Warping the Defence Timeline: Non-disruptive Proactive Attack Mitigation for Kubernetes Clusters

Sima Bagheri*, Hugo Kermabon-Bobinnec*, Suryadipta Majumdar*,
Yosr Jarraya§, Lingyu Wang*, Makan Pourzandi§
*Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada
Emails: {sima.bagheri, hugo.kermabonbobinnec, suryadipta.majumdar}@concordia.ca, wang@ciise.concordia.ca
§Ericsson Security Research, Ericsson, Canada. Emails: {yosr.jarraya, makan.pourzandi}@ericsson.com

Abstract—**In spite of being the de-facto standard of container orchestrators, Kubernetes reportedly suffers from security vulnerabilities and misconfigurations which may lead to severe security threats to the containerized environments it manages. Mitigating such threats based on alerts raised by existing security monitoring solutions (e.g., Falco) can be challenging. First, taking actions upon every alert can cause unacceptable service disruption, as many such alerts may turn out to be false positives. Second, validating each alert by administrators before taking actions may render the mitigation too late to prevent irreversible damages, e.g., denial of service. In this paper, we propose a non-disruptive proactive mitigation approach to address those limitations. Our main idea is to proactively trigger mitigation ahead of an attack to prevent irreversible damages, while designing the mitigation actions to be non-disruptive to avoid any service disruption caused by false alerts. We implement and integrate our approach with Kubernetes, and show its effectiveness and efficiency.**

Index Terms—**Attack mitigation, container, Kubernetes**

I. Introduction

Due to its inherent agility and flexibility, containerization has emerged as a popular choice for many large-scale applications, such as 5G networks. Such containerized applications are typically deployed and managed through container orchestrators, such as the de-facto standard choice, Kubernetes [1]. Even though orchestrators play a vital role for the applications, existing orchestrators reportedly suffer from various vulnerabilities and misconfigurations whose exploitation may allow attackers to cause severe service disruption (e.g., Kubernetes privilege escalation shown at Black Hat USA 2022 [2]). Therefore, attack mitigation is critical for ensuring the service continuity of containerized applications.

To that end, most existing security solutions for Kubernetes fall short. First, the well-known high false positive rate of existing security monitoring tools (e.g., Falco [3]) means that an aggressive approach of taking mitigation actions upon seeing every alert is impractical, as any false alarm may lead to unacceptable service disruption. Instead, most administrators would validate the alerts before taking mitigation actions, which can be tedious and cause significant delay to the mitigation. Second, there exist "proactive" attack mitigation solutions (e.g., [4]–[6]) which reduce such a delay by performing costly security verification in advance based on prediction. Nonetheless, as their mitigation steps are still triggered *after* the attacks occur, it may still be too late to prevent irreversible damages such as denial of service or information leakage. Those limitations will be further illustrated to motivate towards our solution through an example in Section II-B.

In this paper, we propose a novel approach to proactively trigger mitigation actions *before* attacks actually occur, namely, *WARP* (i.e., warping the defence timeline between attack and its mitigation). To avoid any service disruption caused by false alarms, our idea is to rely on non-disruptive mitigation actions (e.g., live migration of containers) with negligible delay. Specifically, WARP first builds a predictive model of attacks based on Falco alerts in Kubernetes, as well as MITRE ATT&CK framework [7] for attacker's tactics and strategies. Second, using the predictive model, WARP predicts the attacker's next steps at runtime, identifies the resources that might be involved, and evaluates the risks of potential damages. Finally, it migrates the resources according to an optimal strategy for mitigating the overall damages to the cluster. WARP is implemented for Kubernetes and evaluated using real-world APT attacks simulated in a controlled environment.

In summary, our main contributions are as follows:

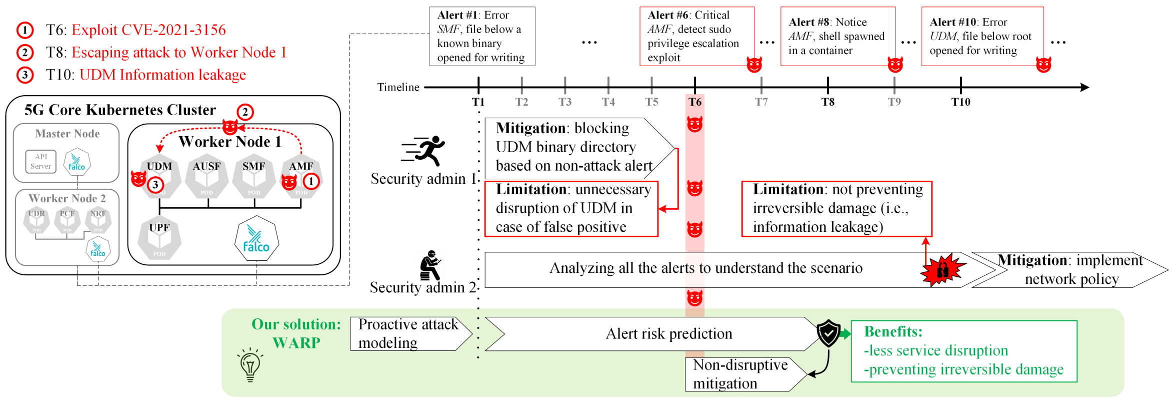
- As per our knowledge, this is the first proactive attack mitigation approach that can also avoid the service disruption caused by potential false alarms.
- We instantiate this approach based on Kubernetes through developing techniques for predictive model construction, attack prediction, risk evaluation, and optimal migration.
- We build the first large-scale Kubernetes attack dataset with 231k Falco alerts based on real-world APT attacks simulated in a controlled environment. Our experiments using the dataset show WARP's efficacy (e.g., mitigating 81% of the alerts with a cumulative delay of 30 seconds per hour for the entire cluster during attack).

