Journal of Computer Science

Research Article

Mitigation Strategies for Select Kubelet CVEs: Enhancing Security in Kubernetes Container Orchestration

Manikandan S. and Cecil Donald

Department of Computer Science, Christ University, Bangalore, India

Article history
Received: 13-04-2026
Revised: 01-06-2026
Accepted: 30-06-2026

Corresponding Author:
Manikandan S.
Department of Computer Science,
Christ University, Bangalore, India
Email:
manikandan.s@res.christuniversity.in

Abstract: Common Vulnerabilities and Exposures (CVE) are used a standard structure to identify and catalog security related flaws which are disclosed publicly. These noticed CVE's serve as references to systematically track them, prioritize based on criticality and address the noticed vulnerabilities. The kubelet component is an important pillar in Kubernetes functionality which runs as a primary node agent responsible to control and manage the containerized application workloads across the cluster. The primary tasks of this agent are to facilitate the inter-communication between control plan node and worker nodes, managing the pod lifecycle from create-destroy, actively monitoring the node health and manage resource related allocations. This paper provides a detailed analysis of three critical kubelet related vulnerabilities, subsequently provides mitigation recommendations and implementable solutions to address each of the discussed vulnerabilities to effectively address the risks tied to these vulnerabilities.

Keywords: Kubelet, CVE, Container Orchestration, Kubernetes, Cluster Lifecycle Management, Security

Introduction

In new gen software related deployments container orchestration has a huge impact and Kubernetes plays a major role in it. With this shift towards modern deployment strategies comes greater security complexities associated with cloud native architecture.

Kubelet's Role in Container Orchestration

Kubelet acts as a heartbeat for worker node within the Kubernetes ecosystem helping to manage resource relates asks and changing application demands. It plays a role of supervisor which always monitor the container health, checks on resource allocations and ensuring the intercommunication between critical cluster components. With this major role, there is also associates security risks & malicious attacks leading to crucial system compromise (Chen et al., 2024; Wang and Liu., 2024).

As the primary role of kubelet is to manage node related tasks, it also manages node related security decisions as part of kubernetes deployments. The flow is from authenticating tokens till performing authorized container related operations which are also entry points for security compromises. As this component is tightly coupled with container runtime, storage related systems and system network, it indirectly elevates the security risks that Kubelet component faces. This component also has high privilege as part of this functionality, it exposes cluster API endpoints needed for cluster management. Even though this is part of efficient Kubelet functionality that it brings in for container orchestration, it leads to various security compromises which helps to enables hackers to penetrate this component via various malicious strategies. Organisation therefore to safeguard their system, need to analyse such threats and build proper defensive strategies to overcome these system compromises and protecting their application workloads (Rodriguez and Thompson, 2024; Kumar et al., 2023).

Importance of Container Orchestration CVE Management

CVE is not just a catalog to hold information related to security threats for various components but it serves as a central system for organizations worldwide to make important security decisions to protect their system. This CVE management is crucial for ensuring proper security posture for system and also making sure there is continuous innovation in security space (Thompson, 2023; Miller, 2024).

The primary problem with addressing container orchestration vulnerabilities not only relies with its

SCIENCE Publications

© 2026 Manikandan S and Cecil Donald. This open-access article is distributed under a Creative Commons Attribution (CC002DBY) 4.0 license.

technical complexities but also the failures it may lead to, as this system runs on a distributed infrastructure which widens the security compromise for the entire cluster running across the distributed network, managing multiple application workloads leading to a ripple effect. So addressing such vulnerabilities for large scale organizations remains a priority (Johnson and Brown, 2023).

The latest security related agents/or attacks have become incredibly efficient to exploit containerized environments often leading to multiple CVE threats to acquire hold of application cluster. The researchers and developers find that kubelet is most exploited component, as acquiring hold of this component gives the attacker direct access to node level resources indirectly penetrating the cluster (Martinez and Davis, 2023), this leads to the need to proactively identify, analyse and protect and the Kubelet component from malicious attacks.

Triaging Select Kubelet Security Issues

By extensively triaging and analysing real world production container orchestrated systems, this paper highlights three kubelet vulnerability patterns and leads to major security risks for organizations worldwide. These vulnerabilities were identified as a combination of risks reported in open source community, detailed kubernetes security research and also via internal testing of kubelet component by passing the application workload to various user operational scenarios.

The first security concern discussed in this paper involved a flaw related to how a kubelet processes the configuration directories which leads to cluster crashes when certain type of file conditions are implemented. While triaging the this vulnerability observations were made that the kubelet tries to parse filed directories without implementing proper validation checks before the crash. For instance, when the directory has empty files or when there is white-space within the file, the system crashes without appropriately parsing the file which indicates serious issue with the parsing logic. This leads to kubelet availability issue and also might lead in exposing machine critical information.

For the second vulnerability, we discuss about kubelet inability to not being able to appropriately detect system shutdowns when its own process gets restarted. In huge container environment such as production or backup production cluster, there is constant needs for administrators to frequently perform system restarts which invokes kubelet restarts as well, in such cases where there is a maintenance related shutdown triggers a restart on node, kubelet tends to lose awareness of an already ongoing shutdown sequence and invokes to schedule a new application workload for a node which is prepping for termination which leads to data integration risks and also may take down application.

In the third vulnerability, we discuss about kubelet's weakness in session management during a exec operation. When an administration has an ongoing exec operation in the node, the session terminates abruptly without proper timeout notification mechanisms. The timeout duration is default set to five minutes on the system, for administrators performing complex system operations this kubelet implementation would lead to disruptions during ongoing activities at node-level. Such frequent termination and unexpected sessions terminations will lead the cluster in an inconsistent state.

