgroups [53]. CVE-2017-6507 was discovered for incorrect handling of unknown AppArmor profiles [54], and CVE-2022-0185 affects the user namespaces though buffer overflow [55].

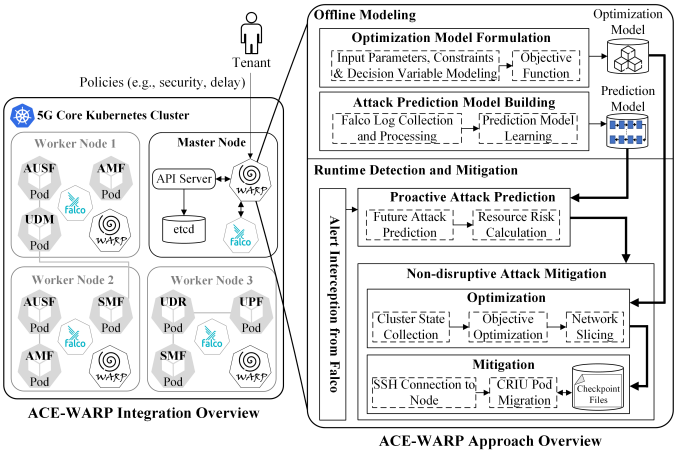


Fig. 2: An overview of ACE-WARP.

III. ACE-WARP Overview

Figure 2 shows an overview of ACE-WARP, including the overview of its integration to Kubernetes cluster (on the left) and overview of its major steps (on the right).

Integration Overview. ACE-WARP is integrated with 5G core Kubernetes cluster. Specifically, there is an ACE-WARP agent deployed in each Node (both Worker and Master) of cluster. The ACE-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 ACE-WARP.

Approach Overview. ACE-WARP approach is performed in two major phases: (i) *Offline Modeling*, and (ii) *Runtime Detection and Mitigation*. During the offline phase, ACE-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, ACE-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), ACE-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, ACE-WARP proactively prevents the attacker from proceeding with the attack. In the following, we further illustrate ACE-WARP's approach using an example.

Example 1. Figure 3 illustrates a toy example to show major steps of ACE-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, ACE-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, ACE-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, ACE-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 IV), 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, ACE-WARP performs those two migrations and thus, the cluster threat is reduced.

IV. Offline Modeling

This section describes the offline steps of ACE-WARP.

A. 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.,

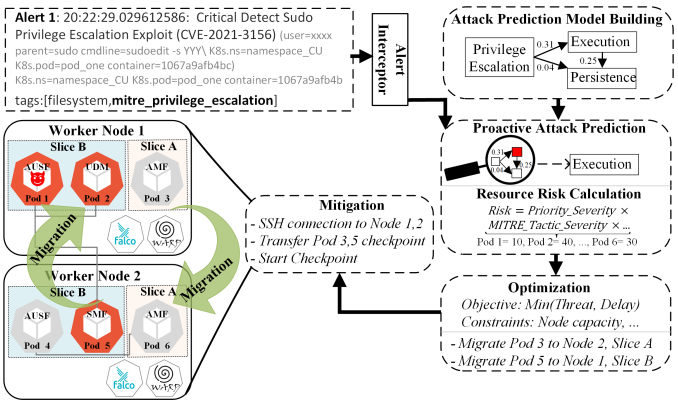


Fig. 3: Illustrative example of ACE-WARP approach overview.

TABLE II: 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

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, ACE-WARP aims at formulating the Pods placement as an optimization problem. Table II 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:

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

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 [56] 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 [57] (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. □

B. 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, ACE-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, ACE-WARP learns the predictive model from the sequences of tactics by leveraging Bayesian network [58] for this model where Nodes indicate MITRE ATT&CK tactics, edges indicate their transitions and are labeled with probabilities of transitions. Note that Falco alerts already contain MITRE tactics, and hence no additional mapping is needed (mapping Falco alerts to MITRE techniques, which are more fine-grained than MITRE tactics, can potentially make our attack prediction more accurate, although such a mapping is non-trivial [59].

Example 2. Figure 4 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.

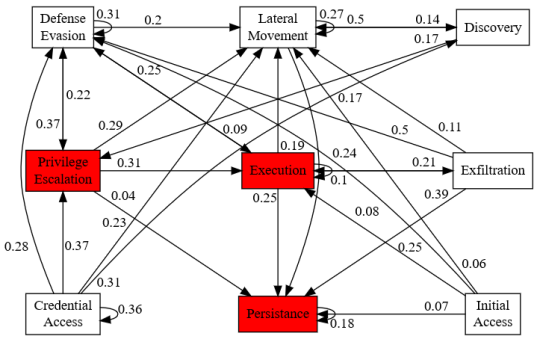


Fig. 4: Attack prediction model example.

V. Runtime Detection and Mitigation

This section presents the steps during our runtime phase.