The rest of this paper is organized as follows: Section II provides preliminaries. Section III, IV and V detail our approach, implementation and experiments, respectively. Section VI reviews literature and Section VII draws conclusion.

II. Preliminaries

A. Background

Kubernetes is widely regarded as one of the most popular container orchestrators [8]. Left side of Figure 1 shows a simplified view of a Kubernetes cluster. Such a cluster contains



① T6: Exploit CVE-2021-3156
② T8: Escaping attack to Worker Node 1
③ T10: UDM Information leakage
5G Core Kubernetes Cluster
Master Node
API Server
falco
Worker Node 2
UDR
PCF
NRF
falco
Worker Node 1
UDM
AUSF
SMF
AMF
UPF
POD
falco
Alert #1: Error SMF, file below a known binary opened for writing
Alert #6: Critical AMF, detect sudo privilege escalation exploit
Alert #8: Notice AMF, shell spawned in a container
Alert #10: Error UDM, file below root opened for writing
Timeline
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
Security admin 1
Mitigation: blocking UDM binary directory based on non-attack alert
Limitation: unnecessary disruption of UDM in case of false positive
Limitation: not preventing irreversible damage (i.e., information leakage)
Security admin 2
Analyzing all the alerts to understand the scenario
Mitigation: implement network policy
Our solution: WARP
Proactive attack modeling
Alert risk prediction
Non-disruptive mitigation
Benefits:
-less service disruption
-preventing irreversible damage

Fig. 1: Motivating example.

multiple worker nodes and a master node, where each node has several Pods as deployment unit to host various services, such as network functions in a 5G network (UDM, AUSF, etc.). Such a cluster might also be configured with Falco, which is a Kubernetes runtime security solution that monitors syscalls and raises alerts about occurring events in a cluster.

B. Motivating Example

As a motivating example, Fig. 1 illustrates an attack scenario on a Kubernetes cluster hosting a 5G core due to a real-world vulnerability (CVE-2021-3156 [9]) in Kubernetes. As shown on the left, the attacker performs three major steps: (1) at time `T6`, exploiting CVE-2021-3156 to infect a Pod (`AMF`) in `Worker node 1`, (2) at `T8`, escaping to the node level at `Worker node 1`, and (3) at `T10`, leaking information at another Pod (`UDM`) in the same node. During the progress of this attack, Falco (mentioned above) raises three alerts: `Alert #6` at `T6`, `Alert #8` at `T8`, and `Alert #10` at `T10`. To mitigate this attack based on those alerts, the right side of the figure shows two potential approaches:

- **Fast (but disruptive) mitigation**: As shown in the figure, suppose `Security admin 1` adopts a fast mitigation approach by immediately blocking the access to the binary directory of the adjacent Pod (`AUSF`) upon observing `Alert #1` at `T1` (which indicates a writing in `SMF` binary directory) with the hope of limiting the attack damage. However, suppose the admin later finds out that `Alert #1` is actually a non-attack alert (caused by a false positive). Consequently, this approach has caused an unnecessary disruption to the `AUSF` service.
- **Non-disruptive (but slow) mitigation**: As shown in the figure, suppose `Security admin 2` adopts a non-disruptive mitigation approach by analyzing each alert (till `Alert #10`) to fully understand the scenario before implementing a new network policy for mitigation. However, as the mitigation happens after the attacker moves to the worker node, this approach fails to prevent the information leakage at `UDM`, whose damage is irreversible.

Our Solution. As shown on the bottom of the figure, WARP overcomes both limitations by proactively building an attack prediction model (before `T1`), continuously predicting the risks as alerts are received (e.g., at `T6`, predicting the risk for all Pods in `Worker node 1` from `Alert #6`), and at `T6`, applying non-disruptive mitigation (i.e., live migration of Pod) for `UDM`. Consequently, WARP can limit the damage of information leakage at `UDM` without causing any unnecessary service disruption (despite the presence of false alarms).