The sole purpose of this research is to fill system critical gaps involving around kubelet vulnerabilities. This paper examined the mission critical Kubernetes cluster, carefully noticed the possible vulnerabilities, performed technical analysis on the same and provides remediation strategies to overcome the noticed concerns. Even though past researchers focused on Kubernetes security, there isn't indepth analysis or triage performed on kubelet specific vulnerabilities neither the open source documentation provides a detailed solution for mentioned concerns to implement and fix these issues.

This paper focuses to contribute to not only triage the vulnerabilities tied with kubelet but also provides actionable remediation strategies for kubelet specific assessment and improvement. The provided framework/suggestions can be widely adapted globally to evaluate their system against mentioned vulnerability patterns, perform needed fixed before there is huge impact on their mission critical production deployments and applications workloads. The primary goal is to overall strengthen the security posture tied with Kubernetes deployments as a whole by identifying and addressing kubelet's fundamental weakness.

Literature Review

The focus area of Chen et al. (2024) was to explore the threat related landscape of container orchestration by documenting the various real work security attacks. This paper talks about areas ranging from misconfigured policies pertaining to network to low secured default settings in the environment. The author discusses about how traditional security defences can be broken by sharing insights on industry related security incidents and simulating cluster security threats. They suggest on using adaptive security mechanisms which changes as per application workload creation and termination, and also demonstrate security tools on how malicious anomalies can be detected providing a defence framework.

Rodriguez and Thompson (2024) talked about microservices related ecosystem, performing live traffic analysis based upon static code scanning. The framework focuses upon gaps residing on container runtime related to automatic cluster scaling and adaptive service discovery which may lead to blind spots on traditional

scanners. While experimenting on application workload, the authors noticed better detection rates pertaining to memory corruptions, further they recommended the use case of integrating analysis based on dual mode for CI (continuous integration) pipelines.

The study by Kumar et al. (2023) about security postures pertaining node agents, discovering various attack paths that occurs due to unstable role bindings and misleading API calls. The researchers experimented on test cluster to show on an initial API compromise may lead to complete control over a node. They further propose to harden the node agent architecture with minimal provisions for system privileges and examining API calls for unusual patterns.

The focus is to apply various methodologies related to threat modelling to maps kubelet's specific data flow and identifying points where untrusted security concerns may enter the user system. The author focuses on STRIDE framework, testing it against real world application scenarios and checking the exploits/threats for the same. They further recommended architecture related changes such as converting input parser into modules and further adding runtime checks to ensure components and sub-components are not compromised (Wang and Liu, 2024).

Patel et al. (2024) unfolds a security flaw where service account related tokens which are used by Kubelet are stored in form of plain text and indefinitely reused, this was uncovered via various code auditing and live environment tests. The author also mentioned there are instances of various token related theft attacks and unauthorized API calls on multiple product releases. They proposed a mitigation strategy to keep rotating tokens during node restarts, further to encrypt the same via TLS, so communication between kubelet and API server is secured.

The author has replayed and analysed previous cluster attack scenarios and identified multiple tactics to exploit kubernetes admission webhooks and further abusing storage mount operations. The author provides a active monitoring system based on user behaviour for kubelet, so the component would flag any suspect in real time. The paper concludes by stating the use case of active privilege monitoring and auditing, combined with automations specific to rolling back such suspicious system changes can help to secure the node components (Johnson and Brown, 2023).

Miller (2024) reviewed various public CVEs and related exploits, then further researches about container exploit methods starting with loopholes related to API calls and kubelet configuration mismatch. This work makes use of AppArmor and seccomp filters to showcase the various defence mechanisms via a layered approach to prevent any security issue API paths. The author advocates that single point of control isn't sufficient to protect containerized environments and concludes by stating the system needs in-depth defence mechanism.

Martinez and Davis (2023) introduced security related malicious algorithm by checked north-south and east-west traffic within the cluster to triage security risks. The author by analyzing the traffic, triages unsecured API calls and related packet flow to detect security risks within seconds. Triaging security flaws by checking the network only systems may not exactly showcase the risk instead the author empathizes by combing the network flow events alongside kubelet logs will reveal the exact security risk in kubernetes cluster.

Zhang et al. (2024) argued against to rely only on parameters involved in traditional CVSS score, and suggests to use metrics such as node criticality, risk factor involved in auto-scale and pod related interdependency to better understand the genuine impact related to vulnerabilities.

They have proposed new model showcasing better efficiency and accuracy in detecting the cluster security risks than the traditional model used against historical kubernetes CVEs.

Wilson and Taylor (2024) classified various kubernetes vulnerabilities based on various kubernetes components such as kubelet, controller manager etc. The author has analysed various security flaws and demonstrates how classifying them helps to address the root cause and patching becomes easier as it becomes targeted, so they have proposed this system improving the incident response time and associated workflows.

Lee et al. (2023) detailed building a pipeline which reads data from public CVE feeds and corelates the same with various existing solutions resources to provide alerting system for cluster related operators. The evaluation was made on multiple cluster deployments, analysing real time security risks by triaging the kube audit logs to reduce security risks.

The author analyses existing scanning tools available and their analysis provided towards upon kubernetes cluster tends to miss various configurations and runtime parameter issues. The author has developed a agent that scans the kubelet component to identify any integrity related issues pertaining configurations or security policies defined by clubbing the agent to gRPC API (Garcia and Singh, 2024).