A. Proactive Attack Prediction

This step is to predict future attacks based on alerts raised by Falco at runtime. To that end, ACE-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 V-B).

Example 3. Figure 5 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, the next likely tactic is `Execution`. Other variables of the risk formula are extracted from alert as shown in Figure 5. Finally, the overall risk for `pod_one` is calculated using the following formula.

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

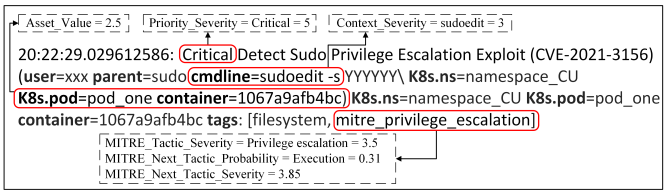


Fig. 5: Parameters value extracted from a received Falco alert.

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.

B. 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. ACE-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, ACE-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, ACE-WARP utilizes CRIU [60] 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.

1) Objective Optimization with P0P: As the optimization problem formulated in Section IV-A is non-linear in nature, we propose our heuristic algorithms (Algorithms 1 and 2) which finds the minimum number of migrations and optimal destination in order to mitigate the attack and reduce the cluster threat. Additionally, we explore the potential of using standard heuristics such as genetic algorithms to find near-optimal solutions for the problem. However, as shown in Section VII, such a standard optimization algorithm not only finds sub-optimal solutions, but is also computationally more expensive.

Next, the *Network Slicing* sub-step ensures the Pods are placed in their corresponding slices. Finally, using the migrations discovered using Algorithm 1, ACE-WARP performs the actual Pod migration from source to destination Node.

The optimal Pods placement problem is detailed into three sub-problems to prevent recomputing the steps (a dynamic programming-based approach [61]) as follows. Both algorithms participate in finding and performing the optimal migration.

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*

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

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

Require: G: Cluster graph, *MaxDelay*, *MaxCapacity*, L_{Nodes}: Nodes list, $L_{NodeCapacity}$: Nodes capacity list
Ensure: *Delay*: Migration delay
1: **for** $Node$ *in* L_{Nodes} **do**
2: $L_{NodeCapacity} = Max\_Num\_Pods(Node)$
3: **while** $Len(L_{NodeCapacity}) > 0$ **do**
4: **if** $Check\_Capacity(Destination)$ **then**
5: **if** $Check\_Delay(Pod)$ **then**
6: $MigDelay = CRIU(Pod, Destination)$
7: $Adjust\_Network\_Slice()$
8: **else**
9: $Return$: 0 {Beyond *MaxDelay*}
10: **else**
11: $Destination = L_{NodeCapacity}.pop(0)$ {Next available Node}
12: **Return MigDelay**

2) ACE-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.

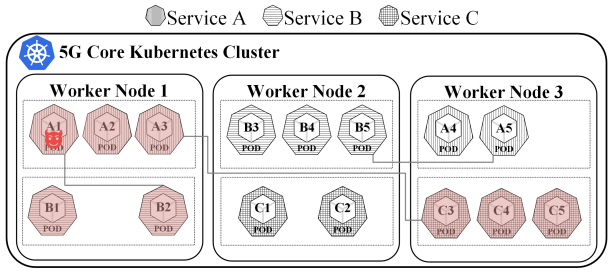
No Network Slicing. In the first method, all services are deployed in the Kubernetes cluster without being logically separated (i.e., no network slicing). Although there is much less network complexity, it is not possible to differentiate between different tenants services. As a result, ACE-WARP is obliged 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 ACE-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 ACE-WARP decides to migrate *Pod p* to *Node n*, and *Node n* does not host its slice, the migration plan must be modified.

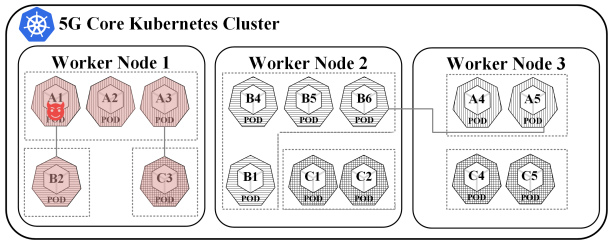
Network Slicing - Type two. In the third method, all slices are present in each Node, to overcome the previous method's disadvantages. Therefore, not only it does not interfere with P0P's optimization objectives, but also it distributes the Pods over multiple Nodes, using the resources efficiently, and resulting in less risk of SPOF. Example 4 depicts the P0P logic.

Example 4. **Real-world Scenario:** Figure 6 depicts a functional example of P0P and illustrates its adjustability to the tenants' requirements in a 5G scenario. Three services (`A, B, C`) are deployed in their slices in the cluster. As shown in Figure 6a, `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.



Cluster threat = AV(A1,A2,A3) + AV(B1,B2) + AV(C3,C4,C5)

(a) Before: attacker initial step.



