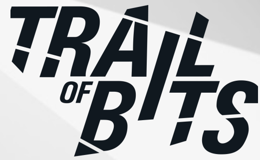

# Eclipse JKube

## Security Assessment

**September 14, 2023**

*Prepared for:*
**Marc Nuri San Felix**
The Eclipse Foundation

Organized by the Open Source Technology Improvement Fund, Inc.


*Prepared by:* **Artur Cygan, Kelly Kaoudis, and Emilio López**

# About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

**Trail of Bits, Inc.**
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

# Notices and Remarks

## Copyright and Distribution

© 2023 by Trail of Bits, Inc.

All rights reserved. Trail of Bits hereby asserts its right to be identified as the creator of this report in the United Kingdom.

This report is considered by Trail of Bits to be public information; it is licensed to the Eclipse Foundation under the terms of the project statement of work and has been made public at the Eclipse Foundation's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

The sole canonical source for Trail of Bits publications is the Trail of Bits Publications page. Reports accessed through any source other than that page may have been modified and should not be considered authentic.

## Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

# Table of Contents


**About Trail of Bits** **1**
**Notices and Remarks** **2**
**Table of Contents** **3**
**Executive Summary** **5**
**Project Summary** **7**
**Project Goals** **8**
**Project Targets** **9**
**Project Coverage** **10**
**Threat Model** **11**
Data Types 11
Data Flow 12
Components 13
Trust Zones 16
Trust Zone Connections 17
Threat Actors 20
Threat Scenarios 21
Recommendations 25
**Automated Testing** **28**
**Codebase Maturity Evaluation** **29**
**Summary of Findings** **31**
**Detailed Findings** **32**
1. Insecure defaults in generated artifacts 32
2. Risk of command line injection from secret 34
**A. Vulnerability Categories** **36**
**B. Code Maturity Categories** **38**
**C. Non-Security-Related Findings** **40**
**D. Docker Recommendations** **43**
**E. Hardening Containers Run via Kubernetes** **47**
Root Inside Container 47
Dropping Linux Capabilities 47
NoNewPrivs Flag 48
Seccomp Policies 48
Linux Security Module (AppArmor) 48

**F. Fix Review Results** **49**
Detailed Fix Review Results 50
**G. Fix Review Status Categories** **51**

# Executive Summary

## Engagement Overview

The Open Source Technology Improvement Fund engaged Trail of Bits to review the security of Eclipse JKube. JKube is a collection of plugins and libraries that are used for building, containerizing, and deploying Java applications to Kubernetes or OpenShift. It also provides a set of tools to improve the development experience of such cloud applications.

One consultant conducted a lightweight threat modeling exercise from March 20 to March 24, 2023. Two consultants performed a secure code review from May 1 to May 10, 2023, for a total of four engineer-weeks of effort. Our threat modeling exercises focused on examining the documentation, design, and specification of JKube to identify the system's trust boundaries, control flows, and any architecture-level weaknesses that could threaten the system. Our testing efforts focused on reviewing the JKube codebase and the artifacts that JKube produces when deploying an application. With full access to the source code and documentation, we performed static and dynamic testing of the JKube codebase and provided examples, using automated and manual processes.

## Observations and Impact

The testing portion of this assessment uncovered only two findings. The first one concerns the security of the default configuration for produced artifacts. The second is a weakness in data validation that could be used to execute unwanted code.

In general, we observed that JKube has a generally positive development and security posture, despite a few minor issues. The code maturity evaluation scores reflect this fact. However, there is significant room for improvement in the default configuration artifacts produced by JKube. Work in that area will help improve the security posture of the ecosystem in general and JKube users in particular.

## Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that the Eclipse JKube team take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.

- **Implement additional static analysis in the CI/CD process.** Tools such as CodeQL, Semgrep, and Checkov allow developers to spot issues early on in the development process. Integrate these tools not just on the JKube codebase itself, but also on the resulting artifacts produced by JKube.

- **Review the recommendations from the threat model and appendices.** The threat model recommendations and appendices D and E contain additional recommendations to improve the security posture of JKube and applications deployed with JKube.

The following tables provide the number of findings by severity and category:

### EXPOSURE ANALYSIS

| *Severity* | *Count* |
| --- | --- |
| High | 0 |
| Medium | 0 |
| Low | 1 |
| Informational | 1 |
| Undetermined | 0 |

### CATEGORY BREAKDOWN

| *Category* | *Count* |
| --- | --- |
| Configuration | 1 |
| Data Validation | 1 |

# Project Summary

## Contact Information

The following managers were associated with this project:

**Dan Guido**, Account Manager
dan@trailofbits.com

**Jeff Braswell**, Project Manager
jeff.braswell@trailofbits.com

The following engineers were associated with this project:

**Artur Cygan**, Consultant
artur.cygan@trailofbits.com

**Kelly Kaoudis**, Consultant
kelly.kaoudis@trailofbits.com

**Emilio López**, Consultant
emilio.lopez@trailofbits.com

## Project Timeline

The significant events and milestones of the project are listed below.

| Date | Event |
|---|---|
| **March 20, 2023** | Lightweight threat model kickoff call |
| **March 23, 2023** | Discovery call |
| **March 28, 2023** | Lightweight threat model readout meeting |
| **May 2, 2023** | Code review kickoff call |
| **May 15, 2023** | Delivery of report draft; report readout meeting |
| **August 1, 2023** | Delivery of fix review appendix |
| **September 14, 2023** | Delivery of comprehensive report |

# Project Goals

The engagement was scoped to provide a security assessment of the Eclipse Foundation's JKube project. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are the artifacts produced by the JKube software correct and secure?
- Are secrets managed securely?
- Does the software consume external input securely? Are all inputs and system parameters properly validated?
- Could the system experience a denial of service?
- Does the codebase conform to industry best practices?
- Are there any identifiable areas of improvement for the JKube CI/CD or SDLC?
- Is the existing test suite sufficient? Can additional testing be added that will improve the security of the project?

# Project Targets

The engagement involved a review and testing of the following target.

### Eclipse JKube

| | |
|---|---|
| Repository | https://github.com/eclipse/jkube |
| Version | `cfe5ee5eafd50c9b0ce2fd3a84b273a38ca680e1` (threat model) |
| | `c013e41cc7916719f2f4b54c58600a52141461b9` (code review) |
| Type | Java |
| Platform | JVM |

# Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Documentation review:** We reviewed the documentation prepared during the threat modeling exercise.
- **Static analysis:** We performed static analysis of the JKube codebase using CodeQL and Semgrep. We also used Checkov to analyze artifacts generated by JKube.
- **Manual review:** We reviewed the JKube source code, focusing on the code paths that generate artifacts and manage secrets.

## Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review:

- The third-party dependencies used by JKube
- The base Docker images and the CEKit tool used to build them
- The build and release pipelines

# Threat Model

As part of the OSTIF-organized Eclipse JKube audit, Trail of Bits conducted a lightweight threat model, drawing from Mozilla's "Rapid Risk Assessment" methodology and the National Institute of Standards and Technology's (NIST) guidance on data-centric threat modeling (NIST 800-154). We began our assessment of the design of JKube by reviewing the documentation of the current JKube release and the examples in the GitHub repository.