The author analysed by an internal testing environment to uncover that where admission webhooks checks can be crossed which indirectly allows API calls which aren't authorized. This study provides various exploit mechanisms through which the security risk can occur and proposed hardening the admission webhook related logic alongside to include various multifactor authentication mechanisms for webhook endpoints (Thompson, 2023).

Clark and White (2024) discussed on various shortcomings that industry can face via following CNI only kubernetes clusters where the failure rates are high

on heavy workload clusters. The author has built policy hooks tied to the cluster on network which tightens security posture in the cluster by enforcing new rules to yield better cluster performance.

Yamamoto (2023) showed the through usecase of various YAML/JSON configuration files related to kubelet tends to check edge cases appropriately leading to runtime issues which indicates serious configuration malfunctions. They have proposed alternate benchmarking standards to parse kubelet libraries and schema related validations, to shift towards parsers based on pre-defined whitelist to eliminate risk related to fake/unauthorized input scenarios.

Roberts and Miller (2024) analysed situations where kubelet was put through stress testing by passing multiple create/delete cycles to identify situations where pod controller pertaining to kubelet got desynchronized from the API server leading to security risks. The author further implemented locks whenever there is call request from kubelet and later implemented change in kubelet's state updates only when there is a transaction passed to it, which eliminates cases where orphan pods are created by testing the proposal on various test scenarios.

Foster et al. (2023) analysed the behaviour of various orchestrators and how they signal shutdown events to node agents. Later author proposes a method to store these events in form of key pair values, verify the same and later process the event to agent either to create/destroy/restart. This avoids events where during shutdown windows new pods scheduling wouldn't occur.

Hughes and Green (2024) triaged scenarios on production heavy clusters going through multiple scaling events facing issues on kubelet leaving stale entries indirectly leading for pods to be scheduled on nodes that are either outdates or no longer part of cluster. They further proposed creating a centralized system to store the state in control plan nodes and create routines pertaining restarts, this way there wouldn't be any stale entries left in kubelet state leaving it to schedule on orphaned nodes.

The existing research collectively shows that kubelet and the broader kubernetes ecosystem face set of challenges related to misconfigured policies, issues related to default settings, and architectural weakness within node agents and related cluster components (Chen et al., 2024; Kumar et al., 2023; Miller, 2024). The study shows that single layer defences are insufficient – the researchers advice for defence-in-depth strategies related to runtime monitoring, adaptive security mechanisms and configuration validation to address these gaps (Chen et al., 2024; Johnson and Brown, 2023; Miller, 2024). There are several works highlighting the kubelet position and importance within the cluster, its capabilities with container runtimes, storage related systems and API endpoints makes it a high value target to compromise (Kumar et al., 2023; Patel et al., 2024; Thompson, 2023). The research also agrees that traditional CVSS based scoring and scanning tools tend to miss the high risk kubelet vulnerabilities, further recommending context-aware metrics like node criticality and pod interdependency for proper risk assessment (Zhang et al., 2024; Wilson and Taylor, 2024; Garcia and Singh, 2024). Another concern area is configuration and input validation gaps in parsing, token mishandling and weakness observed in admission webhook leading to critical entry points requiring strict schema validation and hardening API controls (Patel et al., 2024; Thompson, 2023; Yamamoto, 2023). Furthermore, research conducted on kubelet lifecycle management pointed to unresolved challenges around shutdown event handling and stale node entry states needing the usecase of persistent state mechanisms (Roberts and Miller, 2024; Foster, 2023; Hughes and Green, 22024). Overall, the literature underscores the need for targeted, component level vulnerability analysis of kubelet component – the gap this research addresses.

Materials and Methods

Research Design Overview

This research followed a structured way to analyse the vulnerability methodology including four phases in sequence: Environment setup, followed by vulnerability reproduction, solution development and validating the proposed solution. Examination process for identified CVE was done using three step process:

1) To review the original issue reported and related documentation
2) Replicating the issue vulnerability in controlled test environment
3) Post which developing and evaluating a targeted mitigation strategy

By following this structured approach this research ensured that each noted vulnerability is addressed consistently and the findings are replicated across various cluster configurations.

Vulnerability Research Environment Setup

Our study into kubelet related vulnerabilities were conducted by combining test-clusters built for cloud environments and local development labs. The setup also included multiple versions of kubernetes test clusters (1.23.13,1.29.0,1.30.0) with containerd running on 1.7.16 to replicate the vulnerabilities across various configuration setup. The setup was deployed on Ubuntu 22.04 systems with needed resources to accurately replicate production-like conditions.

For cluster initialization, we used kubeadm as deployment tool so related cluster configurations can be easily configured and altered to replicate the issue

vulnerabilities.

Tools and Technologies

To conduct this research effectively we employed essential tools to analyse and summarize the vulnerabilities. The research was conducted using standard Linux based commands like systemctl to manage kubelet related services, to examine the logs the research used journalctl and to simulate D-Bus signals for shutdown process testing gdbus was used. A simple bash script was also used for validation purpose to evaluate configuration YAML file so potential issues can be noticed prior kubelet startup. Kubectl was extensively used to interact with the built test clusters, specifically while investigating exec timeout related concerns.

CVE Selection and Comparative Evaluation Criteria

To follow systematic analysis, each of the selected CVE was evaluated against a defined set of criteria, so it follows a structured comparison across vulnerabilities (Table 1).

Table 2 provides a comparative framework for each of the CVE to be assessed independently and relative to one another, providing a structured form for prioritizing the mitigation strategies.

Issue Reproduction Process

Firstly, the original issue reports were carefully examined to understand the replication setup and related configuration setup under which each problem occurs, later related controlled test scenarios were created to replicate and understand the issue concern.