C. Threat Model

The in-scope threats include attacks that are launched by either external attackers or insiders through exploiting misconfigurations or vulnerabilities in a containerized environment such as Kubernetes, and we assume the initial step(s) of those attacks can be detected by an existing detection or monitoring tool such as Falco (the detection itself is out of the scope of this paper). The out-of-scope threats include any attacks which may be effectively prevented through security patches, firewalls, intrusion prevention systems, etc. (as WARP is designed to mitigate the damages caused by “inevitable” attacks such as APTs), zero day attacks which can completely evade existing detection tools (as WARP is triggered by their alerts), or attacks targeting lower-level infrastructures than containerized applications and services (as WARP focuses on Kubernetes). Finally, we assume the integrity of WARP, Falco, and Kubernetes is protected with existing trusted computing techniques (e.g., remote attestation [10]) and any integrity breach is beyond our scope.

III. WARP Approach

A. Overview

Fig. 2 shows an overview of WARP with two major phases: (i) *Offline Proactive Attack Modeling*, and (ii) *Runtime Detection and Mitigation*. The inputs to WARP are the Falco alerts collected from Kubernetes environment. During the offline phase, WARP extracts the MITRE ATT&CK tactics property from Falco alerts and builds a predictive model with probabilistic dependency relationships among MITRE ATT&CK tactics. During the runtime phase, when *Alert Interception* detects a Falco alert, WARP leverages the predictive model to predict the attacker’s next move, calculates a risk value for all the resources, and optimally selects Pods to be migrated to another node. In the following, we elaborate on both phases.

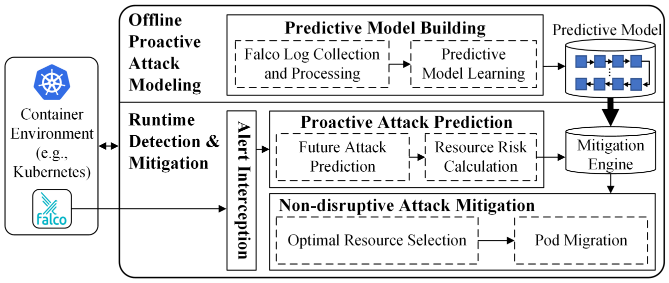
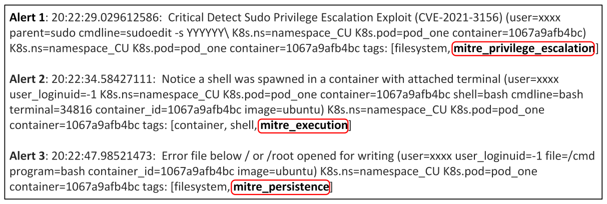
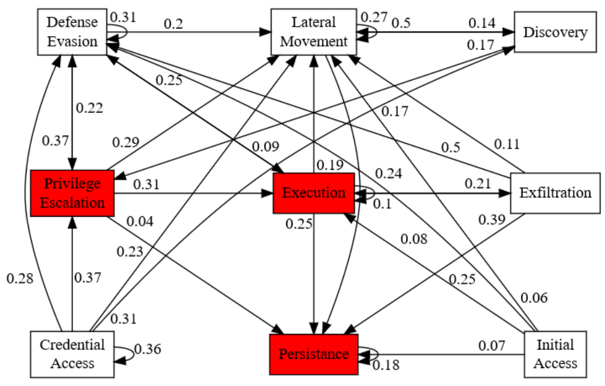

Container Environment (e.g., Kubernetes)
Falco
Offline Proactive Attack Modeling
Predictive Model Building
Falco Log Collection and Processing
Predictive Model Learning
Predictive Model
Runtime Detection & Mitigation
Alert Interception
Proactive Attack Prediction
Future Attack Prediction
Resource Risk Calculation
Mitigation Engine
Non-disruptive Attack Mitigation
Optimal Resource Selection
Pod Migration

Fig. 2: Overview of WARP approach.