## Data Types

Depending on its configuration, JKube accepts input in the following formats:

- XML (POMs, XML fragments)
- JSON
- YAML (Helm charts, other Kubernetes or OpenShift configurations)
- Dockerfile
- Java, .properties files

Depending on its configuration, JKube produces output locally or remotely in the following formats:

- XML
- JSON
- YAML

Protocols over which JKube and/or its dependencies can communicate with the cluster include the following:

- HTTP
- HTTPS

## Data Flow

The following diagram presents common data flows that can occur during JKube usage. The box labeled "dockerd" refers to the use of either the literal Docker daemon or of dockerd via minikube for interacting with the cluster. JKube can run as a Maven or Gradle plugin, or it can interact with dockerd in a standalone fashion. JKube can also use Jib, podman, or the fabric8 Kubernetes client instead of dockerd and/or minikube to interact with the remote cluster. JKube can read provided application files and generate a Dockerfile or image configuration, or it can add to a provided XML configuration or Dockerfile. Profile fragments and other application files may be sourced locally or remotely. The JVM may resolve file references to remote locations; authentication/authorization are not currently required for such remote connections.

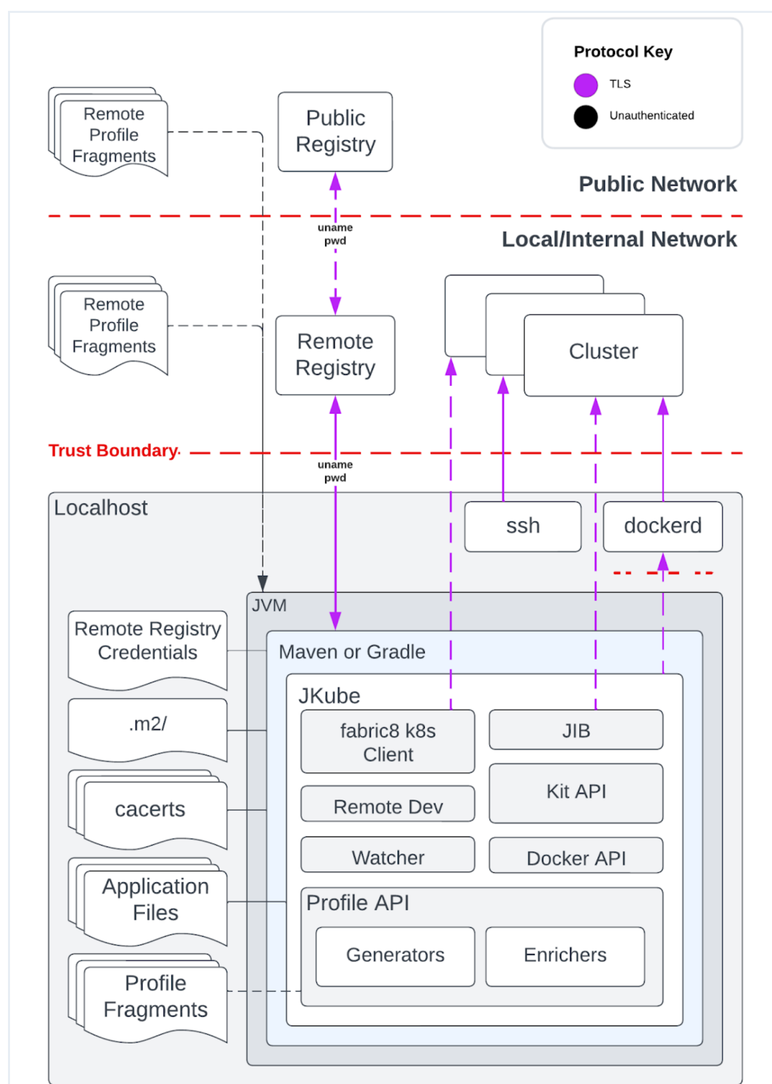


*Figure 1: Common JKube usage data flows*

## Components

The following table describes each JKube component and relevant dependency identified for our analysis. It also indicates whether the component or dependency is *not* in scope; an asterisk (*) next a component's name indicates that it was out of scope for this assessment. We explored the implications of threats involving out-of-scope components that directly affect in-scope components, but we did not consider threats to out-of-scope components themselves.

| Component | Description |
| --- | --- |
| JKube Kit | This represents the core logic of JKube. |
| Enricher API | An enricher API provides a Java interface for integrating tools and services such as Prometheus, Service Discovery, and health checks into containers built or deployed via JKube-created configurations. Enrichers can be combined in a profile or profile fragment(s). |
| Generator API | A Generator API provides a Java interface for consuming application source code and full or fragmentary configuration files to create special-purpose images/containers, charts, or manifests. It can create production container images or containers that allow administrators and application developers to conduct remote debugging. Generators can be combined in a profile or profile fragment(s). |
| Watcher | The JKube Watcher allows for hot reloading of image configurations and resources. |
| Remote Dev | The JKube Remote Dev module allows developers to configure deployments for later remote SSH connections. |
| Kit API | The Kit API is the standalone public API for building and deploying container images or configurations without using Maven or Gradle. |
| Docker API | The Docker API is a bespoke API that allows users to integrate JKube with dockerd directly or through a tool such as minikube. |
| Fabric8 Kubernetes Client (*) | JKube can integrate with the fabric8 Kubernetes client to produce resource files and deployments and to interact with the remote cluster. This component is out of scope. |
| Jib (*) | Jib is a Google library that JKube integrates with to build optimized |

| | |
|---|---|
| | Docker and OCI images without a Docker daemon or Dockerfiles. Jib also integrates with Maven and Gradle. This component is out of scope. |
| Gradle (*) | Gradle is an open-source build tool for JVM languages. This component is out of scope. |
| JKube Kubernetes Gradle Plugin | This plugin surfaces core JKube functionality for building Docker images, creating Kubernetes resources, and deploying JKube through Gradle. |
| JKube OpenShift Gradle Plugin | This plugin surfaces core JKube functionality for building S2I images, creating OpenShift resources, and deployment through Gradle. |
| Maven (*) | Maven is an open-source build tool for JVM languages. This component is out of scope. |
| JKube Kubernetes Maven Plugin | This plugin surfaces core JKube functionality for building Docker images, creating Kubernetes resources, and deployment through Maven. |
| JKube OpenShift Maven Plugin | This plugin surfaces core JKube functionality for building S2I images, creating OpenShift resources, and deployment through Maven. |
| Supporting Infrastructure (*) | This represents additional out-of-scope dependencies and components, noted here for completeness purposes. |
| Cluster (*) | This is the remote cluster, orchestrated by Kubernetes or OpenShift. This component is out of scope. |
| Docker, dockerd (*) | Docker (and its daemon, dockerd) is a local container management system that can integrate with minikube (an optional Kubernetes component). This component is out of scope. |
| Remote Registry (*) | JKube can push container images using delegated credentials to registries such as Docker Hub and Quay. Maven or Gradle can source packages from tools such as JFrog and Maven Central.<br><br>This component is out of scope. |
| Application Files (*) | These files include developer-provided code, configurations, and |

| | |
|---|---|
| | Dockerfiles. JKube can identify certain types of configuration files and choose to build a particular type of image; alternatively, JKube can extend a provided Dockerfile. This component is out of scope. |
| Source Control (*) | Source control is the infrastructure providing version control, hosting the codebase, facilitating the submission of pull requests and issues, and allowing maintainers to release security advisories. This component is out of scope. |