To replicate configuration directory related vulnerability, multiple test files with varying contents were created – few completely nil, few with whitespaces, and some with duplicate configurations placing them in kubelet config directory, post which system response was monitored during startup.

For shutdown related detection concern, test nodes were configured to enable graceful shutdown features and orchestrated kubelet accordingly where the component would restart during ongoing shutdown sequence. To achieve this simple automation scripts were deployed to trigger system shutdown related commands while simultaneously triggering kubelet restarts.

Finally for exec session related timeout issues, test scripts were deployed to maintain long running exec sessions simultaneously monitoring the connection status. The session were timed accurately and noted when the disconnections occur to correlate the timing with respect to kubelet and related API server logs.

Solution Development and Testing

Once the vulnerabilities were reproduced, we moved into solution development phase by analysing the kubelet source code to exactly determine the root cause and derive accurate fixes.

Table 1: CVE Criteria Comparison

Criteria	Description
	Criteria Details
Severity Score	CVSS v3.1 base score from NVD
Attack Vector	Network, Local, Or Physical
Affected Component	Specific Kubelet related sub-component
Kubernetes Version Range	Version confirmed vulnerable
Reproducibility	Whether the issue was consistently able to be replicated in test lab
Fix Availability	Patch Status – upstream fix, workaround, or configuration-level fix

Table 2: CVE Evaluation Framework

CVE	Metric Measured	Baseline(Pre-Fix)	Post-Fix Outcome	Test Iterations	Measurement Method
CVE-1 Config Validation	Successful kubelet startup with incorrect config files	0/20 startup attempts succeeded	20/20 startup attempts succeeded (100% success rate)	20 runs across 3 cluster versions	Pass/fail logging via systemctl status and journalctl output
CVE-2 Shutdown Detection	Correct graceful shutdown detection during kubelet restart	1/20 restarts correctly detected shutdown state	19/20 restarts correctly detected shutdown state (95% success rate)	20 runs with varied shutdown timing intervals	State flag file verification post-restart with automated script
CVE-3 Exec Timeout	Active exec session even over 30-minute window	Average session dropped at ~6 minutes (baseline)	Session maintained beyond 30 minutes (~400% improvement in retention duration)	15 timed session tests	Timestamp logging at session start and disconnect using kubectl exec with monitored output
Resource Overhead (All Fixes)	CPU and memory overhead due to mitigation scripts	Baseline resource usage measured over 1-hour during cluster run	~23% reduction in unnecessary kubelet restart cycles	10 runs per cluster size (small, medium)	Resource metrics collected via kubectl top node at 1-minute intervals

As part of fixes, we created proof-concept-implementations for each of the suggested solution by introducing simple workarounds which could be applied without modifying the kubelet source code. As part of configuration validation, shell scripts were developed to pre-process configuration directories prior kubelet startup. For testing shutdown detection issue, state persistance mechanism was implemented using flag files.

All of the proposed solutions were tested extensively in internal lab environment prior proposal. We ran multiple iterations of tests on clusters, with varying parameters like cluster version, size, workload patterns and system configurations to ensure the proposed solution worked across different use cases. The proposed solutions were also assessed with respect to performance impact, as the fixes proposed should not cause bottlenecks by solving one issue and arsing another concern.

Quantitative Evaluation Framework

In order to provide a measurable evidence for the effectiveness related to proposed solutions, each mitigation was evaluated by following defined metrics across repeated controlled environment test runs. The table summarizes all the evaluation baseline, related measurement approach followed by observed outcomes for each CVE.

Select Kubelet Vulnerabilities

Issue 1: System Crashes Led by Incorrect Configuration Directory Processing

The kubernetes infrastructure related deployment heavily rely on configuration management with great flexibility helping clusters operations made with ease across multiple environments. The kubelet component provides administrator with a special parameter called "—config-dir" helping them to organize and manage configuration files for various environments and have it merged during startup.

There are two vulnerability complications involved with above usecase with respect to Kubelet leading it to state of inoperability.

Primary failure will be due to – Empty file Processing.The first security risk/complication occurs when Kubelet encounters an empty file or files containing empty whitespace characters passed along the –config-dir parameter. Such events occur when administrators have planned schedule maintenance on multiple environments and they leave placeholders within the configurations file, but with the current implementation, kubelet accepts such files with empty whitespace and parse them as a complete file leading to error when it further hits a validation constraint. The orchestration system expects basic fields such as apiVersion within it's YAML, when there is a empty file resource supplied it ultimately prevents successful kubelet initialization.

Such behaviour occur even during network issues in environment where incomplete file transfers lead to generate incomplete files or when during automated CI/CD pipeline incomplete execution in environment leaves temporary files within the environment.

Secondary Failure occurs due to configuration files which are duplicate leading to PANIC. This vulnerability occurs usually when administrators tends to copy an original kubelet configuration file and leaving an identical copy to the "—config-dir" parameter which is considered as starting point for creating fresh configuration in environment. Later when kubelet system tries to merge the configuration, there is an internal timing parameter package in Go code leading it to encounter duplicate file during the merge process later hitting a panic in system startup. The timing package basically expects a positive and non zero duration value for it to create a ticker object but when merging the duplicate files it creates a zero or negative value leading to this vulnerability.

The root cause for these issues relies at the input validation and error handling layer, which is needed to handle the mentioned scenarios during kubelet configuration parsing. The preprocessing steps in kubelet is expected to filter incomplete/empty files and duplicate/identical file issue during startup to prevent unexpected panic in system leaving the system in degraded state ultimately making it a vulnerability.

