Volcano.sh Security Audit

In collaboration with the CNCF, OSTIF and the Volcano.sh maintainers

Arthur Chen, Adam Korczynski, David Korczynski, Ada Logics

14th May 2025

About Ada Logics

Ada Logics is a software security company founded in Oxford, UK, 2018 and is now based in London. We are a team of dedicated, pragmatic security engineers and security researchers that work hands-on with code auditing, security automation and security tooling.

We are committed open source contributors and we routinely contribute to state of the art security tooling in the fuzzing domain such as advanced fuzzing tools like Fuzz Introspector and continuous fuzzing with OSS-Fuzz. For example, we have contributed to fuzzing of hundreds of open source projects by way of OSS-Fuzz. We regularly perform security audits of open source software and make our reports publicly available with findings and fixes, and we have audited many of the most widely used cloud native applications.

Ada Logics contributes to solving the challenge of securing the software supply-chain. To this end, we develop the tooling and infrastructure needed for ensuring a secure software development lifecycle, and we deploy these tools to critical software packages. On the tooling and infrastructure side, we contribute to projects such as the OpenSSF Scorecard project as well as the Sigstore projects like SLSA and Cosign.

Ada Logics helps some of the most exposed organisations secure their software, analyse their code and increase security automation and assurance, and if you would like to consider working with us please reach out to us via our website.

We write about our work on our blog. You can also follow Ada Logics on Linkedin, Twitter and Youtube.

Ada Logics ltd
71-75 Shelton Street,
WC2H 9JQ London,
United Kingdom

Contents

About Ada Logics 1

Audit contacts 3

Introduction 4

Risk scoring . . . 4

Scope . . . 5

Volcano.sh threat model 6

Volcano.sh trust boundaries . . . 6

Volcano.sh example attacks . . . 7

Volcano.sh threat actors . . . 8

Found issues 9

Attacker with foothold in elastic service or extender plugin can cause denial of service of scheduler . . . 10

Resources unnecessarily run as root . . . 11

Resources lack seccomp profile . . . 12

Resources are not configured with seLinux . . . 13

Resources may have unnecessary capabilities . . . 14

Resources allow privilege escalation . . . 15

Servers lack header timeouts . . . 16

Scheduler exposes profiling endpoints by default . . . 18

Missing warning when Volcano is configured to skip SSL certificate verification . . . 19

Source files are stored in repository as executables . . . 20

Audit contacts

Contact	Role	Organisation	Email
Adam Korczynski	Auditor	Ada Logics Ltd	adam@adalogics.com
David Korczynski	Auditor	Ada Logics Ltd	david@adalogics.com
Xavier Chang	Maintainer	Volcano.sh	cxz2536818783@gmail.com
William Wang	Maintainer	Volcano.sh	wang.platform@gmail.com
Amir Montazery	Facilitator	OSTIF	amir@ostif.org
Derek Zimmer	Facilitator	OSTIF	derek@ostif.org
Helen Woeste	Facilitator	OSTIF	helen@ostif.org

Introduction

In March and April 2025, Ada Logics carried out a security audit of Volcano.sh. The audit was a collaborative effort between Ada Logics, the Volcano.sh maintainers and Open Source Technology Improvement Fund and was funded by the CNCF. This report describes the work that Ada Logics (henceforth also reffered to as "we") carried out during the audit, the results of the work and the mitigations steps that Volcano.sh took.

The audit lasted five weeks during which we audited Volcano and had weekly meetings and ad hoc, unscheduled discussions with the Volcano team. The high level goal was to perform a holistic security audit of Volcano, and to do that the audit had the following goals:

1. Threat modelling. During the first week we focused primarily on threat modelling Volcano to map and enumerate threats, sensitive data, threat actors, data flow and more formalities for reasoning over Volcanos security measures later in the audit.
2. Manual auditing. After the first week our primary focus switched towards manually auditing Volcano. This involved a combination of code auditing, static analysis-guided auditing, dynamic analysis-guided auditing and in-cluster auditing.
3. Fuzzing. We integrated Volcano into OSS-Fuzz and wrote two fuzzers for the integration. The two fuzzers and Volcanos OSS-Fuzz build script were merged into the Volcano source tree.