## Trust Zones

Trust zones capture logical boundaries where controls should or could be enforced by the system and allow developers to implement controls and policies between components' zones.

| Zone | Description | Included Components |
|---|---|---|
| Public Network | The public network is the broader internet. | • Remote registry (Docker Hub, Quay)<br>• Remote profile fragments or resources |
| Local/Internal Network | The local/internal network is an internally administered zone. This zone could be hosted remotely in AWS, Azure, or a similar platform. | • Remote registry (JFrog, etc.)<br>• Cluster (OpenShift, Kubernetes)<br>• Remote profile fragments or resources |
| Localhost | The localhost is the machine on which JKube runs. | • SSH<br>• Minikube<br>• Docker, dockerd<br>• Podman<br>• Local test cluster |
| JVM | The JVM is the local runtime. | • Maven<br>• Gradle<br>• JKube<br>• JDK<br>• Other JKube dependencies |

## Trust Zone Connections

At a design level, trust zones are delineated by the security controls that enforce the differing levels of trust within each zone. Therefore, it is necessary to ensure that data cannot move between trust zones without first satisfying the intended trust requirements of its destination. We enumerate such connections between trust zones below.

| Origin Zone | Destination Zone | Description | Connection Types | Authentication Types |
|---|---|---|---|---|
| JVM | Local Network | A JKube user connects to the remote cluster via the fabric8 Kubernetes client. | • HTTP | • TLS<br>• Cluster service account token |
| JVM | Local Network | A JKube user connects to the remote cluster via Jib. | • HTTP | • TLS<br>• Cluster service account token |
| JVM | Localhost | A JKube user connects to dockerd, optionally via minikube. | • HTTP | • TLS<br>• System user access controls |
| JVM | Local Network | A JKube user connects to a remote dockerd, optionally via minikube. | • HTTP | • TLS<br>• Cluster service account token |
| Localhost | Local Network | dockerd, optionally via minikube, connects to the remote cluster on behalf of the JKube user. | • HTTP | • TLS<br>• System user access controls<br>• Cluster service account token |

| | | | | |
|---|---|---|---|---|
| Localhost | Local Network | The local user connects to the remote cluster for debugging or remote development. | • SSH<br>• SCP<br>• HTTP | • Asymmetric cryptography<br>• TLS<br>• System user access controls<br>• Cluster user account |
| JVM | Localhost | A JKube user starts a container in a test cluster locally. | • UNIX sockets<br>• HTTP | • TLS<br>• Username and password<br>• System user access controls |
| Localhost | JVM | A local user makes changes to the JVM's configuration or environment, or sends signals to a running JVM process. | • Filesystem<br>• UNIX sockets<br>• IPC signals<br>• Java reflection | • System user access controls |
| Local Network | JVM | A JKube user provides profile fragments or resource URLs that resolve to external sources. | • java.net.URL<br>• HTTP | • TLS |
| JVM | Public Network | A JKube user sources a base image from a public remote registry.<br><br>Alternatively, a JKube user pushes updates to an image stored in a public remote registry. | • HTTP | • TLS<br>• Username and password |

| | | | | |
|---|---|---|---|---|
| JVM | Local Network | A JKube user sources a base image from an internally administered remote registry.<br><br>Alternatively, a JKube user pushes an image to an internal remote registry. | • HTTP | • TLS<br>• Username and password |
| Local Network | Public Network | Internally administered services, such as a JFrog Artifactory deployment on the internal network, can proxy access to remote resources located in public repositories (e.g., Maven Central, NPM, Docker Hub). | • HTTP | • TLS<br>• Username and password |

## Threat Actors

When conducting a threat model, we define the types of actors that could threaten the security of the system. We also define other "users" of the system who may be impacted by, or induced to undertake, an attack. For example, in a confused deputy attack such as cross-site request forgery, a normal user who is induced by a third party to take a malicious action against the system would be both the victim and the direct attacker. Establishing the types of actors that could threaten the system is useful in determining which protections, if any, are necessary to mitigate or remediate vulnerabilities. We will refer to these actors in descriptions of the security findings that we uncovered through the threat modeling exercise.

| Actor | Description |
| --- | --- |
| External Cluster User | External cluster users can access deployed components or endpoints available on the public internet. |
| Internal Cluster User | Internal cluster users can access deployed application resources, endpoints, or components that are *not* available on the public internet. |
| Namespace User | A namespace user is a developer and deployer of Kubernetes- or OpenShift-orchestrated applications with access to some, but perhaps not all, namespaces in the data plane. |
| Infrastructure Administrator | An infrastructure administrator can perform tasks on Kubernetes and OpenShift control and data plane components either locally or via SSH through a bastion host. They can also control related cloud resources. |
| Application Developer | An application developer is a contributor to the application logic, configuration, and other resources deployed in a container created with or managed via a configuration created with JKube. |
| Local User | Local users control a process or user account on the same host as the running JKube instance and can affect the local system environment, including the filesystem. |
| JKube Developer | A JKube developer is a contributor who has merged at least one commit to the main repository branch. |

## Threat Scenarios

The following table describes possible threat scenarios given the design, architecture, and risk profile of JKube.