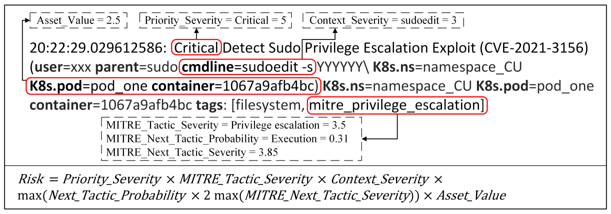


```
Alert 1: 20:22:29.029612586:  Critical Detect Sudo Privilege Escalation Exploit (CVE-2021-3156) (user=xxxx
parent=sudo cmdline=sudoedit -s YYYYYY\ K8s.ns=namespace_CU K8s.pod=pod_one container=1067a9afb4bc)
K8s.ns=namespace_CU K8s.pod=pod_one container=1067a9afb4bc tags: [filesystem, mitre_privilege_escalation]

Alert 2: 20:22:34.58427111:  Notice a shell was spawned in a container with attached terminal (user=xxxx
user_loginuid=-1 K8s.ns=namespace_CU K8s.pod=pod_one container=1067a9afb4bc shell=bash cmdline=bash
terminal=34816 container_id=1067a9afb4bc image=ubuntu) K8s.ns=namespace_CU K8s.pod=pod_one
container=1067a9afb4bc tags: [container, shell, mitre_execution]

Alert 3: 20:22:47.98521473:  Error file below / or /root opened for writing (user=xxxx user_loginuid=-1 file=/cmd
program=bash container_id=1067a9afb4bc image=ubuntu) K8s.ns=namespace_CU K8s.pod=pod_one
container=1067a9afb4bc tags: [filesystem, mitre_persistence]
```

(a) Falco log alerts example



(b) Predictive model example



```
Asset_Value = 2.5    Priority_Severity = Critical = 5    Context_Severity = sudoedit = 3

20:22:29.029612586: Critical Detect Sudo Privilege Escalation Exploit (CVE-2021-3156)
(user=xxx parent=sudo cmdline=sudoedit -s YYYYYY\ K8s.ns=namespace_CU
K8s.pod=pod_one container=1067a9afb4bc) K8s.ns=namespace_CU K8s.pod=pod_one
container=1067a9afb4bc tags: [filesystem, mitre_privilege_escalation]

        MITRE_Tactic_Severity = Privilege escalation = 3.5
        MITRE_Next_Tactic_Probability = Execution = 0.31
        MITRE_Next_Tactic_Severity = 3.85
```

Risk = *Priority_Severity* × *MITRE_Tactic_Severity* × *Context_Severity* ×
max(*Next_Tactic_Probability* × 2 max(*MITRE_Next_Tactic_Severity*)) × *Asset_Value*

(c) Risk calculation example

Fig. 3: WARP's initial offline and runtime steps.

B. Offline Proactive Attack Modeling

Predictive Model Building. First, each Falco alert contains several properties with more detailed information about abnormal behavior. One of these properties is a MITRE ATT&CK tactic that indicates the MITRE ATT&CK tactic related to the alert (e.g., *Privilege Escalation* - MITRE ATT&CK TA0004). Therefore, the first sub-component is *Falco Log Collection and Processing*, which parses the alert log entries collected and aggregated by Falco from different Pods in order to extract the `mitre_<tactic_name>` tag. Then, the training dataset including the extracted tactics is sent to the second sub-component, i.e., *Predictive Model Learning*, which generates the predictive model. The predictive model represents the potential sequential transitions from one tactic to another as well as the relative probabilities of such transitions, based on historical data. This model is represented as a Bayesian network where nodes indicate MITRE ATT&CK tactics, edges indicate their transitions and are labeled with probabilities of transitions. To build the Bayesian network model, we use the *pgmpy* Python library [11].

Example 3.1: As an example, Fig. 3a shows three log entries collected by Falco for our attack scenario, with the MITRE ATT&CK tactic highlighted. First, `Alert #1` relates to a potential *Privilege Escalation* tactic employed by an attacker, followed by *Execution* then lastly *Persistence*. Such Falco log entries serve as the inputs for learning a Bayesian network model about the relationship between tactics and their probabilities. Fig. 3b shows an example of this model covering our attack scenario (highlighted in red) starting from *Privilege Escalation*, leading to *Execution* then *Persistence*, with a probability of 31% and 25%, respectively.

C. Runtime Attack Detection and Mitigation

Proactive Attack Prediction. This component first performs *Future Attack Prediction* using the predictive model and then performs *Resource Risk Calculation* by examining the Falco alert properties, as shown in Fig. 3c. As an example, we assign each parameter a value on scale from 1 to 5 according to the definition given below. Then we apply a risk formula (inspired by [12]) to estimate the risk associated with each Pod.

- **Priority Severity** $[1-5]$**:** Priority parameter inside each Falco alert (i.e., Critical, Error, Warning, Notice, Debug).
- **MITRE ATT&CK Tactic Severity** $[1-5]$**:** Average priority severity of all alerts for one tactic.
- **Context Severity** $[1-5]$**:** Assigned to parameters identified as malicious (e.g., user, command).
- **Next Tactic Probability** $[0-1]$**:** Probability of the next tactic according to the predictive model.
- **Asset Value** $[1-5]$**:** Assigned to each Pod based on the relative importance of hosted services and information.