We found ten issues during the audit which we reported to the Volcano team as they came up through private channels. We found the issues during different stages of the audit. One of the issues was assigned a CVE and released with a GitHub advisory. This vulnerability could allow an attacker with limited privileges in the cluster to deny the Volcano scheduler of service which is a central component of Volcano. Four of the issues were hardening recommendations to make Volcano secure by default which the Volcano team completed by diligently evaluating the privileges that the components and resources need in the cluster. In doing so, they could assign the lowest needed privileges to components and resources.

We shared the first version of this report with the Volcano team and OSTIF on 18th April 2025 and the second version on 14th May 2025.

We would like to thank the Volcano team, OSTIF and the CNCF for the collaboration.

Risk scoring

During the audit we used a simplified risk scoring system that considers risk exposure and risk impact. Exposure is the level at which an issue is exposed to an attacker. Impact is the level of privilege escalation an attacker can obtain by exploiting the security issue. We score both on a scale of 1-5 and add the two scores together for a final combined score. This score determines the severity of security issues. We assign this severity to the issues we find.

Risk Exposure

- 5: The security issue exists in core component(s) and is exposed in all use cases to untrusted input.
- 4: The security issue exists in widely used component(s) and is enabled by default. Users of the component(s) expose the issue by default to untrusted input.
- 3: The issue is exposed to authenticated and/or authorized users only.
- 2: The issue exists in component(s) that users need to enable to be affected.
- 1: The issue is only exposed to trusted users.

Risk Impact

- 5: An attack will have the highest possible impact.
- 4: An attack will have high impact with some constraints or limitations.

- 3: An attack can cause partial harm.
- 2: An attack can result in privilege escalation that will cause limited harm.
- 1: An attack can result in limited privilege escalation but requires further privilege escalation to cause harm.

We score each issue on both scales and then add the scores for a combined total score. The total score is the basis for the overall severity of found issues.

- 10: Critical
- 9 - 8: High
- 7 - 6: Moderate
- 5 - 4: Low
- 3 - 1: Informational

Scope

The audit included the code in the following code repositories:

1. https://github.com/volcano-sh/volcano

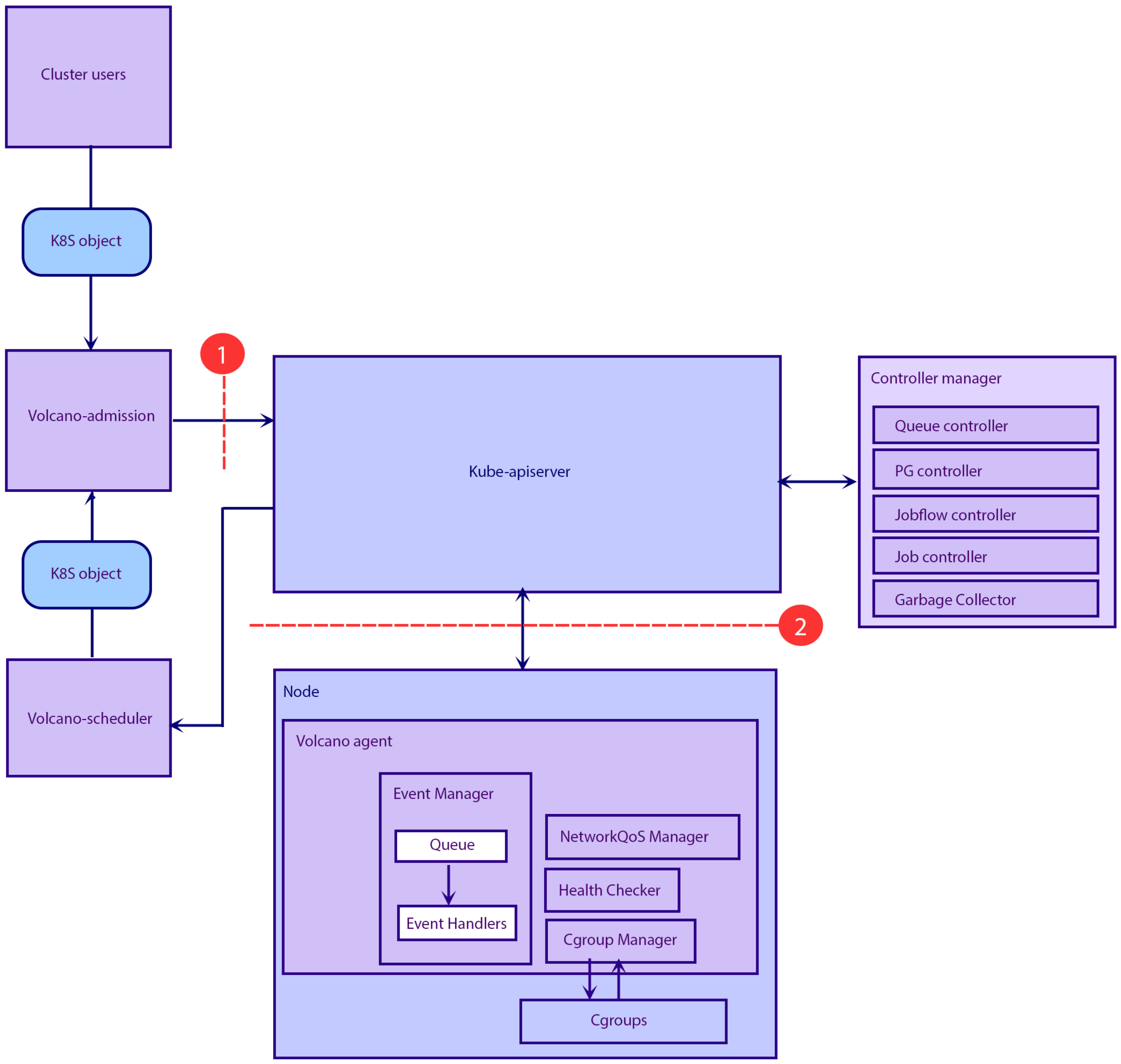
The audit was not fixed to a particular commit; we worked constantly against the latest master branches.

Volcano.sh threat model

Volcano.sh trust boundaries

Volcano.sh runs inside a Kubernetes cluster and inherits its security model. Kubernetes sets boundaries to data inside the cluster, and attackers against Volcano and its users in a runtime context are limited by Kubernetes' security controls. In this section we discuss Volcanos trust boundaries with the assumption that Volcano inherits Kubernetes' trust model. We enumerate the trust boundaries as they are intended to be, that is, the trust boundaries we draw are accepted in Volcano. Trust boundaries are conceptual lines that separate different parts of Volcano based on their level of trust. These boundaries indicate where control or responsibility changes between components, users, or systems and where Volcano must carefully validate data, enforce permissions, and secure communications.

We identify the following trust boundaries in Volcano.



1. The first trust boundary is between volcano-admission and the Kube APIServer. If objects are accepted by volcano-admission, their trust increase. Kubernetes admission controllers receive requests after the requests are authenticated

and authorized. At trust boundary 1, the request crosses the final boundary before it is accepted.

2. The second trust boundary is between nodes in the cluster and the Kube APIServer. This is largely inherited from the Kubernetes security model where nodes are isolated by a trust boundary. Nodes have lower trust than the Kube APIServer, and as such, trust flows from low to high in the direction of nodes to the Kube APIServer.