<table>
<tr><th>Threat</th><th>Scenario</th><th>Actors</th><th>Components</th></tr>
<tr><td>Unsafe or missing default security options</td><td>JKube lacks commonly supportable security defaults and supports unsafe default configuration settings. An attacker could gain unauthorized access to many users' applications that are configured or deployed with JKube by exploiting standard insecure settings.</td><td><ul><li>External user</li><li>Internal user</li><li>Namespace user</li></ul></td><td><ul><li>Cluster</li><li>JKube Kit<ul><li>Remote Dev</li><li>Enricher</li><li>Generator</li></ul></li></ul></td></tr>
<tr><td>Cluster client denial of service</td><td>JKube's unsuitable general defaults and failure to warn users about the risks of selecting conflicting settings for remote cluster access or resource consumption could result in throttling or denial of service for remote cluster clients.</td><td><ul><li>External user</li><li>Internal user</li><li>Application developer</li><li>Local user</li></ul></td><td><ul><li>JKube Kit<ul><li>Generator</li><li>Enricher</li></ul></li></ul></td></tr>
<tr><td>Insecure remote file sourcing</td><td>The use of java.net.URL (e.g., in <code>jkube/kit/common/util</code> classes and in enrichers such as <code>DependencyEnricher</code>) to represent likely local files, and allowing remote file sourcing in JKube with neither remote location allowlisting nor requiring connection encryption are insecure practices. For example, they could allow an attacker to include a malicious configuration file in other JKube users' deployments or to replace another user's known required file with a malicious remote redirect or file.</td><td><ul><li>Application developer</li><li>External user</li><li>Local user</li></ul></td><td><ul><li>Cluster</li><li>JKube Kit<ul><li>Generator</li><li>Enricher</li></ul></li></ul></td></tr>
</table>

| | | | |
|---|---|---|---|
| Malicious remote configuration download, leading to local machine takeover | A user who unintentionally downloads a disguised malicious remote file to their local machine via JKube could allow a remote attacker to execute code on the user's local machine, possibly by exploiting JKube's `ExternalCommand` class, which allows the execution of an arbitrary child process in the runtime, or via `ClassLoader` manipulation. | • Application developer<br>• External user<br>• Local user | • Local system<br>• JKube Kit<br>  ○ Generator<br>  ○ Enricher |
| Insufficient hand-written user input sanitization | JKube's codebase has a pattern of insufficient user input sanitization, escaping, and other safeguards across multiple user input formats (XML, YAML, JSON, Java classes, etc.) in parsing utility classes and elsewhere. As a result, users who attempt to download and parse malicious or poorly written configuration or application files via JKube could allow an attacker to consume excessive local system resources (e.g., via an XML bomb/exponential entity expansion attack).<br><br>Alternatively, an attacker could craft malicious input to bypass case-by-case safeguards or input escape routines in order to attack a user's remote cluster. | • Application developer<br>• External user<br>• Local user | • Local system<br>• JKube Kit<br>  ○ Generator<br>  ○ Enricher |
| Unsafe deserialization | JKube's codebase has a general pattern of unsafe deserialization of `Serializable` objects in JKube (e.g., `DeepCopy`). As a result, an attacker could execute arbitrary code from JKube on a user's local machine. | • Application developer<br>• External user<br>• Local user | • Local system<br>• JKube Kit<br>  ○ Generator<br>  ○ Enricher |

| | | | |
|---|---|---|---|
| Attacker-controlled application files | JKube does not enforce access permission requirements for the JKube project workspace and included configuration files. As a result, a local attacker with sufficient system permissions could edit or overwrite a benign Dockerfile or Spring Boot configuration in a user's JKube project workspace or other configuration locations to add malicious content. Then, if configured, the appropriate Watcher would "hot" build the new content and deploy the content to the remote cluster. | • Local user<br>• Application developer<br>• External user<br>• Internal user<br>• Namespace user | • Local system<br>• JKube Kit<br>  ○ Watchers |
| Lack of opinionated secure defaults | JKube does not provide users the ability to set token expiry for JKube-created or JKube-replaced service account tokens. As a result, an attacker could obtain indefinite access to the namespace and application pod(s) that a service account can access in a user's cluster by either accidentally discovering or brute-forcing the token and then authenticating to the remote cluster API. | • Application developer<br>• External user<br>• Internal user<br>• Namespace user<br>• Local user | • JKube Kit<br>• Cluster |
| Hard-coded secrets and sensitive information allowed in user input | JKube has insufficient user input validation for JKube configuration and application files in the `SecretEnricher` and similar enrichers. As a result, users could take actions such as the following:<br>• Check keys or secrets in configuration files, resource fragments, and similar locations into source control<br>• Unintentionally deploy such data along with their application to the remote cluster | • Local user<br>• Application developer<br>• External user | • Cluster<br>• Source control<br>• JKube Kit<br>  ○ Generator<br>  ○ Enricher |

| | | | |
|---|---|---|---|
| | • Push an image with such data to a remote registry<br><br>Alternatively, an attacker with sufficient local system access permissions could locally obtain this sensitive hard-coded data. | | |
| Malicious commits, leading to backdoored or modified deployments | JKube pull requests do not require maintainer review before they are merged into JKube's `master` branch. As a result, a malicious developer could merge commits that pass static analysis but allow the developer to perform actions like the following:<br><br>• Eavesdrop on JKube user communications<br>• Add an open port into generated configurations<br>• Harvest cluster locations and credentials for later abuse | • JKube developer<br>• External user<br>• Internal user<br>• Namespace user | • Source control<br>• JKube Kit |
| Sensitive data in public GitHub issues | Users are not sufficiently warned against including sensitive data like keys, secrets, and `Authorization` header values when creating public issues and pull requests in the JKube GitHub repository, which makes it more likely that users will mistakenly post sensitive data publicly (e.g., this GitHub issue, which includes a `Basic` authentication token). Such publicly available data could allow a remote attacker scraping GitHub for sensitive data to gain unauthorized access to JKube users' clusters. | • JKube developer<br>• Application developer<br>• External user<br>• Internal user<br>• Namespace user | • Source control |

## Recommendations

- Implement and document a carefully selected set of security-related default configuration settings. Internally or externally audit these security defaults at least once a year to make sure that they still apply to the most common JKube configuration and deployment patterns and that they do not inadvertently make users' projects less secure.
    - The following are examples of good security defaults to consider:
        - Allow users to set token expiry for generated Kubernetes service accounts.
        - Prevent the use of `system:admin` user or group impersonation.
        - Do not allow JKube-facilitated cluster communications without TLS.
    - The following references from the OWASP "Cheat Sheet" series could be helpful:
        - "Input Validation"
        - "Web Service Security"
        - "XML Security"
        - "Secrets Management"
        - "Docker Top 10"
        - "Kubernetes"
    - The following are additional references to consider:
        - The NSA/CISA's 2021 "Kubernetes Hardening Guide"
            - Red Hat's blog post on the guide
        - Kubernetes' "Security Checklist"
        - OpenShift's "Security and Compliance"
        - CVEs and security advisories in JKube dependency projects (SnakeYAML, `fabric8-kubernetes-client`, Spring Boot, etc.) and integrator projects (advisories reported by Maven, CVE search results for Maven, advisories reported by Gradle, CVE search results for Gradle, Kubernetes, OpenShift)

        - "Kubernetes Failure Stories": A collection of talks and blog posts about Kubernetes failures
        - Red Hat: "12 Kubernetes Configuration Best Practices"
        - MITRE: "Weaknesses in Software Written in Java"