Example 3.2: Considering the aforementioned attack scenario (Fig. 3), the first Falco alert received at runtime is a *Privilege Escalation* caused by CVE exploitation. Based on the model (Fig. 3b), the most likely next attack step is *Execution* as its edge shows the maximum probability. Therefore, as shown in Fig. 3c, the next attack step with its probability is identified, and the risk for the associated Pod is calculated.

Next, if the calculated risk exceeds a predefined threshold, which is a percentage of the maximum risk specified by the security admin based on the desired level of threat awareness, then the next component, *Non-disruptive Attack Mitigation*, will be triggered to perform Pod migration. Meanwhile, the risk calculation process will continue for the next alert.

Non-disruptive Attack Mitigation. The first step for performing non-disruptive mitigation is to find the optimal destination node for migration. The *Optimal Resource Selection* sub-component applies linear programming to find the optimal destination node. An optimal selected node follows three

Attack ID	Attack Campaign	CVE Number	Attack Features[a] PL	PA	INJ	IG	BD	MITRE ATT&CK Tactic Sequence
1	APT 3 [13]	2015-3113	*	*	*	*	*	Execution, Defense Evasion, Discovery, Defense Evasion, Lateral Movement
2	Spam campaign [14]	2017-11882		*	*	*	*	Discovery, Persistence, Execution, Defense Evasion, Defense Evasion, Lateral Movement, Exfiltration
3	APT 29 [15]	2021-36934	*	*	*	*	*	Persistence, Execution, Defense Evasion, Privilege Escalation, Defense Evasion, Discovery, Lateral Movement, Initial Access, Persistence, Privilege Escalation, Defense Evasion
4	Escape attack [16]	2021-3156				*		Privilege Escalation, Execution, Persistence
5	Simulated cryptominer spread [17]	2017-10271	*		*	*	*	Discovery, Execution, Persistence, Defense Evasion, Lateral Movement
6	Root data theft via memory corruption [18]	2020-14386			*	*	*	Discovery, Persistence, Privilege Escalation, Exfiltration, Persistence, Lateral Movement
7	SWC [19]	2015-5122	*		*	*	*	Discovery, Execution, Defense Evasion, Persistence
8	Targeted .gov phishing [20]	2015-5119	*		*	*	*	Discovery, Persistence, Lateral Movement, Exfiltration

TABLE I: Overview of simulated APT attacks and exploits for WARP dataset.
[a]PL: Phishing email link. PA: Phishing email attachment. INJ: Injection. IG: Information gathering. BD: Backdoor.

objectives as demonstrated in the following. First, it minimizes the migration of resources with higher asset values. This is to reduce the negative impact of any migration delay on more important resources. Moreover, it regroups the Pods during migration by the service they serve to avoid introducing additional communication overhead. Finally, it isolates the Pods under attack (i.e., minimize its co-located Pods and their combined asset value). At last, the *Pod Migration* sub-component migrates the selected Pods to the optimal node.

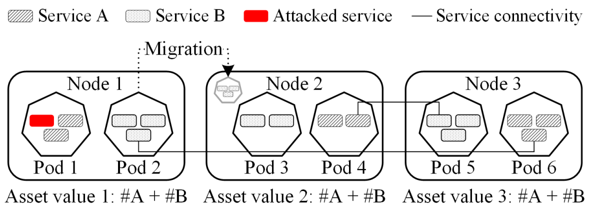


Fig. 4: An example of WARP optimal mitigation.

Example 3.3: Fig. 4 shows an example for finding the optimal destination node for migration. There are two different services running in different Pods in three nodes. We assume the asset value is calculated based on the services inside each Pod and service A has a higher asset value. As shown, the attacked Pod is co-located with Pod 2. Our goal is to minimize any chance for the attacker to perform lateral movement and move to other Pods. Therefore, either the attacked Pod or the co-located Pod should be migrated. Since Pod 2 has a lower asset value, the optimal migration is to isolate the attacked Pod in Node 1 and migrate Pod 2 to a less risky node. Node risk is associated with the asset value of its inside services. In this particular example, Node 2 is hosting fewer services and data, turning it into the less risky destination.

IV. Falco Alert Dataset Building

We build a relatively large Falco alert dataset for Kubernetes with both normal and APT attacks data to facilitate the learning of our predictive model (discussed in Section III-B) and to support future research. Our dataset is available on GitHub [21]. For the attack alerts, we apply CALDERA [22], an adversary emulation platform, developed by MITRE to mimic the attacks in a Kubernetes cluster in the form of MITRE ATT&CK tactic sequences (as summarized in Table I). For the normal alerts, we take advantage of the fact that Falco generates normal daily routine alerts even in the absence of any attack. We then label these alerts as “attack” or “normal”, respectively. Our dataset contains 231K alerts (including 2,314 attack alerts and 228,686 normal alerts).