Cluster threat = AV(A1,A2,A3) + AV(B2) + AV(C3)

(b) After: ACE-WARP mitigation.

Fig. 6: ACE-WARP and network slicing.

In the second part of this example, we elaborate on the advantage of network slicing for tenants' requirements adjustability. Table III 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, the real-world use case for this approach in case of not utilizing network slicing is that 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 III: ACE-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

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 [62]. 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 [60] 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 ACE-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. Therefore, any container in the new Pods will use the latest secure snapshot instead of the compromised one. 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 mainly affected by the sizes of containers (discussed in Section VII-A. As a result, tenants will not experience interrupted connections, except some delay in responses. Such a delay of a few seconds can be considered acceptable to tenants since even longer delays may naturally happen over the internet connection between the tenants and cloud.

We overlook the checkpoint transfer time to the disk, as several transfer methods (e.g., SSH/SCP, FTP, rsync [63]) exist with various performances.

VI. Implementation

A. 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 [21] 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 [21]).

Furthermore, the communication from the control plane API server to the worker nodes is through the `kubelet` present on each node in the cluster. Similarly to Falco, ACE-WARP has one agent (Pod) in the control plane and every worker node. Thus, any communication from master node to the worker nodes is through the API server which is based on the standard trust model of Kubernetes [1]. Kubernetes provides SSH tunneling or `–kubelet–certificate–authority` flag to prevent man-in-the-middle and untrusted connection. Moreover, as a replacement to the SSH tunnels, Kubernetes has provided the "Konnectivity" service which provides TCP level proxy for the control plane to cluster communication. Therefore, all the information sent from worker nodes to the control plane is secure and reliable.

B. Implementing and Integrating ACE-WARP with Kubernetes

ACE-WARP is implemented and integrated with Kubernetes following the architecture shown in Figure 7. Specifically, ACE-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 [64]. 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.

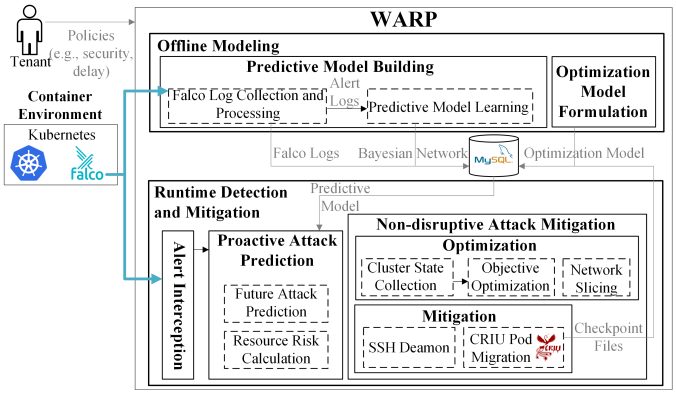


Fig. 7: ACE-WARP architecture.

C. Auto-scaling ACE-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 ACE-WARP in such a way that it can also scale horizontally using Kubernetes DaemonSet [65]. Specifically, new ACE-WARP agents will be automatically created upon Nodes scaling up, and those agents will communicate with the main ACE-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 ACE-WARP agents.

D. Portability to Other Cloud Platforms

ACE-WARP can potentially be adapted to other major cloud platforms, such as Amazon Elastic Container Service (ECS) [66] and Microsoft Azure [67]. The components of ACE-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 [10], Sysdig Secure [68], and Prometheus [69]) 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* [70] and *AWS* [71]). Second, although the optimization step is platform-independent, the migration step of ACE-WARP will depend on specific migration techniques used in the cloud platform. Finally, ACE-WARP is independent of the container orchestrator and can thus work for other orchestration systems such as Docker Swarm [72] or OpenShift [73].

E. Building Dataset

To facilitate learning our prediction model (see Section IV-B) and to support future research, we build a relatively large dataset of Falco alerts for Kubernetes, which is publicly available on GitHub [11]. 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

TABLE IV: Overview of ACE-WARP dataset.

Attack ID	Attack Campaign	Attack Features[a] PL	PA	INJ	IG	BD	MITRE ATT&CK Tactic Sequence[b]
1	APT 3 [74]	✓	✓	✓	✓	✓	Exe, DE, Dis, DE, LM
2	Spam campaign [75]		✓	✓	✓	✓	Dis, Per, Exe, DE, DE, LM, Exf
3	APT 29 [76]	✓	✓	✓	✓	✓	Per, Exe, DE, PE, DE, Dis, LM, IA, Per, PE, DE
4	Escape attack [77]				✓		PE, Exe, Per
5	Simulated cryptominer spread [78]	✓		✓	✓	✓	Dis, Exe, Per, DE, LM
6	Root data theft [79]			✓	✓	✓	Dis, Per, PE, Exf, Per, LM
7	SWC [80]	✓		✓	✓	✓	Dis, Exe, DE, Per
8	Targeted gov phishing [81]	✓		✓	✓	✓	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

we label these samples as "normal". For the attack alerts, we leverage CALDERA [82], 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 IV 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.