While these trust boundaries define a typical use case, there are numerous ways attackers can escalate the privileges without crossing these trust boundaries. For example, while requests that pass the admission controls of volcano-admission increase in trust, they may still have limited privileges. For example, a request that creates a pod may be accepted on that premise, but the request may be able to trigger a series of events inside the cluster that can lead to privilege escalations such as leaking data or creating other resources. The faults that could allow such privilege escalations may not have their root cause in volcano-admission but rather in other components that mishandle cluster resources. As such, trust boundaries are useful for understanding accepted trust flow, and in the next section, we consider example attacks that malicious cluster users may attempt to launch against the Volcano and its users.

Volcano.sh example attacks

In this section we consider example attacks that can threaten the confidentiality, integrity and availability of Volcano and its data. In these example attacks we draw on our previously reaning over the Volcano attack surface, threat actors and its security boundaries and pragmatically map these to production-level threats that could face Volcano. The list includes both runtime and supply-chain attacks.

Example attack 1: Authenticated user can control batch resources

An authenticated user may attempt to control the way Volcano batches compute at the expense of other users' jobs. In this way, other users' jobs would not be included in batches and this attack would constitute an insider threat actor, denial of service attack against other users. The attacker would need to be authenticated and authorized.

Example attack 2: Authenticated user can read other users' jobs

An authenticated user may attempt to read other authenticated users' jobs which can contain private or personal data. This can leak other users' data and compromise the integrity of their jobs and internal processes.

Example attack 3: Authenticated user can modify other users' jobs

An authenticated user can attempt to modify other users' jobs in the cluster. This could lead to lack of integrity as the attacker could change the data as part of the job. The attacker needs to be authenticated, and they would seek write-privileges to other users' jobs.

Example attack 4: Attacker can attempt to deny critical components from being operational

An attacker can attempt to make critical component fail such that no user in the cluster can use them. Denial of service of a critical component can result in user not being able to use the entire cluster for their needs.

Example attack 5: MitM pre-job

An attacker may attempt to change the data that Volcano batches as part of another user's job. As such, the victim intends for Volcano to process data "A", but an attacker intercepts the request between the user and the Volcano session and makes Volcano process data "B" instead. As such, the victim believes Volcano is processing one job but is in fact processing another which can impact their business decisions in an

Example attack 6: MitM post-job

An attacker may attempt to modify the result of another user's job after the job has completed. As such, the job may result with outcome "A" but the attacker presents the user with "B" instead. This can disrupt the victim's business decisions.

Volcano.sh threat actors

A threat actor is a persona with malicious intent that can affect the security of Volcano.sh. Threat actors can hold different levels of permissions and can be any persona that has the potential to elevate their privileges. In this section we enumerate Volcanos threat actors. Threat actors include actors that have a level of trust and will use that level of trust to further increase their privileges, however, they are only a *threat* actor if they behave with such intent. For example, the list below includes the Volcano.sh maintainers; Volcano.sh maintainers are by default trusted actors who do valuable work for the community. At the same time, maintainers' privileges are limited to maintaining the project and should not allow them to compromise users' deployments.

#	Name	Level of privilege	Description
1	Volcano.sh maintainers	Moderate	The Volcano.sh have high privileges in the context of Volcano.sh's software development lifecycle but have no privileges inside users' clusters. Maintainers are valuable members of the Volcano.sh community and are responsible for reviewing code contributions, triaging issues, cutting releases and supporting and engaging with the community. Attackers may seek to obtain the trust of the community over a long period and obtain maintainer status if it can allow them to exploit high valuable users of Volcano.sh. As such, Volcano.sh should not overprivilege maintainers.
2	Volcano.sh contributors	Low	Volcano.sh contributors contribute new code or code changes to the Volcano.sh code repository. They may attempt to introduce vulnerabilities in the code that they can exploit at a later stage to target users at runtime.
3	Dependency maintainers and contributors	None	Maintainers and contributors working on software projects that Volcano.sh consumes as 3rd-party dependencies may be able to introduce vulnerabilities in their software that could compromise Volcano.sh at runtime.
4	Cluster maintainers	High	Volcano.sh adopters will likely have a cluster maintainer or devops team. If an attacker is able to impersonate a cluster maintainer, their privileges may vary depending on the use case. In this audit, we consider the cluster maintainer to have sudo privileges in the cluster and as such does not represent a threat. In other words, if the cluster maintainer - and no other threat actor - can bring Volcano.sh in a vulnerable state, the root cause does not constitute a security vulnerability.
5	Internal cluster users	Low to High	These are users with permissions inside the cluster albeit limited privileges. Users will likely have granular permissions such as `CREATE` permissions for particular resources. Internal cluster users should not be able to elevate their privileges beyond their granularly assigned privileges or to any permissions that allow them to compromise the security of the cluster, Volcano.sh and its users.
6	External user	None	The external user does not have any access to the cluster or any permissions assigned to it. The external user is not be able to authenticate and authorize. As such, any way that this threat actor can elevate their privileges in the cluster constitutes a security breach.