- Either prevent remote file reference and resolution within JKube, or implement Origin or domain allowlisting and file-content sanitization.
    - If remote file reference and resolution is not a desired JKube feature, prefer using non-network-capable file resolution methods rather than java.net.URL. This will significantly reduce JKube's attack surface.
    - However, if JKube intentionally supports remote file reference and resolution, require TLS for these connections, implement a user- or administrator-editable reference allowlist, and implement stringent download sanitization to better enable users to build and deploy secure applications.
- Prevent insecure deserialization.
    - Disallow unsafe reflection and arbitrary class loading or casting, which can lead to arbitrary local code execution.
    - In each class that implements `Serializable`, especially if user input (including previously saved JKube output) is deserialized, override `ObjectInputStream#resolveClass()`, and be generally cautious with uses of `ObjectInputStream#readObject()` to prevent unintentional deserialization of arbitrary classes.
    - Refer to the following resources for more information on unsafe reflection, deserialization, and class casting:
        - CWE-470: "Unsafe Reflection"
        - OWASP: "Deserialization Cheat Sheet"
        - PortSwigger: "Exploiting insecure deserialization vulnerabilities"
        - Ysoserial: A proof-of-concept tool for generating payloads that exploit unsafe Java object deserialization
        - CODE WHITE: Java Exploitation Restrictions in Modern JDK Times
        - Fortify Taxonomy: "Unsafe Reflection"

- One-off exceptional-case parsing sanitization regular expression uses for parsing and sanitizing user input (such as XML, JSON, YAML, and .properties files) can be brittle. If possible, replace such one-offs with more rigorous and reusable user input parsing and sanitation.

- For each received bug or security report, configure regression rules for JKube's Sonar static analysis pull request scans. That way, accidental or purposeful regressions will be easy to detect.

- Do not require (preferably, do not allow) developers or administrators using JKube to save plaintext passwords or sensitive data such as `Authorization` header values in stored configuration or .properties files.

    - Add support for the use of secret interpolation from services such as AWS Secrets Manager, HashiCorp Vault, and 1Password at runtime so that developers and administrators using JKube are not required to hard-code sensitive data in cleartext.

- Do not allow potentially sensitive data to be included in public GitHub issues or pull requests.

    - For example, issue #603 in the JKube repository includes the `Basic` authentication value of the `Authorization` header, which could be sensitive.

    - Use an issue template to gently remind contributors and question askers that GitHub issue and pull request content is public, and no sensitive or personally identifiable information such as usernames, tokens, and passwords should be submitted.

- Keep dependencies as updated as possible to ensure that upstream security fixes are applied.

    - If possible, use the most recent version of a single well-supported serialization/deserialization library such as Jackson for YAML, JSON, and XML, rather than allowing multiple versions of several libraries on the classpath that introduce duplicate functionality (e.g., Jackson, Google GSON, SnakeYAML, and JAXP). That way, only one library will need to be updated when new dependency releases come out. This is important because such libraries help to safeguard (but cannot completely protect) JKube and user-generated JKube output from the potential effects of malicious input.

    - Bump bouncycastle and other cryptographic dependencies to 1.8on if possible so that the most modern TLS cipher suites are supported.

# Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

We used the following tools in the automated testing phase of this project:

- Semgrep: An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time
- CodeQL: A code analysis engine developed by GitHub to automate security checks
- Checkov: An open-source static code analysis tool for detecting security misconfigurations in infrastructure as code (IaC)

# Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

| Category | Summary | Result |
|---|---|---|
| Arithmetic | The code does not perform many arithmetic operations; we did not find any issues concerning the operations it does perform. | Strong |
| Auditing | The project consistently uses a lightweight logging framework developed in-house (KitLogger, PrefixedLogger).<br><br>We did not identify any attempt at performing structured logging that could be useful for integrating with JKube. | Satisfactory |
| Complexity Management | The code is organized in reasonably sized modules and functions. We found occasional code duplication. JKube uses newer Java features such as lambdas that help to reduce the complexity. | Satisfactory |
| Configuration | We found that some defaults in generated artifacts can be insecure (TOB-JKUBE-1). | Moderate |
| Cryptography and Key Management | JKube performs very little cryptography and uses a few keys to authenticate to the third-party services. We did not identify any issues that could cause those keys to be exposed. | Satisfactory |
| Data Handling | The data is validated consistently; however, we found minor cases of insufficient validation, detailed in finding TOB-JKUBE-2 and in appendix C, which lists non-security-related findings. | Moderate |
| Documentation | Most of the functions are documented in code, but we found class documentation to be scarce. There is | Satisfactory |

| | | |
|---|---|---|
| | extensive documentation external to the code, which lives in the doc/ directory and is accessible on Eclipse's website. | |
| Maintenance | The project can be easily built and tested. The repository has a CI process set up and includes prepared contribution templates. | **Strong** |
| Memory Safety and Error Handling | The project uses memory-safe Java language and does not interface with native code through JNI. Errors appear to be handled correctly. | **Satisfactory** |
| Testing and Verification | The majority of the code is tested with unit tests, and most of the critical logic appears to be covered. | **Satisfactory** |

# Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

| ID | Title | Type | Severity |
|---|---|---|---|
| 1 | Insecure defaults in generated artifacts | Configuration | **Informational** |
| 2 | Risk of command line injection from secret | Data Validation | **Low** |

# Detailed Findings

| 1. Insecure defaults in generated artifacts | |
|---|---|
| Severity: **Informational** | Difficulty: **Undetermined** |
| Type: Configuration | Finding ID: TOB-JKUBE-1 |
| Target: Artifacts generated by JKube | |

### Description

JKube can generate Kubernetes deployment artifacts and deploy applications using those artifacts. By default, many of the security features offered by Kubernetes are not enabled in these artifacts. This can cause the deployed applications to have more permissions than their workload requires. If such an application were compromised, the permissions would enable the attacker to perform further attacks against the container or host.

Kubernetes provides several ways to further limit these permissions, some of which are documented in appendix E.

Similarly, the generated artifacts do not employ some best practices, such as referencing container images by hash, which could help prevent certain supply chain attacks.

We compiled several of the examples contained in the `quickstarts` folder and analyzed them. We observed instances of the following problems in the artifacts produced by JKube:

- Pods have no associated network policies.
- Dockerfiles have base image references that use the `latest` tag.
- Container image references use the `latest` tag, or no tag, instead of a named tag or a digest.
- Resource (CPU, memory) limits are not set.
- Containers do not have the `allowPrivilegeEscalation` setting set.
- Containers are not configured to use a read-only filesystem.
- Containers run as the root user and have privileged capabilities.
- Seccomp profiles are not enabled on containers.

- Service account tokens are mounted on pods where they may not be needed.

**Exploit Scenario**
An attacker compromises one application running on a Kubernetes cluster. The attacker takes advantage of the lax security configuration to move laterally and attack other system components.

**Recommendations**
Short term, improve the default generated configuration to enhance the security posture of applications deployed using JKube, while maintaining compatibility with most common scenarios. Apply automatic tools such as Checkov during development to review the configuration generated by JKube and identify areas for improvement.

Long term, implement mechanisms in JKube to allow users to configure more advanced security features in a convenient way.