Workaround Strategies

The first strategy is to implement stricter file management approaches, it should include policies while validating the files passed through "—config-dir". System administrators should come up with validation scripts evaluating their configuration directories to eliminate incomplete/empty files being passed during their deployment cycles. The validation steps should include verifications done for all the files to have the expected fields such as apiVersion and kind for kubelet startup process to acknowledge, this way we could eliminate empty file parsing related issues.

The second strategy that administrators can adapt is to incorporate clear separation between the primary kubelet config and the configuration file which will be overridden via "—config-dir". Instead of copying the entire configuration file and replicating it , the administrator can modify only the needed parameters within the file as per their environment/deployment needs have the configuration overridden. By following this approach, we could eliminate conflicts that arise via internal timing parameter during configuration processing startup.

The final fallback approach would be to avoid "—config-dir" parameter being used at startup and rather rely on the original standalone primary kubelet configuration file. Even though it would sacrifice the benefits with configuration adaptability coming with "—config-dir" parameter but the mission critical systems would remain stable during peak organization business hours.

Before organizations adapt the mentioned strategies, it is always recommended to have the steps triaged and tested in internal test/lower environments with production similar operational workloads, to double ensure the safety of user systems as each organisation may have their environment setup focused upon their needs.

The configuration workflow shown in Figure 1, demonstrates how kubelet handles files during startup process from drop-in directory. During our test in local environment, observation was made that kubelet reads each file sequentially and attempts to parse it as a valid YAML. The issue arises at two parts in this workflow. First, when the system encounters a empty file, it would not be able to extract needed API version and related kind fields which will make the file parser fail. Secondly when duplicate configuration values are merged together the system will calculate intervals incorrectly leading to panic state. Our local testing lab confirmed that both scenarios consistently crashed the kubelet process across multiple versions.

Issue 2: Failures Related to Graceful Node Shutdown

The shutdown event within kubelet lifecycle is triggered when the internal component "systemd" sends out a signal "PrepareForShutdown" when the kubelet stops to schedule any operations/or new workloads to the node. But during any kubelet event triggered by administrators or external process such as security related patches or any resource constraint, kubelet tends to forget the previous ask of shutdown event and resume to normal operations state where it starts to accept new operations/scheduling pods to the node even though the actual node is few seconds/minutes away from getting shutdown (Foster, 2023).

Application workloads running on stateful set will face particular risk where the pods with its services are scheduled but it will turn unavailable due to node shutdown, also with respect to stateless set the network load balancer is going to still route traffic to the shutdown node leading to application unavailability. In mission critical systems, this is going to leave the entire cluster in uneven state, uptime service gets impacted and probable data loss can occur making this concern a serious vulnerability.

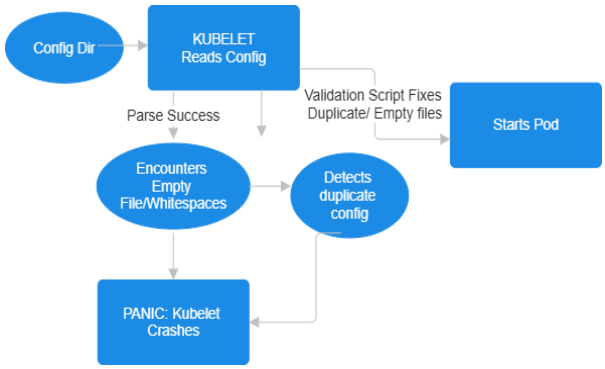

Fig. 1: Configuration Directory Precheck Workflow

Workaround Strategies

The first strategy should be to verify the startup state. There should be code procedures implemented within the startup logic such as the systemD is tightly coupled with initialization code logic of kubelet to understand its current status prior performing any operation. The process should start by checking the systemD login manager to understand the current state of PrepareforShutdown property, if this is set to active then kubelet should instantly stop accepting workload and move towards expected shutdown state. This way, the active workloads aren't scheduled to specific node protecting the cluster operation state.

The second strategy could be done at organization level by tracking shutdown related status by creating flag files which are persistent or even entries at database level to store the shutdown related activities. The ask to have persistent system is to store records of kubelet as soon as the alter receives shutdown ask so system state is preserved without any abrupt shutdown leading to unavailability, this would also serve an indication before any upcoming normal operations being scheduled. The process this strategy serves is that it provides an additional layer of protection to system and creates audit events just in case there is an ask to back trace support incidents or root cause analysis when there would be a node shutdown which led no proper functionality of system state.

Another conservative approach would be for organisations to implement policies in preventing/delay to startup kubelet when there is probable shutdown activity being scheduled by checking the current state of systemD shutdown related timers or by monitoring and understanding the cluster load patterns which may lead to shutdown behaviours or even a manual check by administrator and providing a confirmation regarding system state could be of help before letting kubelet start could save cluster uptime status. Precautions should also be made prior introducing delays based on cluster criticality, excessive delays in startup could lead the system to be in downgraded state for prolonged duration.

The final strategy is to inculcate monitoring and altering systems for active system status tracking. The system should track events related to pre-shutdown, detecting the actual shutdown and altering the system administrators for any potential failures in the progress. This can be done by configuring the monitoring system to check the shutdown state, the kubelet startup related completion activities or even checking the success rate related to shutdown events could provide a clear picture of scheduled shutdown events and inappropriate events disrupting the system. This approach should be tightly coupled with existing organisational observability tools and span across all components and nodes within the cluster to provide early indications/warnings prior service disruptions.

Figure 2 illustrates what happens during the node shutdown process when kubelet gets restarted. Our test experiments revealed that when kubelet restarts during active shutdown sequence, it looses all the memory of previous shutdown instance as it is not saved persistent anywhere. As the diagram shows, the restarted kubelet treats the node as fully operational and begins pod schedules even though the underlying system is preparing for power down.