Challenges. During building this dataset, we encounter several challenges as follows. As Falco reports alerts together for all resources in a cluster, we need to aggregate those alerts to reconstruct the attack steps. For this purpose, we first automatically group the alerts by resources (i.e., container ID) and then extract MITRE ATT&CK tactics’ property from the sequence of alerts. Additionally, the dataset is unbalanced with the number of normal alerts significantly higher than the attack alerts, as Falco generates a considerable number of similar alerts for system events. To obtain a realistic balanced dataset, we undersample the normal and oversample the attack alerts.

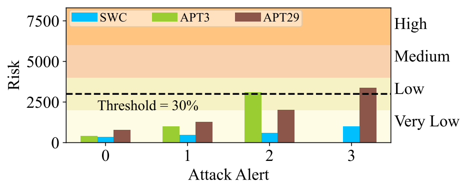
V. Experiments

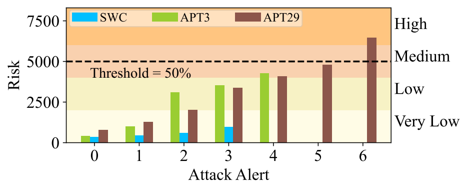
A. Implementation and Experimental Setup

WARP is implemented in Python and integrated with Kubernetes v1.20.2. For collecting runtime security alerts, we deploy Falco in a Kubernetes cluster using the official Helm deployment. Our Kubernetes cluster is hosted over 11 VMs running Lubuntu 20.04. One VM as master node (eight vCPUs and 32GB RAM), and ten other VMs as worker nodes (each four vCPUs and 8GB RAM). We apply VirtualBox6.1 as the hypervisor, and CRIU v0.27.0 [23] for Pod migration. The physical hardware of our cloud is composed of one physical rack-mount server with 2x Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz and 128GB of DDR4-2933 running Debian 10.

B. Experimental Results

Attack Progress. The first set of experiments (as shown in Fig. 5) evaluates the effectiveness of WARP for mitigating three APT attacks (i.e., SWC, APT 3, and APT 29) under different threshold risk values as those attacks progress (reflected by the number of received Falco alerts). The risk values are calculated using the method described in Section III-C upon receiving each Falco alert. For instance, APT 29 is completed with 10 alerts received, and the associated risk value stays in the range of *very low* during the first three alerts and reaches to *high* after Alert 7. As the figure shows, for different thresholds (30%, 50%, and 70%), indicated by horizontal dashed line, WARP will mitigate the attacks at different risk levels. A threshold of 30% (Fig. 5a) can mitigate all the attacks and prevent potential damages to the cluster. A higher threshold of 50% (shown in Fig. 5b) will mitigate APT


(a) Threshold=30%


(b) Threshold=50%

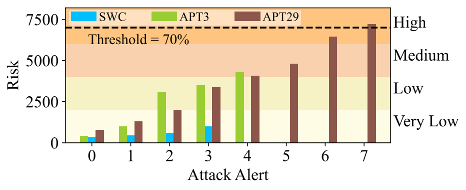

(c) Threshold=70%

Fig. 5: Attack progress.

29, while allowing APT 3 and SWC to successfully complete without being mitigated. Finally, a threshold of 70% (shown in Fig. 5c) will only mitigate APT 29 at the *high* risk level, and the other attacks will complete successfully. The implication of those results is that a security admin could generally choose a lower threshold to mitigate attacks more effectively (although it also implies a slightly higher migration delay, as shown in the following experiments).

Mitigation Delay. The second set of experiments is to measure the mitigation delay caused by WARP under different migration techniques and different service (container) sizes. We consider two popular techniques for container migration:

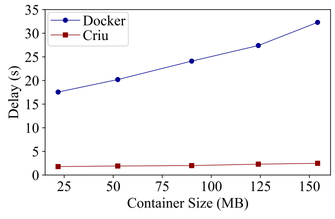
- **Docker:** Where the migration is performed by first stopping the pod and then copying it to the new location.
- **CRIU [23]:** A checkpoint is set on a Pod and restored in a new Pod in a new node from that checkpoint.