**References**
- Appendix D: Docker Recommendations
- Appendix E: Hardening Containers Run via Kubernetes

## 2. Risk of command line injection from secret

| Severity: **Low** | Difficulty: **Medium** |
|---|---|
| Type: Data Validation | Finding ID: TOB-JKUBE-2 |
| Target: `jkube-kit/jkube-kit-spring-boot/src/main/java/org/eclipse/jkube/springboot/watcher/SpringBootWatcher.java` | |

### Description

As part of the Spring Boot watcher functionality, JKube executes a second Java process. The command line for this process interpolates an arbitrary secret, making it unsafe. This command line is then tokenized by separating on spaces. If the secret contains spaces, this process could allow an attacker to add arbitrary arguments and command-line flags and modify the behavior of this command execution.

```
StringBuilder buffer = new StringBuilder("java -cp ");
(...)
buffer.append(" -Dspring.devtools.remote.secret=");
buffer.append(remoteSecret);
buffer.append(" org.springframework.boot.devtools.RemoteSpringApplication ");
buffer.append(url);

try {
    String command = buffer.toString();
    log.debug("Running: " + command);
    final Process process = Runtime.getRuntime().exec(command);
```

*Figure 2.1: A secret is used without sanitization on a command string that is then executed.* (`jkube/jkube-kit/jkube-kit-spring-boot/src/main/java/org/eclipse/jkube/springboot/watcher/SpringBootWatcher.java#136–171`)

### Exploit Scenario

An attacker forks an open source project that uses JKube and Spring Boot, improves it in some useful way, and introduces a malicious `spring.devtools.remote.secret` secret in `application.properties`. A user then finds this forked project and sets it up locally. When the user runs `mvn k8s:watch`, JKube invokes a command that includes attacker-controlled content, compromising the user's machine.

### Recommendations

Short term, rewrite the command-line building code to use an array of arguments instead of a single command-line string. Java provides several variants of the `exec` method, such as `exec(String[])`, which are safer to use when user-provided input is involved.

Long term, integrate static analysis tools in the development process and CI/CD pipelines, such as Semgrep and CodeQL, to detect instances of similar problems early on. Review uses of user-controlled input to ensure they are sanitized if necessary and processed safely.

# A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

| Vulnerability Categories | |
|---|---|
| Category | Description |
| **Access Controls** | Insufficient authorization or assessment of rights |
| **Auditing and Logging** | Insufficient auditing of actions or logging of problems |
| **Authentication** | Improper identification of users |
| **Configuration** | Misconfigured servers, devices, or software components |
| **Cryptography** | A breach of system confidentiality or integrity |
| **Data Exposure** | Exposure of sensitive information |
| **Data Validation** | Improper reliance on the structure or values of data |
| **Denial of Service** | A system failure with an availability impact |
| **Error Reporting** | Insecure or insufficient reporting of error conditions |
| **Patching** | Use of an outdated software package or library |
| **Session Management** | Improper identification of authenticated users |
| **Testing** | Insufficient test methodology or test coverage |
| **Timing** | Race conditions or other order-of-operations flaws |
| **Undefined Behavior** | Undefined behavior triggered within the system |

| Severity Levels | |
|---|---|
| **Severity** | **Description** |
| **Informational** | The issue does not pose an immediate risk but is relevant to security best practices. |
| **Undetermined** | The extent of the risk was not determined during this engagement. |
| **Low** | The risk is small or is not one the client has indicated is important. |
| **Medium** | User information is at risk; exploitation could pose reputational, legal, or moderate financial risks. |
| **High** | The flaw could affect numerous users and have serious reputational, legal, or financial implications. |

| Difficulty Levels | |
|---|---|
| **Difficulty** | **Description** |
| **Undetermined** | The difficulty of exploitation was not determined during this engagement. |
| **Low** | The flaw is well known; public tools for its exploitation exist or can be scripted. |
| **Medium** | An attacker must write an exploit or will need in-depth knowledge of the system. |
| **High** | An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue. |

## B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

| Code Maturity Categories | |
|---|---|
| Category | Description |
| **Arithmetic** | The proper use of mathematical operations and semantics |
| **Auditing** | The use of event auditing and logging to support monitoring |
| **Authentication / Access Controls** | The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system |
| **Complexity Management** | The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions |
| **Configuration** | The configuration of system components in accordance with best practices |
| **Cryptography and Key Management** | The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution |
| **Data Handling** | The safe handling of user inputs and data processed by the system |
| **Documentation** | The presence of comprehensive and readable codebase documentation |
| **Maintenance** | The timely maintenance of system components to mitigate risk |
| **Memory Safety and Error Handling** | The presence of memory safety and robust error-handling mechanisms |
| **Testing and Verification** | The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage |

| Rating Criteria | |
|---|---|
| **Rating** | **Description** |
| **Strong** | No issues were found, and the system exceeds industry standards. |
| **Satisfactory** | Minor issues were found, but the system is compliant with best practices. |
| **Moderate** | Some issues that may affect system safety were found. |
| **Weak** | Many issues that affect system safety were found. |
| **Missing** | A required component is missing, significantly affecting system safety. |
| **Not Applicable** | The category is not applicable to this review. |
| **Not Considered** | The category was not considered in this review. |
| **Further Investigation Required** | Further investigation is required to reach a meaningful conclusion. |

## C. Non-Security-Related Findings

The following recommendations are not associated with specific vulnerabilities. However, implementing them may enhance code readability and may prevent the introduction of vulnerabilities in the future.

- The following `if` condition is always true. `i` is always less than `objects.length`; otherwise, the `for` loop would not be executing. The developer likely intended to use `++i` instead of `i++`.

```
for (int i = 0; i < objects.length; ) {
    sb.append(objects[i]);
    if (i++ < objects.length) {
        sb.append(joinWith);
    }
}
```

*Figure C.1: This `if` condition is always true.*
*(`jkube/kube-kit/build/api/src/test/java/org/eclipse/jkube/kit/build/api/helper/PathTestUtil.java#69–74`)*

- The `AssemblyManager` singleton may not work as expected on multi-threaded environments. Consider making the initialization synchronized.

```
public static AssemblyManager getInstance() {
    if (dockerAssemblyManager == null) {
        dockerAssemblyManager = new AssemblyManager();
    }
    return dockerAssemblyManager;
}
```

*Figure C.2: This initialization is not thread-safe.*
*(`jkube/jkube-kit/build/api/src/main/java/org/eclipse/jkube/kit/build/api/assembly/AssemblyManager.java#81–86`)*

- The following format string call has more arguments than parameters.

```
throw new DockerAccessException(e, "Unable to add tag [%s] to image [%s]",
targetImage,
        sourceImage, e);
```

*Figure C.3: This format string has an extra argument.*
*(`jkube/jkube-kit/build/service/docker/src/main/java/org/eclipse/jkube/kit/build/service/docker/access/hc/DockerAccessWithHcClient.java#476–477`)*