VII. Evaluation

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

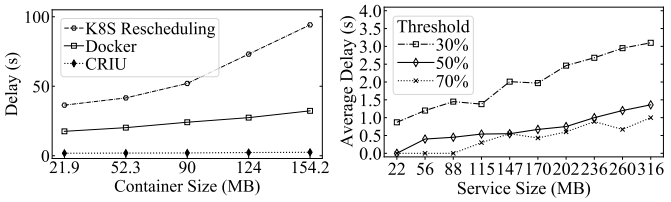
RQ1. What is the cost of Pod migration?
RQ2. How effective is our optimization algorithm (P0P)?
RQ3. How effective is ACE-WARP for mitigating attacks?
RQ4. How much overhead does ACE-WARP incur?
RQ5. What is the impact of network slicing on ACE-WARP?
RQ6. What is the impact of false positives on ACE-WARP?
RQ7. How much does ACE-WARP surpass MTD approaches?

A. Migration Cost

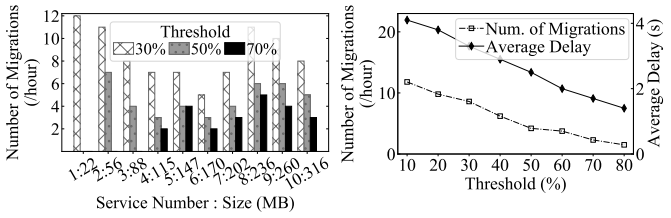
To answer RQ1, we measure the cost of migration.

Migration Approaches Comparison. As discussed in Section V-B3, 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 8a, 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) [60].

Migration Delay on Services. Figure 8b 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 8c 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 8d 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.



(a) Migration approaches comparison. (b) Migration average delay/service.



(c) Number of migrations/hour. (d) Migration frequency and delay.

Fig. 8: Migration approaches comparison and cost analysis.

B. Optimization Effectiveness

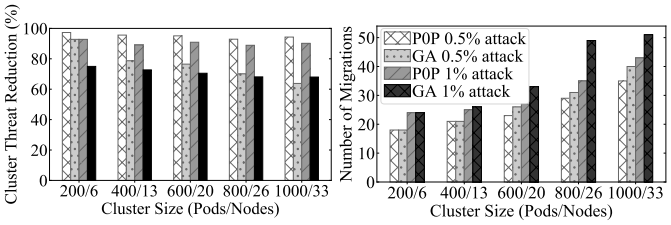
To answer RQ2, we compare our P0P heuristic algorithm with a standard optimization solution (genetic algorithm (GA) [83]).

To evaluate the effectiveness of those two algorithms, we measure the cluster threat reduction (Figure 9a) and the number of migrations (Figure 9b) 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.

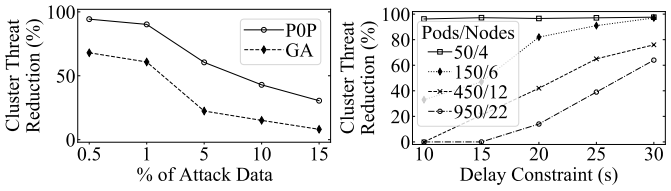
We also compare the number of migrations required by P0P and GA for reducing the cluster threat (Figure 9b). P0P achieves more threat reduction while migrating less Pods under all sizes of clusters. Note that the total number of migrations is reasonable considering the fact that the average migration delay per Pod is only about 0.87 (seconds) using CRIU (Figure 8a).

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 9c) 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 IV-A). Figure 9d 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.

Fig. 9: Cluster threat reduction and delay comparison ((a), (b), (c): no delay constraint).

C. ACE-WARP Effectiveness

To answer RQ3, we evaluate the effectiveness of ACE-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 V shows the numerical results for six of the simulated attacks. We selected six out of the eight attack campaigns from our dataset (Table IV) based on the simulation results which generated more Falco alerts and diversity of observed MITRE tactics. As a result, we considered these six attacks as a case study in Table V.

For a lower threshold value (e.g., 30%), ACE-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 (Figure 10). Therefore, adopting such a lower threshold is generally preferable, especially considering that the false positives have less impact under ACE-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).

TABLE V: ACE-WARP effectiveness per attack and dataset.