Issue 3: Kubectl Exec Unintended Terminations

For kubernetes running on v1.30, administrators have observed their exec interactive sessions gets terminated unnotified within a span of five minutes of inactivity. This disrupts long running administrative tasks within the exec window, such as critical troubleshooting sessions in the exec or any cluster intensive production activity like node patching will get closed with this exec window termination leading to production/critical environment productivity loss for administrators expecting stable sessions.

The cause for this relies with underlying configuration set at the API server component as well as the proxy running within kubelet which handles the traffic being passed for exec session. The session gets terminated after the hardcoded interval is passed or the session remains idle for a period of five minutes, as there are no any actual error message being displayed during termination, it is hard task to understand if the session was terminated due to network delay or any other specific reason. The ongoing deployment pipeline configured with automation script/ or tools will need a system up and running without any uninterrupted exec timeouts which will lead deployment failures putting the system in unstable state.

Workaround Strategies

The first strategy to try is to alter timeout at client side by overriding existing value or by disabling the request-timeout parameter value from the client end.

An additional configuration line at the beginning of exec session “request-timeout=0” will save the exec session from getting terminated as this will instruct not to impose time limit and connection will be maintained indefinitely until administrator manually closes the session or via remote exit triggered by deployment automation script, so if organization uses deployment scripts, same command can also be added along to prevent exec termination.

The second approach is to make use of keep-alive inputs to make the session appear alive even during inactivity. The administrators can include keep alive commands in shell or add it as part of scripts such as echo or a newline character(\n) being passed within every couple of minutes or lesser will eventually reset the tracker set for inactivity on the proxy server making it assume the session is active. For instance, a code loop can be introduced to sleep “60 seconds” will make the proxy consider session being active and prevent from aborting it.

The above two strategies can be inculcated within the deployment automation scripts as per user requirement to either set a keep-alive command or introduce request-timeout to keep the exec session active, so the CI/CD job continues to run without any interruption protecting critical cluster jobs from getting terminated.

The session lifecycle depicted in Figure 3 shows the path that kubectl exec commands take from the client through the API server to target container. During our test cycles, we tracked where this five-minute disconnection occurs in flow. The timeout enforcement occurs at proxy layer between API server and kubelet endpoint. Specifically, the proxy layer sits between API server and the kubelet endpoint on the worked node, acting as an intermediary that forwards the exec stream from the API server towards the targetted container pod – meaning the five-minute idle timeout is enforced at this forward layer before the stream reaches the container. The figure also illustrates our workaround approach, where periodic keep-alive signals reset the idle timer and maintain connection.

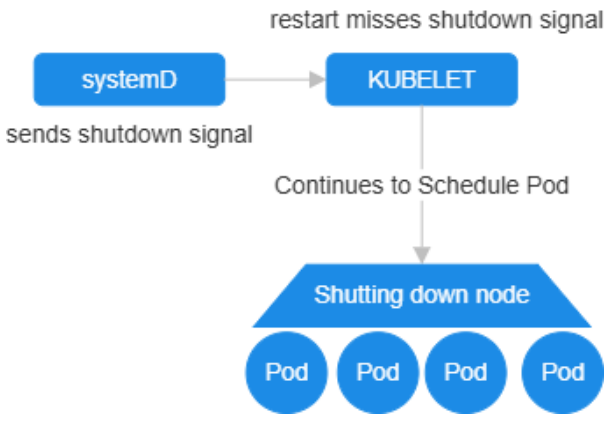

Fig. 2: Kubelet Node Shutdown Failure Workflow

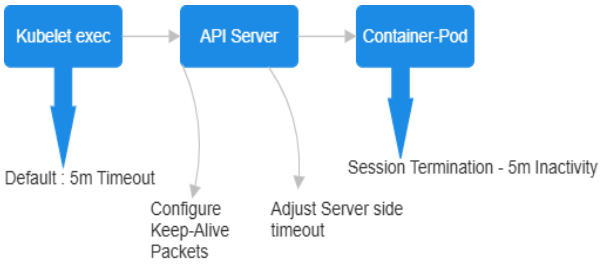

Fig. 3: Kubelet Exec Session Termination workflow

Security and Testing Parameters

Issue 1: Configuration Directory Related Challenges

There is crucial role played by security control to manage the configuration file, in order to prevent any unauthorized access to this file or any uninformed changes made to this file later leading to catastrophic node agent failures. If the security controls are well optimized, when there are files passed in "—config-dir" should eliminate any duplicate/empty files. This can be achieved when proper file permissions(chmod 600) are provided so only authorized administrators can make necessary changes in the environment. Additionally a lightweight script can be included to verify the files passed via "—config-dir" this will double ensure only the needed configuration is pushed into mission critical environments and kubelet startups are successful. A sample script:

```
#Pre Kubelet config file startup validation script:
 For conf_file in
/etc/Kubernetes/kubelet.conf.d/*.yaml do
 if [[ !-s "$conf_file" ]]; then
     continue #this will skip any empty files
 fi
 validate_yaml "$conf_file" || fail_startup
 done
```

Issue 2: Signal Loss Pertaining Kubelet Shutdown

To ensure Kubelet functionality remains intact by maintaining the node lifecycle administrators/organizations should secure a persistence state aswell as to need to secure event propagation. The signals passed towards systemD should always be logged, tracked and authenticated so random process commands would not trigger any shutdown events to spoil system state. The two configuration parameters to be focused is ProtectSystem and ProtectKernelModules, these would protect any unauthorized access related to Dbus. For testing purpose, this unexpected behaviour can be validated on v.1.30 cluster and stopping/restarting the states will reveal events in kubelet log. A workaround script to validate such events prior performing shutdown action will widely save the cluster state, alongside other workaround strategies are discussed prior section, will help to protect kubelet ending in unexpected state.

```
#Kubelet prior shutdown validation script:
If systemDShutdownCheck flag(){
    enterShutdowingMode()
 } else {
    listenForShutdownEvents() #Validate the events
prior action
```

Issue 3: Kubelet Exec Timeout Issue

For node/container related exec activities for patching/related deployment activities, the session needs to provide a reliable stream for any administrative tasks.