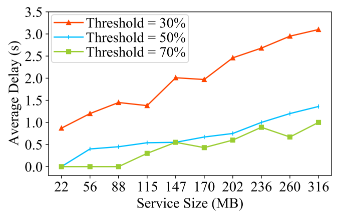
Fig. 6a shows the migration delay for both techniques for different container sizes (in MB). CRIU has an overall constant migration time of 0.94s for different sizes of containers, while Docker has exponentially growing delay (up to 32s for the largest container). Note that, except for the tightly coupled containers which are meant to be inside the same Pod, the Kubernetes best practice of running only one application or container per Pod will result in 0.87s migration delay on average by WARP. Fig. 6b depicts the impact on migration delay for ten different container services with different sizes (from 22 MB to 316 MB), with the *Average Migration Delay* ranging from 0.4s to 3.1s for different thresholds (30%, 50%, and 70%). Additionally, Fig. 6c measures the *Migration Frequency* (i.e., the expected number of delays per hour) for those ten different container services (i.e., *Services 1-10*) for different thresholds. Higher threshold values generally imply less frequent migration and hence less delay (e.g., *Services 1, 2, and 3*). Finally, Fig. 6d experiments on both *Migration Frequency* and *Average Migration Delay* for different threshold values; which shows both the migration frequency and delay decreases as threshold increases. The implication of those results is that, although there is a tradeoff between mitigation effectiveness (mentioned above) and migration delay, the impact on services is generally negligible and non-disruptive as migration preserves the network connection state.

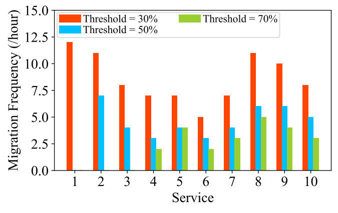
Mitigation Accuracy. The third set of experiments is to measure the accuracy of WARP in terms of mitigating attack-related alerts (i.e., true positives), non-attack alerts (false positives), and missed attack alerts (false negatives). Table II

Threshold		**Attack ID**						**Total per attack(%)**	**Total Dataset (%)**
		1	**2**	**3**	**4**	**5**	**6**		
30%	**Mitigated Attack Alert(%)**	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

TABLE II: WARP effectiveness per attack and dataset.


(a) Container migration delay


(b) Migration delay per service


(c) Delay frequency per hour

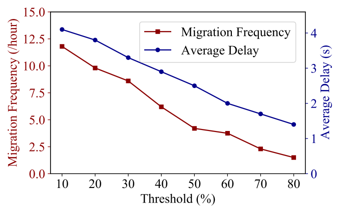
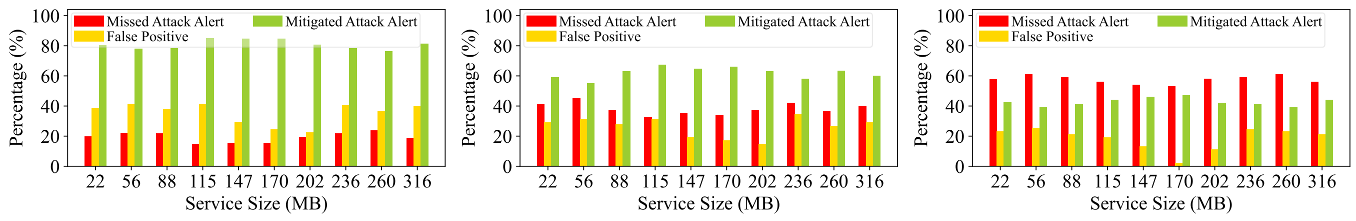

(d) Migration delay/frequency

Fig. 6: Migration delay and frequency for ten different sized services and various thresholds.

shows the numerical results for six of the simulated attacks, respectively, and for the whole dataset. As mitigated attack alert rate and missed attack alert rate are complementary (i.e., adding to 100%), we mention only mitigated attack alert rate in the Table II. Fig. 7 depicts that, for three selected thresholds (30%, 50%, and 70%), lower threshold values result in higher mitigated attack alert rates, lower missed attack alert rates, and higher false positive rates for different-sized services. The implication of those results is that a lower threshold is generally preferable thanks to the non-disruptive nature of migration (i.e., false positives are of less concern as they only mean slightly more delay).

VI. Related Work

Existing mitigation approaches (e.g., [4]–[6]) for container and cloud environments perform computationally expensive steps in advance and keep the final mitigation step for the attack to occur. However, as these approaches only start the mitigation upon critical events, they may still be too



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

Fig. 7: Threshold effectiveness.

late to prevent irreversible damages caused by the attacks, while our solution launches mitigation earlier to minimize such damages. There exist many attack detection and analysis approaches for containers, such as anomaly-based (e.g., Unicorn [24]), provenance-graph-based (e.g., ProvDetector [25]), and machine learning-based methods (e.g., [26]–[28]). Several works counter specific classes of attacks for containers, e.g., proactively preventing lateral movements by enabling least privilege policies [28]–[31] and applying graph optimization to audit logs [32]. Finally, there exist other approaches for containers (e.g., [12], [33]–[36]) which are either heuristic or rule-based, thus requiring an effort for maintaining and updating the heuristics or rules. Nonetheless, most existing works do not directly provide a general mitigation solution, and are thus complementary to our work.