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

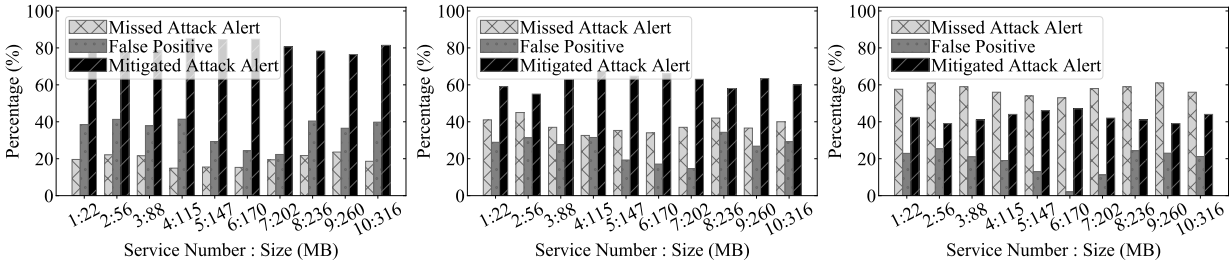
Although our solution might suffer from false negatives (either due to Falco misdetection, or inaccurate risk prediction), ACE-WARP is designed to catch long-lasting multi-step attacks and is unlikely to miss all the attack steps.

D. Performance

To answer RQ4, we perform two experiments to evaluate the execution time and CPU consumption. Figure 11a 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 11b shows the CPU consumption of ACE-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.

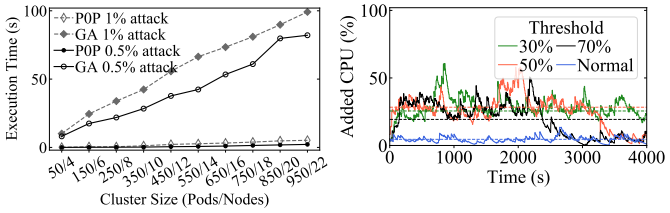
E. Adjustability to Tenants' Requirements

To answer RQ5, this experiment studies how network slicing can meet different services' requirements, and how ACE-WARP can be leveraged as an adjustable security solution. Figure 12 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 ACE-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 ACE-WARP agent (residing in a Node) performs the migration for threat reduction based on its own threshold value. Subsequently, ACE-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.



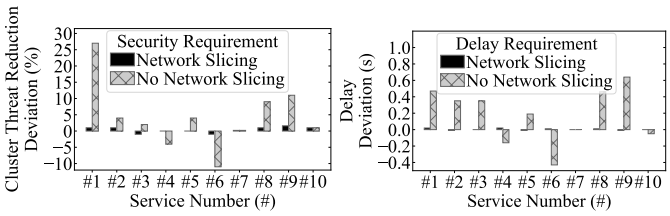
(a) Threshold = 30%. (b) Threshold = 50%. (c) Threshold = 70%.

Fig. 10: ACE-WARP effectiveness.



(a) P0P Execution time. (b) ACE-WARP CPU consumption.

Fig. 11: Performance overhead.



(a) Threat reduction. (b) Imposed delay.

Fig. 12: Effect of network slicing.

F. False Positives Impact

Since ACE-WARP acts based on the risk value calculated for each alert, there is a possible scenario where attackers generate fake alerts to purposefully trigger ACE-WARP. This may happen in case the attacker gains knowledge of the risk formula and ACE-WARP configuration (e.g., threshold values). Therefore, an attacker might produce false positive events to trigger migration and overwhelm the system. Note that, even if the attacker compromises a Pod and triggers endless migration, the malicious pods will not be spreading over the cluster. This is because we designed our threat reduction algorithm (i.e., P0P) in a way that isolates the source of an attack.

To answer RQ6, Figure 13 illustrates that by varying the percentage of false positives alerts, how ACE-WARP is still able to reduce the threat by isolating the attack and keep the other resources safe. As a result, in large cluster (i.e., 950 Pods and 22 Nodes), ACE-WARP reduces the threat even with a false positive rate up to 55%.

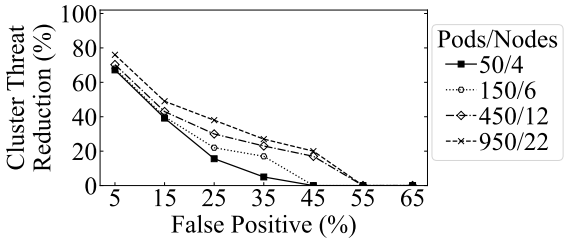


Fig. 13: Threat Reduction in case of Excessive False Positives

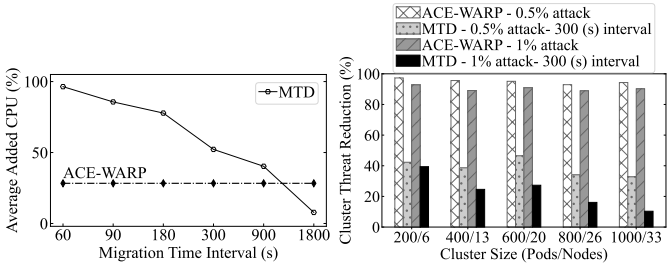
G. ACE-WARP Comparison with a Proactive MTD

To answer RQ7, we conduct a set of experiments (shown in Figure 14) where we compare ACE-WARP's overhead and effectiveness with other MTD solutions (as reviewed in Section I-A). In the following, we first discuss how we implement a comparable MTD solution and then present our comparison results.