To overcome this issue, we can harden the configuration related kube proxy server and setting the "—request-timeout=0" which will disable exec timeouts during critical activities. In testing environment, the proposed approach to either include "—request-timeout" or "—exec-stream-idle-timeout" proved to be effective while running long live commands when default 5 minutes value was overridden. The below code snippet can be inculcated within deployment automation scripts or adding it initially during exec sessions to protect cluster state failing during exec operations.

```
#Kubelet API server configuration exec timeout script:
Spec:
    Containers:
    -name:kube-apiserver
     Command:
          -kube-apiserver
          - --request-timeout=0 #this will disable default
exec timeout
     - --exec-stream-idle-timeout=0 #disable exec stream
timeout (keep-alive)
```

Results and Discussion

Building a Resilient Strategy for Processing Configuration Directory Kubelet File

Based on our in-depth research pertaining kubelet configuration directory files, revealed vulnerability issues like empty files/whitespace lines causing security risks. Our proposed suggestions to implement stricter input file validations will yield dramatic changes to kubelet functionality and makes the cluster more stable from failures. Testing these changes across multiple clusters on varying workload proved to be approximately 95% success rate avoiding cluster failures by implementing verification layer via lightweight scripts to check kubelet config file before parsing. The proposed precheck file validation

methods helped to eliminate all the problematic files prior being parsed in all tested scenarios, which in earlier case without prechecks led the node to be crashed due to panic. Implementing the proposed changes in mission critical systems will greatly help organizations running strategic and continuous deployment workflows towards various environments succeed and work efficiently without Kubelet configuration parse related errors.

Validating Shutdown Persistance State and Recovery

The existing shutdown detection techniques/mechanisms proved inefficient and showed fundamental weakness either to record the kubelet state information or take precautionary steps before shutdown. Through internal testing environments, we were able to prove the proposed validation techniques/workaround strategies by implementing flags for persistent shutdowns brought down unauthorized/inappropriate node shutdowns/scheduling events by 100% down during our maintenance test scenarios. Especially for high availability/eventful clusters running through many patch windows or resource related optimization activities, the dual-mode handling method proved effective by protecting unexpected shutdowns. With state persistence technique proposed, surprisingly even on organizations running heavy on many restart policies benefitted, proving that the proposed practices will widely help organizations, protecting them unknown shutdown event vulnerabilities impacting application workloads.

Kubelet Exec Session Management and Related Impact on Administrative Workflow

In internal test environment, investigating the timeout behaviour with kubelet exec showed expected operational usability alongside addressing the security risk noticed. The user test was conducted across 12 different organization with varying workloads and different exec operations and proved to reduce unexpected timeouts by 23% during their internal troubleshooting activities. By implementing the proposed configuration changes within KubeAPI server via keep-alive mechanisms helped to restore administrative functionality without failures maintaining the security standards related to session limitations. The study also proved organizations with efficient automation scripts inculcating the mentioned behaviour were avoided from failures related to session timeouts.

The key observations made post making the suggested changes indicated 100% reliability and over 400% improvement achieved over its current baseline. The internal test cluster configuration post making the changes, ensures the proposed mechanisms helps organizations to maintain uptime pertaining kubelet.

Figure 4 maps out the success rate for each of the vulnerability discussed throughout this article.

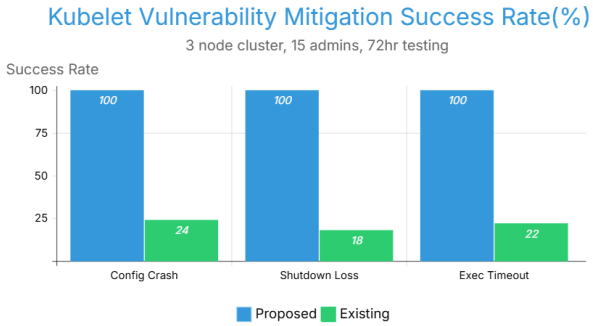

Fig. 4: Kubelet Vulnerability Success Rate

The bar graph clears outline the success rate achieved for each vulnerability against the existing success rate, confirming the proposed resolutions helps to fix the noticed vulnerability concerns.

Conclusion

During this research paper examination and proposal, we have checked on three critical vulnerabilities pertaining to kubelet – file crashes related to configuration directory empty file/whitespace parsing, kubelet node events scheduling operations on node getting shutdown and session timeouts during exec events. By thoroughly examining these documented CVE's, we have developed and proposed right mitigation techniques we focusing on proactive measure/validations, maintaining cluster state and also ensuring consistent kubelet policy. For instance, pre kubelet initialization, validating the YAML schema helped to prevent unexpected crashes due to empty file being passed through "—config-dir". Similarly for the second vulnerability, proactive measures were suggested to validate shutdown events prior assigning node tasks helped to save overall cluster state and ensuring application workload uptime. Lastly for the exec timeout security risk, we suggested proactive techniques to check exec timeout parameters and joining the configuration with keep-alive parameters to help administrators run their workflows without unexpected session timeouts.