VII. Conclusion and Future Work

In this paper, we proposed a proactive non-disruptive attack mitigation approach for Kubernetes clusters. We derived a predictive model from MITRE ATT&CK tactics, and utilized it to proactively migrate the risky resources to the optimal nodes. We also built a Falco alert dataset for Kubernetes attacks, implemented, and evaluated our solution for Kubernetes. In our future work, we intend to explore other mitigation methods to complement migration for better coverage.

Acknowledgment. We thank the anonymous reviewers for their valuable comments. 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] Y. Avrahami and S. Ben Hai. Kubernetes privilege escalation: Container escape == cluster admin? In *Black Hat USA*, 2022.
[3] Falco. https://falco.org/. [Accessed 30-3-2022].
[4] S. Majumdar et al. LeaPS: Learning-based proactive security auditing for clouds. In *ESORICS*. Springer, 2017.
[5] S. Majumdar et al. Learning probabilistic dependencies among events for proactive security auditing in clouds. In *Journal of Computer Security*, 2019.
[6] H. Kermabon-Bobinnec et al. Prospec: Proactive security policy enforcement for containers. In *ACM CODASPY*, 2022.
[7] MITRE Att&CK. https://attack.mitre.org/. [Accessed 30-3-2022].
[8] CNCF 2020 Survey Report. www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf, 2020. [Accessed 10-10-2022].
[9] CVE-2021-3156. https://nvd.nist.gov/vuln/detail/CVE-2021-3156/. [Accessed 30-3-2022].
[10] M. Li et al. MyCloud: supporting user-configured privacy protection in cloud computing. In *ACSAC*, 2013.
[11] A. Ankan et al. pgmpy: Probabilistic graphical models using python. In *SCIPY*. Citeseer, 2015.
[12] WU. Hassan et al. Tactical provenance analysis for endpoint detection and response systems. In *IEEE SP*, 2020.
[13] APT 3. https://attack.mitre.org/groups/G0022/. [Accessed 30-3-2022].
[14] 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/, 2017. [Accessed 30-3-2022].
[15] APT 29. https://attack.mitre.org/groups/G0016/. [Accessed 30-3-2022].
[16] Escape attack. Exploiting CVE-2021-3156. https://www.helpnetsecurity.com/2021/01/27/cve-2021-3156/. [Accessed 30-3-2022].
[17] 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].
[18] Root data theft. Kernel vulnerability exploiting cve-2020-14386. https://nvd.nist.gov/vuln/detail/CVE-2020-14386/. [Accessed 30-3-2022].
[19] Strategic web compromise. https://www.fireeye.com/blog/threat-research/2015/07/second_adobe_flashz0.html/. [Accessed 30-3-2022].
[20] Pierluigi Paganini. Phishing campaigns target us government agencies exploiting hacking team flaw cve-2015- 5119. https://securityaffairs.co/wordpress/38707/cyber-crime/phishing-cve-2015-5119.html/, 2015. [Accessed 30-3-2022].
[21] Falco alert dataset with APT attacks. https://github.com/simabagheri1/Falco-Alerts-Dataset-with-APT-attacks.
[22] CALDERA. https://caldera.mitre.org/. [Accessed 30-3-2022].
[23] CRIU. https://criu.org. [Accessed 30-3-2022].
[24] X. Han et al. Unicorn: Runtime provenance-based detector for advanced persistent threats. In *NDSS*, 2020.
[25] Q. Wang et al. You are what you do: Hunting stealthy malware via data provenance analysis. In *NDSS*, 2020.
[26] A. Alsaheel et al. ATLAS: A sequence-based learning approach for attack investigation. In *USENIX Security*, 2021.
[27] Y. Shen et al. ATTACK2VEC: Leveraging Temporal Word Embeddings to Understand the Evolution of Cyberattacks. In *USENIX Security*, 2019.
[28] F. Liu et al. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *ACM CCS*, 2019.
[29] B. Bowman et al. Detecting lateral movement in enterprise computer networks with unsupervised graph AI. In *RAID*, 2020.
[30] Q. Liu et al. Latte: Large-scale lateral movement detection. In *MILCOM*. IEEE, 2018.
[31] S. Freitas et al. D^2M: Dynamic defense and modeling of adversarial movement in networks. In *SDM*, 2020.
[32] Y. Kwon et al. MCI: Modeling-based causality inference in audit logging for attack investigation. In *NDSS*, volume 2, page 4, 2018.
[33] WU. Hassan et al. OmegaLog: High-fidelity attack investigation via transparent multi-layer log analysis. In *NDSS*, 2020.
[34] MN. Hossain et al. Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In *IEEE SP*, 2020.
[35] S. Milajerdi et al. Holmes: real-time APT detection through correlation of suspicious information flows. In *IEEE SP*, 2019.
[36] R. Yang et al. UIScope: Accurate, instrumentation-free, and visible attack investigation for GUI applications. In *NDSS*, 2020.