ACE-WARP is a predictive solution which migrates the Kubernetes Pods based on a received alert and a predicted risk in order to proactively mitigate any potential attack. However, proactive MTDs are time-based and non-attack dependent migration. Thus, in order to fairly compare the effectiveness of our approach with MTD solutions (i.e., reducing the threat), we specifically implement a MTD approach that randomly (independent of attacks) chooses Pod(s) to migrate at a regular time interval. Figure 14 illustrates the experimental results in terms of imposed average overhead, and effectiveness (cluster threat reduction). As shown in Figure 14a, the average added CPU by MTD is significantly higher than that of ACE-WARP, due to its additional migrations (mainly in lower time intervals). However, since ACE-WARP performs selective migrations, it avoids the unnecessary overhead. Based on our results in Figure 11b, for this experiment, we selected the threshold of 30% for ACE-WARP as it is more sensitive (performs more migrations). Note that, the ACE-WARP line in Figure 14a is an indicator, and MTD can only perform one migration every 1,000 seconds to have a similar CPU as ACE-WARP which is giving enough time to attacker to proceed in the cluster.

Figure 14b shows the cluster threat reduction for both ACE-WARP and MTD under both 0.5%, and 1% attack data. As observed, ACE-WARP significantly surpasses MTD for reducing the cluster threat for both types of attack data. This

is because a larger number of migrations do not necessarily reduce the cluster threat. Since the migrations in MTD are not based on attack indicator and predicted risk, the MTD migrates Pods randomly and do not necessarily move the riskiest Pods in the cluster to reduce the attack surface. Although MTD may prevent zero-day attacks as it performs migration on a timely manner and zero-day attacks do not have indicator, this prevention is not guaranteed due to its random selection of Pods for migration, and not necessarily the attacked Pod. However, ACE-WARP uses predictive model to forecast the attacker's potential next step, calculates risk, and select the riskiest Pods using our P0P optimization algorithm. In contrast to MTD, ACE-WARP imposes the migration delay only on the Pods under attack. As a result, ACE-WARP is able to reduce the cluster threat twice as much as the MTD with much less overhead, and delay.



(a) Threat reduction. (b) Imposed delay.

Fig. 14: ACE-WARP overhead and effectiveness vs. MTD

VIII. Conclusion

We presented ACE-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.

Discussion. First, we clarify our comprehension of preventive and proactive terms as the nature of ACE-WARP may overlap in both terms. The high-level objective of ACE-WARP is preventive in nature as we aim to prevent further damages caused by attack propagation. On the other hand, the way we achieve this objective is proactive in the sense that we perform mitigation *before* real attacks occur, so our mitigation is indeed independent of those real attacks (which explains why we need non-disruptive mitigation options, since the real attacks may or may not occur after the mitigation). In other words, prevention is our goal, and proactive mitigation is our methodology to achieve it. The main reason we do not use the term preventive is to avoid the potential confusion that our method is designed to prevent all attacks (e.g., it cannot prevent the already detected attack).

Second, since ACE-WARP is deployed by the provider as the security solution for the cluster, it already takes into account all tenants' workloads. Moreover, the optimization algorithm ensures that a malicious Pod will not be detrimental to any tenants regardless. We provide tenants the flexibility of assigning relative importance (α and β) to the two objectives (i.e., *cluster threat* and *delay*) based on their specific needs (see Section IV-A). In practice, tenants can determine those parameters based on their cluster sizes and delay constraints (as specified in Service Level Agreements (SLAs)) by referring to our experimental results shown in Figure 9d, which illustrate how the reduction in cluster threat will be affected by the cluster size and delay constraints. For instance, tenants with smaller clusters (50 Pods) can choose larger β values (hence smaller α) since stricter delay constraints will not have significant impact on threat reduction, whereas tenants with larger clusters should choose smaller β values based on the delays their SLAs can tolerate.

Limitations and Future Work. We are aware that ACE-WARP utilizes 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, ACE-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.

Acknowledgment

We thank the anonymous reviewers for their valuable comments and suggestions. This work was supported by the Natural Sciences and Engineering Research Council of Canada and Ericsson Canada under the Industrial Research Chair in SDN/NFV Security, and the Canada Foundation for Innovation under JELF Project 38599.