- The following issue appears to be fixed upstream. Consider removing the workaround or adjusting the comment if it is still desirable to keep Prometheus disabled.

```
// Switch off Prometheus agent until logging issue with WildFly Swarm is resolved
// See:
// - https://github.com/fabric8io/fabric8-maven-plugin/issues/1173
// - https://issues.jboss.org/browse/THORN-1859
ret.put("AB_PROMETHEUS_OFF", "true");
ret.put("AB_OFF", "true");
```

*Figure C.4: The code references an upstream issue that has been resolved.* (`jkube/jkube-kit/jkube-kit-thorntail-v2/src/main/java/org/eclipse/jkube/thorntail/v2/generator/ThorntailV2Generator.java#41–46`)

- The `parsedCredentials` array is indexed without first being checked to ensure that it has enough elements. This may cause the program to fail. This code is repeated in `jkube/jkube-kit/build/api/src/main/java/org/eclipse/jkube/kit/build/api/auth/RegistryAuth.java`.

```
public static AuthConfig fromCredentialsEncoded(String credentialsEncoded, String
email) {
  final String credentials = new String(Base64.decodeBase64(credentialsEncoded));
  final String[] parsedCredentials = credentials.split(":",2);
  return AuthConfig.builder()
      .username(parsedCredentials[0])
      .password(parsedCredentials[1])
      .email(email)
      .build();
}
```

*Figure C.5: `parsedCredentials` may have a single element in the array.* (`jkube/jkube-kit/build/api/src/main/java/org/eclipse/jkube/kit/build/api/auth/AuthConfig.java#89–97`)

- The `spring.devtools.remote.secret` secret is logged as part of the printed command. This might not represent a security issue, as this particular secret is also stored in plaintext, but as a general practice, privileged information should not be logged.

```
log.debug("Running: " + command);
```

*Figure C.6: The command string contains the mentioned secret.* (`jkube/jkube-kit/jkube-kit-spring-boot/src/main/java/org/eclipse/jkube/springboot/watcher/SpringBootWatcher.java#170`)

- There are several occurrences across the codebase of `parseInt` calls on user input without adequate error handling. An invalid input on a user-provided property may cause JKube to throw an exception.

# D. Docker Recommendations

This appendix provides general recommendations regarding the use of Docker. We recommend using the steps listed under the "Basic Security" and "Limiting Container Privileges" sections and avoiding the options listed under the "Options to Avoid" section. This appendix also describes the Linux features that form the basis of Docker container security measures and includes a list of additional references.

## Basic Security

- Do not add users to the `docker` group. Inclusion in the `docker` group allows a user to escalate his or her privileges to root without authentication.

- Do not run containers as a root user. If user namespaces are not used, the root user within the container will be the real root user on the host. Instead, create another user within the Docker image and set the container user by using the USER instruction in the image's Dockerfile specification. Alternatively, pass in the `--user $UID:$GID` flag to the `docker run` command to set the user and user group.

- Do not use the `--privileged` flag. Using this flag allows the process within the container to access all host resources, hijacking the machine.

- Do not mount the Docker daemon socket (usually `/var/run/docker.sock`) into the container. A user with access to the Docker daemon socket will be able to spawn a privileged container to "escape" the container and access host resources.

- Carefully weigh the risks inherent in mounting volumes from special filesystems such as `/proc` or `/sys` into a container. If a container has write access to the mounted paths, a user may be able to gain information about the host machine or escalate his or her own privileges.

## Limiting Container Privileges

- Pass the `--cap-drop=all` flag to the `docker run` command to drop all Linux capabilities and enable only those capabilities that are necessary to the process within a container using the `--cap-add=...` flag. Note, though, that adding capabilities could allow the process to escalate its privileges and "escape" the container.

- Pass the `--security-opt=no-new-privileges:true` flag to the `docker run` command to prevent processes from gaining additional privileges.

- Limit the resources provided to a container process to prevent denial-of-service scenarios.

- Do not use root (`uid=0` or `gid=0`) in a container if it is not needed. Use `USER ...` in the Dockerfile (or use `docker run --user $UID:$GID ...`).

The following recommendations are optional:

- Use user namespaces to limit the user and group IDs available in the container to only those that are mapped from the host to the container.
- Adjust the Seccomp and AppArmor profiles to further limit container privileges.
- Consider using SELinux instead of AppArmor to gain additional control over the operations a given container can execute.

## Options to Avoid

| **Flag** | **Description** |
|---|---|
| `--privileged` | Gives all kernel capabilities to the container and lifts all the limitations enforced by the `device` cgroup controller (i.e., allowing the container to do almost everything that the host can do) |
| `--cap-add=all` | Adds all Linux capabilities |
| `--security-opt apparmor=unconfined` | Disables AppArmor |
| `--security-opt seccomp=unconfined` | Disables Seccomp |
| `--device-cgroup-rule='a *:* rwm'` | Enables access to all devices (according to this documentation) |
| `--pid=host` | Uses the host PID namespace |
| `--uts=host` | Uses the host UTS namespace |
| `--network=host` | Uses the host network namespace, which grants access to all network interfaces available on a host |

## Linux Features Foundational to Docker Container Security

| Feature | Description |
|---|---|
| Namespaces | This feature is used to isolate or limit the view (and therefore the use) of a global system resource. There are various namespaces, such as PID, network, mount, UTS, IPC, user, and cgroup, each of which wraps a different resource. For example, if a process creates a new PID namespace, the process will act as if its PID is 1 and will not be able to send signals to processes created in its parent namespace.<br><br>The namespaces to which a process belongs are listed in the /proc/$PID/ns/ directory (each with its own ID) and can also be accessed by using the lsns tool. |
| Control groups (cgroups) | This is a mechanism for grouping processes/tasks into hierarchical groups and metering or limiting resources within those groups, such as memory, CPUs, I/Os, or networks.<br><br>The cgroups to which a process belongs can be read from the /proc/$PID/cgroup file. A cgroup's entire hierarchy will be indicated in a /sys/fs/cgroup/<cgroup controller or hierarchy>/ directory if the cgroup controllers are mounted in that directory. (Use the mount \| grep cgroup command to see whether they are.)<br><br>There are two versions of cgroups, cgroups v1 and cgroups v2, which can be (and often are) used at the same time. |
| Linux capabilities | This feature splits root privileges into "capabilities." Although this setting is primarily related to the actions a privileged user can take, there are different process capability sets, some of which are used to calculate the user's effective capabilities (such as after running an SUID binary). Therefore, dropping all Linux capabilities from all capability sets will help prevent a process from gaining additional privileges (such as through SUID binaries).<br><br>The Linux process capability sets for a given process can be read from the /proc/$PID/status file, specifically its CapInh, CapPrm, CapEff, CapBnd, and CapAmb values (which correspond to the inherited, permitted, effective, bound, and ambient capability sets, respectively). Those values can be decoded into meaningful capability names by using the capsh --decode=$VALUE tool.<br><br>While the effective capability set is the one that is directly used by the kernel to execute permission checks, it is best practice to limit all other sets |