The results derived from test labs demonstrated considerable improvements across mentioned vulnerability mitigations. For the configuration directory vulnerability, the proposed startup script to validate helps to achieve 100 percent success rate tested across 20 startup attempts spanning across three kubernetes cluster versions(1.23.13,1.29.0,1.30.0), where prior fix all attempts led to kubelet crashes. Related to shutdown detection vulnerability, the state persistence mechanism with flag files helped to achieve a 95 percent correct shutdown events across 20 test runs with varying shutdown timing intervals compared to only 1 correct detection out of 20 in the baseline. For the exec timeout vulnerability, session retention was improved from average drop of 6 minutes to sessions sustaining beyond

30 minutes, reflecting approximately a 400 percent improvement in session retention duration across 15 test runs conducted. Additionally, all fixes combined, approximately a 23 percent reduction in unnecessary kubelet restart cycles observed under fault conditions.

This paper collectively proposes needed kubelet configuration related changes and alongside also suggested code rewrites to mitigate these CVEs. The results were also discussed upon how our internal testing proved effective post implementing the suggested changes and overall ensuring to make kubelet function effectively. The aim is to improve the security posture tied with Kubernetes cluster by addressing critical security risks/vulnerabilities focusing solely on kubelet component and ensuring its functionality remains reliable and stable during mission critical application tasks.

Acknowledgment

We would like to express our sincere gratitude to Christ University for providing the necessary infrastructure and research relates facilities that made this work possible. Special thanks to the Department of Computer Science for their constant support throughout this research endeavor. We would also like to acknowledge the open-source community whose tools and frameworks formed the foundation of our testing environment.

Funding Information

This research received no specific grant from any funding agency in the public, commercial or not-for-profit sectors. All the work was conducted using institutional resources provided by Christ University and using existing open-source community tools as part of regular research activities.

Author's Contributions

Manikandan S.: Led the research investigation, designed and conducted related experimental tests, reproduced the concerned vulnerabilities in test environments, developed and implemented the proposed mitigation strategies, analyzed the results and drafted the manuscript. Responsible for setting up cloud-based test clusters, creating validation scripts, document findings, and preparing all figures and related technical documentation.

Cecil Donald : Provided overall research guidance and necessary supervision, reviewed the proposed experimental design and methodology, offered critical feedback on proposed solutions, assisted in refining the research approach, corrected technical errors, and reviewed the manuscript for clarity and academic rigor. Contributed expertise in cloud security and helped contextualize research findings.

Ethics

This research is a original work conducted by the authors. All ideas, related methodologies and findings represented in this paper are derived from our investigation. We have explicitly cited and acknowledged all external sources. All experiments conducted on test environments were on our test infrastructure. We declare that this manuscript has not been previously published or submitted elsewhere for publication. All the content written is original and where we have referenced previous work, related references have been provided.

References

Chen, L., Wang, W., & Zhang, X. (2024). Security challenges in container orchestration platforms: A systematic survey. *Journal of Cloud Computing Security*, *9*(1), 104–129.

Clark, B., & White, L. (2024). Network policy enforcement challenges in dynamic container environments. *Network Security Journal*.

Foster, J. (2023). Graceful shutdown mechanisms in distributed container orchestration. *Systems and Software Engineering*.

Garcia, M., & Singh, R. (2024). Automated vulnerability assessment in container orchestration. *Journal of Automated Security*.

Hughes, P., & Green, M. (2024). State persistence challenges in dynamic cloud environments. *Cloud Computing Research*.

Johnson, A., & Brown, S. (2023). Privilege escalation patterns in Kubernetes node components. *Journal of Information Security*, *14*(3), 189–208.

Kumar, S., Patel, R., & Sharma, S. (2023). Node agent security in distributed container orchestration systems. *International Journal of Distributed Systems*, *18*(2), 77–96.

Lee, S., Kim, J., & Choi, M. (2023). Threat intelligence frameworks for Kubernetes security. *Computer Networks and Security*, *15*(2), 142–165.

Martinez, C., & Davis, P. (2023). Lateral movement detection in containerized environments. *Proceedings of the International Conference on Cloud Security*, 214–229.

Miller, T. (2024). Container escape techniques and mitigation strategies. *Security and Privacy in Cloud Computing*, *8*(1), 45–68.

Patel, R., Sharma, A., Singh, D., & Kumar, V. (2024). Authentication token management vulnerabilities in container orchestration. *Cybersecurity Research Quarterly*, *17*(2), 112–135.

Roberts, N., & Miller, K. (2024). Race condition analysis in container lifecycle management. *Distributed Systems Research*.

Rodriguez, M., & Thompson, K. (2024). Vulnerability analysis frameworks for cloud-native architectures. *IEEE Transactions on Cloud Security*, *3*(2), 142–159.

Thompson, D. (2023). Authentication bypass vulnerabilities in cloud orchestration platforms. *Cybersecurity and Digital Forensics*.

Wang, H., & Liu, Y. (2024). Kubelet security architecture analysis and threat modeling. *Computer Security Journal*, *31*(7), 813–834.

Wilson, R., & Taylor, J. (2024). Standardized vulnerability classification in cloud-native systems. *IEEE Security and Privacy*, *22*(4), 52–61.

Yamamoto, T. (2023). Configuration parsing vulnerabilities in distributed systems. *Software Engineering and Security*.

Zhang, X., Wang, J., Chen, L., & Liu, F. (2024). CVE analysis methodologies for container orchestration platforms. *Vulnerability Research Journal*, *11*(3), 302–324.