References

[1] "Kubernetes," https://kubernetes.io/, [Accessed 30-3-2022].
[2] "Containers vs. VM," https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm, [Accessed 4-7-2023].
[3] "Container images vul." https://sysdig.com/learn-cloud-native/container-security/docker-vulnerability-scanning, [Accessed 4-7-2023].
[4] Y. Avrahami and S. Ben Hai, "Kubernetes privilege escalation: Container escape == cluster admin?" in *Black Hat USA*, 2022.
[5] "Open Policy Agent/Gatekeeper," Open Policy Agent/Gatekeeper https://open-policy-agent.github.io/gatekeeper/, [Accessed 10-3-2023].
[6] "Seccomp security profiles forDocker," https://github.com/docker/docker/blob/master/docs/security/seccomp.md, [Accessed 4-7-2023].
[7] N. DeMarinis *et al.*, "Sysfilter: Automated system call filtering for commodity software," in *RAID*, 2020.
[8] S. Ghavamnia *et al.*, "Confine: Automated system call policy generation for container attack surface reduction," in *RAID*, 2020.
[9] L. Lei *et al.*, "Speaker: Split-phase execution of application containers," in *DIMVA*. Springer, 2017.
[10] "Falco," https://falco.org/, [Accessed 30-3-2022].
[11] "Falco alert dataset with APT attacks," https://github.com/simabagheri1/Falco-Alerts-Dataset-with-APT-attacks.
[12] "MITRE Att&CK," https://attack.mitre.org/, [Accessed 30-3-2022].