| | |
|---|---|
| | too, since they may allow for gaining more effective capabilities, such as through SUID binaries or programs that have "file capabilities" set. |
| The "no new privileges" flag | Enabling this flag for a process will prevent the user who launched the process from gaining additional privileges (such as through SUID binaries). |
| Seccomp BPF syscall filtering | Seccomp BPF enables the filtering of arguments passed in to a program and the syscalls executed by it. It does this by writing a "BPF program" that is later run in the kernel.<br><br>Refer to the Docker default Seccomp policy. One can write a similar profile and apply it with the `--security-opt seccomp=<file>` flag. |
| AppArmor Linux Security Module (LSM) | AppArmor is LSM that limits a container's access to certain resources by enforcing a mandatory access control. AppArmor profiles are loaded into a kernel. A profile can be in either "complain" or "enforce" mode. In "complain" mode, violation attempts are logged only into the syslog; in "enforce" mode, such attempts are blocked.<br><br>To see which profiles are loaded into a kernel, use the aa-status tool. To see whether a given process will work under the rules of an AppArmor profile, read the `/proc/$PID/attr/current` file. If AppArmor is not enabled for the process, the file will contain an `unconfined` value. If it is enabled, the file will return the name of the policy and its mode (e.g., `docker-default (enforce)`).<br><br>Refer to the Docker AppArmor profile template and the generated form of the profile. |

## Additional References

- Understanding Docker Container Escapes: A Trail of Bits blog post that breaks down a container escape technique and explains the constraints required to use that technique
- Namespaces in Operation, Part 1: Namespaces Overview: A seven-part LWN article that provides an overview of Linux namespace features
- False Boundaries and Arbitrary Code Execution: An old but thorough post about Linux capabilities and the ways that they can be used in privilege escalation attempts
- Technologies for Container Isolation: A Comparison of AppArmor and SELinux: A comparison of AppArmor and SELinux

# E. Hardening Containers Run via Kubernetes

This appendix gives more context for the hardening of containers spawned by Kubernetes. Please note our definitions of the following terms:

- Container: This is the isolated "environment" created by Linux features such as namespaces, cgroups, Linux capabilities, and AppArmor and secure computing (seccomp) profiles. We are specifically concerned with Docker containers since the tested environment uses Docker as its container engine.
- Host: This is the unconfined environment on the machine running a container (e.g., a process run in global Linux namespaces).

## Root Inside Container

User namespaces allow for the remapping of user and group IDs between a host and a container; unless namespaces are used, the root user inside the container will be the root user in the host. In a default configuration of Docker containers, the container features limit the actions that the root user can take. However, if a process does not need to be run as root, it is best to run it with another user.

To run a container with another user, use the USER Dockerfile instructions. In Kubernetes, one can specify the user ID (UID) and various group IDs (GIDs) (e.g., a primary GID, a file system–related GID, and those for supplemental groups) using the `runAsUser`, `runAsGroup`, `fsGroup`, and `supplementalGroups` attributes of a `securityContext` field of a pod or other objects used to spawn containers.

## Dropping Linux Capabilities

Linux capabilities split the privileged actions that a root user's process can perform. Docker drops most Linux capabilities for security purposes but leaves others enabled for convenience. We recommend dropping all Linux capabilities and then enabling only those necessary for the application to function properly.

Linux capabilities can be dropped in Docker via the `--cap-drop=all` flag and in Kubernetes by specifying `capabilities`, `drop`, and `--all` in the `securityContext` key of the deployment's container configuration. Then, to restore necessary capabilities, use the `--cap-add=<cap>` flag in a `docker run` or specify them in `capabilities`, and use `add` in the `securityContext` field in the Kubernetes object manifest.

## NoNewPrivs Flag

The NoNewPrivs flag prevents additional privileges for a process or its children from being assigned. For example, it prevents a UID/GID from gaining capabilities or privileges by executing `setuid` binaries.

The `NoNewPrivs` flag can be enabled in a `docker run` via the `--security-opt=no-new-privileges` flag. In a Kubernetes deployment, specify `allowPrivilegeEscalation: false` in the `securityContext` field to enable it.

## Seccomp Policies

A seccomp policy limits the available system calls and their arguments. Normally, using seccomp requires a call to a `prctl` syscall with a special structure, but Docker simplifies the process and allows a seccomp policy to be specified as a JSON file. Using the default Docker profile is a good start for implementing a specific policy. Seccomp is disabled by default in Kubernetes.

The seccomp policy can be specified with a `--security-opt seccomp=<filepath>` flag in Docker. In Kubernetes, the seccomp policy can be set either by using a `seccompProfile` key in the `securityContext` field of a pod (in Kubernetes v1.19 or later) or by using the `container.seccomp.security.alpha.kubernetes.io/<container_name>:<profile_ref>` annotation (in pre-v1.19 versions). The Kubernetes documentation includes examples of both methods of setting a specific seccomp policy.

## Linux Security Module (AppArmor)

The LSM is a mechanism that allows kernel developers to hook various kernel calls. AppArmor is an LSM used by default in Docker. Another popular LSM is SELinux, but since it is more difficult to set up, it is not discussed here.

AppArmor limits what a process can do and which resources a process can interact with. Docker uses its default AppArmor profile, which is generated from this template. When Docker is used as a container engine in Kubernetes, the same profile is often used by default, depending on the Kubernetes cluster configuration. One can override the AppArmor profile in Kubernetes with the following annotation (which is further described here):

```
container.apparmor.security.beta.kubernetes.io/<container_name>:
<profile_ref>
```

# F. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On July 7, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the JKube team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the two issues described in this report, the JKube team has resolved one and has partially resolved the other. In addition to fixing the potential command line injection issue, the fixes include a new enricher that improves the generated configuration for Kubernetes objects, using more secure settings. JKube users must explicitly opt in to use this new enricher. For additional information, please see the Detailed Fix Review Results below.

| ID | Title | Severity | Status |
|---|---|---|---|
| 1 | Insecure defaults in generated artifacts | Informational | Partially resolved |
| 2 | Risk of command line injection from secret | Low | Resolved |

## Detailed Fix Review Results

**TOB-JKUBE-1: Insecure defaults in generated artifacts**

Partially resolved in PR #2177 and PR #2182. These pull requests introduce a new enricher that enforces several security best practices and recommendations for Kubernetes objects. However, this enricher is not enabled in the default configuration, which means that the generated deployment artifacts remain insecure by default unless the user enables this new feature.

**TOB-JKUBE-2: Risk of command line injection from secret**

Resolved in PR #2169. Among other changes, this pull request rewrote the command-line building code to use an array of arguments instead of a single command-line string. This way of invoking external programs does not present the same injection risk that was identified in the previous code with string interpolation.

# G. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

| **Fix Status** | |
|---|---|
| Status | Description |
| **Undetermined** | The status of the issue was not determined during this engagement. |
| **Unresolved** | The issue persists and has not been resolved. |
| **Partially Resolved** | The issue persists but has been partially resolved. |
| **Resolved** | The issue has been sufficiently resolved. |