[13] "5G network slicing," https://www.ericsson.com/en/network-slicing, [Accessed 10-3-2023].
[14] S. Bagheri *et al.*, "Warping the Defence Timeline: Non-disruptive Proactive Attack Mitigation for Kubernetes Clusters," in *IEEE ICC*, 2023.
[15] M. Patnaik *et al.*, "Prolemus: A proactive learning-based mac protocol against puea and ssdf attacks in energy constrained cognitive radio networks," *TCCN*, 2019.
[16] S. Majumdar *et al.*, "Prosas: Proactive security auditing system for clouds," *TDSC*, 2021.
[17] T. Ma *et al.*, "A mutation-enabled proactive defense against service-oriented man-in-the-middle attack in kubernetes," *IEEE Trans. Comput.*, 2023.
[18] C. Miranda *et al.*, "A collaborative security framework for software-defined wireless sensor networks," *IEEE Transactions on Information Forensics and Security*, 2020.
[19] H. Kermabon-Bobinnec *et al.*, "Prospec: Proactive security policy enforcement for containers," in *ACM CODASPY*, 2022.
[20] H. Liu *et al.*, "Watermark-based proactive defense strategy design for cyber-physical systems with unknown-but-bounded noises," *IEEE Trans. Autom. Control.*, 2022.
[21] A. Alsaheel *et al.*, "ATLAS: A sequence-based learning approach for attack investigation," in *USENIX Security*, 2021.
[22] W. Hassan *et al.*, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *NDSS*, 2019.
[23] X. Han *et al.*, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *NDSS*, 2020.
[24] "Sysdig," Sysdig https://sysdig.com/, [Accessed 10-3-2023].
[25] C.-W. Tien *et al.*, "Kubanomaly: anomaly detection for the docker orchestration platform with neural network approaches," *Engineering reports*, 2019.
[26] Q. Du *et al.*, "Anomaly detection and diagnosis for container-based microservices with performance monitoring," in *ICA3PP*, 2018.
[27] Z. Zou *et al.*, "A docker container anomaly monitoring system based on optimized isolation forest," *IEEE Trans. on Cloud Comput.*, 2019.
[28] "OpenStack Congress," 2015. [Online]. Available: https://wiki.openstack.org/wiki/Congress/
[29] S. Majumdar *et al.*, "LeaPS: Learning-based proactive security auditing for clouds," in *ESORICS*. Springer, 2017.
[30] ——, "Proactive verification of security compliance for clouds through pre-computation: Application to openstack," in *ESORICS*, 2016.
[31] ——, "Learning probabilistic dependencies among events for proactive security auditing in clouds," in *J. Comput. Secur.*, 2019.
[32] W. Hassan *et al.*, "Tactical provenance analysis for endpoint detection and response systems," in *IEEE SP*, 2020.
[33] ——, "OmegaLog: High-fidelity attack investigation via transparent multi-layer log analysis," in *NDSS*, 2020.
[34] R. Yang *et al.*, "UIScope: Accurate, instrumentation-free, and visible attack investigation for GUI applications." in *NDSS*, 2020.
[35] S. Milajerdi *et al.*, "Holmes: real-time APT detection through correlation of suspicious information flows," in *IEEE SP*, 2019.
[36] A. De Domenico *et al.*, "Optimal virtual network function deployment for 5g network slicing in a hybrid cloud infrastructure," *TWC*, 2020.
[37] S. Bagheri *et al.*, "Dynamic firewall decomposition and composition in the cloud," *TIFS*, 2020.
[38] Z. Ding *et al.*, "Kubernetes-oriented microservice placement with dynamic resource allocation," *IEEE Trans. on Cloud Comput.*, 2022.
[39] S. Arzo *et al.*, "Study of virtual network function placement in 5g cloud radio access network," *IEEE Trans. Netw. Serv.*, 2020.
[40] 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.
[41] M. Kabir *et al.*, "Joint routing and scheduling of mobile charging infrastructure for v2v energy transfer," *IEEE Trans. Intell. Veh.*, 2021.
[42] F. Bari *et al.*, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Serv.*, 2016.
[43] W. Attaoui *et al.*, "Vnf and cnf placement in 5g: Recent advances and future trends," *IEEE Trans. Netw. Serv.*, 2023.
[44] J.-H. Cho *et al.*, "Toward proactive, adaptive defense: A survey on moving target defense," *IEEE Communications Surveys & Tutorials*, 2020.
[45] H. Jin *et al.*, "Dseom: A framework for dynamic security evaluation and optimization of mtd in container-based cloud," *IEEE Transactions on Dependable and Secure Computing*, 2019.
[46] O. Stan *et al.*, "Heuristic approach for countermeasure selection using attack graphs," in *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, 2021.
[47] P. Nespoli *et al.*, "Aisga: Multi-objective parameters optimization for countermeasures selection through genetic algorithm," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, 2021.
[48] "Docker." [Online]. Available: https://www.docker.com/
[49] "CNCF Survey Report," www.cncf.io, 2020, [Accessed 10-10-2022].
[50] "Free5GC," https://www.free5gc.org/, [Accessed 30-3-2022].
[51] B. Martini *et al.*, "Intent-based network slicing for sdn vertical services with assurance: Context, design and preliminary experiments," *FGCS*, 2023.
[52] "CVE-2021-3156," https://nvd.nist.gov/vuln/detail/CVE-2021-3156/, [Accessed 30-3-2022].
[53] "cgroup-vuk," 2022. [Online]. Available: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=CVE-2022-0492+
[54] "apparmor-vul," 2017. [Online]. Available: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=CVE-2017-6507
[55] "namespace-vul," 2022. [Online]. Available: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=CVE-2022-0185
[56] I. Munro, "Efficient determination of the transitive closure of a directed graph," *Information Processing Letters*, 1971.
[57] F. Grandoni, "A note on the complexity of minimum dominating set," *J. Discrete Algorithms*, 2006.
[58] D. Heckerman, "A tutorial on learning with bayesian networks," 2021.
[59] "mitre-cti." [Online]. Available: https://attack.mitre.org/resources/learn-more-about-attack/training/cti/
[60] "CRIU," https://criu.org, [Accessed 30-3-2022].
[61] R. E. Bellman, *Dynamic programming*. Princeton university press, 2010.
[62] "CORDON/UNCORDON," https://kubernetes.io/docs/tasks/administer-cluster/safely-drain-node/, [Accessed 10-3-2023].
[63] "Rsync," https://linux.die.net/man/1/rsync, [Accessed 10-3-2023].
[64] "Falco Installation Tools," https://falco.org/docs/getting-started/third-party/install-tools/#helm, [Accessed 10-3-2023].
[65] "DaemonSet | Kuberetes," Daemonset | Kubernetes. https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/, [Accessed 30-3-2022].
[66] "Amazon ECS," Amazon Elastic Container Service (ECS) https://aws.amazon.com/ecs/, [Accessed 25-4-2023].
[67] "Microsoft Azure," Microsoft Azure https://azure.microsoft.com/, [Accessed 25-4-2023].
[68] "Sysdig SECURE," Sysdig SECURE https://sysdig.com/products/secure/, [Accessed 25-4-2023].
[69] "Prometheus," Prometheus https://prometheus.io/, [Accessed 25-4-2023].
[70] "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].
[71] "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].
[72] "Docker Swarm," Docker Swarm https://docs.docker.com/engine/swarm/, [Accessed 25-4-2023].
[73] "OpenShift," OpenShift. https://docs.openshift.com/, [Accessed 30-3-2022].
[74] "APT 3," https://attack.mitre.org/groups/G0022/, [Accessed 30-3-2022].
[75] "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].
[76] "APT 29," https://attack.mitre.org/groups/G0016/, [Accessed 30-3-2022].
[77] "Escape attack," Exploiting CVE-2021-3156. https://www.helpnetsecurity.com/2021/01/27/cve-2021-3156/, [Accessed 30-3-2022].
[78] "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].
[79] "Root data theft," Kernel vulnerability exploiting CVE-2020-14386. https://nvd.nist.gov/vuln/detail/CVE-2020-14386/, [Accessed 30-3-2022].
[80] "Strategic web compromise," https://www.fireeye.com/blog/threat-research/2015/07/second_adobe_flashz0.html/, [Accessed 30-3-2022].
[81] "Pierluigi Paganini," Phishing campaigns target US government agencies exploiting hacking team flaw CVE-2015- 5119. , [Accessed 30-3-2022].
[82] "CALDERA," https://caldera.mitre.org/, [Accessed 30-3-2022].
[83] W. Banzhaf *et al.*, *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., 1998.