Found issues

In this section we describe the issues we found in the audit.

ID	Name	Severity	Status
ADA-VLCN-2025-1	Attacker with foothold in elastic service or extender plugin can cause denial of service of scheduler	High	Fixed
ADA-VLCN-2025-2	Resources unnecessarily run as root	Moderate	Fixed
ADA-VLCN-2025-3	Resources lack seccomp profile	Moderate	Fixed
ADA-VLCN-2025-4	Resources are not configured with seLinux	Moderate	Fixed
ADA-VLCN-2025-5	Resources may have unnecessary capabilities	Moderate	Fixed
ADA-VLCN-2025-6	Resources allow privilege escalation	Moderate	Fixed
ADA-VLCN-2025-7	Servers lack header timeouts	Low	Fixed
ADA-VLCN-2025-8	Scheduler exposes profiling endpoints by default	Low	Fixed
ADA-VLCN-2025-9	Missing warning when Volcano is configured to skip SSL certificate verification	Low	Fixed
ADA-VLCN-2025-10	Source files are stored in repository as executables	Informational	Fixed

Attacker with foothold in elastic service or extender plugin can cause denial of service of scheduler

Severity	High
Status	Fixed
CWE	CWE-400: Uncontrolled Resource Consumption
id	ADA-VLCN-2025-1

This is a disclosure for a security issue in Volcano. The issue allows an attacker who has compromised either the Elastic service or the extender plugin (henceforth AKA "the two vulnerable services") to cause denial of service of the scheduler. This is a privilege escalation, because Volcano users may run their Elastic service and extender plugins in separate pods or nodes from the scheduler. In the Kubernetes security model, node isolation is a security boundary, and as such an attacker is able to cross that boundary in Volcanos case if they have compromised either the vulnerable services or the pod/node in which they are deployed. The two vulnerable services are:

1. https://github.com/volcano-sh/volcano/blob/b4f35c83ec5e028f10e457c6abe89684cb491e1b/pkg/scheduler/metrics/source/metrics_client_elasticsearch.go
2. https://github.com/volcano-sh/volcano/blob/2dff5d83e975578eb6ac53436bfa349599ef68cc/pkg/scheduler/plugins/extender/extender.go

The scheduler communicates with the two vulnerable plugins by sending an HTTP request to the plugin and reading the response. In the case of the elastic service, Volcano uses an elastic client which makes an HTTP request under the hood, and in the case of the extender plugin, Volcano sends its own HTTP request. The issue is that when the scheduler reads the response from the two vulnerable plugins, Volcano does so in an unbounded manner, meaning that it reads the entire response regardless of how large the response is. As such, an attacker with control over the two vulnerable services can return a large response to the scheduler to cause the scheduler to consume excessive memory and be denied of service as a result. This will make the scheduler unavailable to other users and workloads in the cluster. The scheduler will either crash with an unrecoverable OOM panic or freeze while consuming excessive amounts of memory.

The two vulnerable services read the unbounded response on the following lines:

1. https://github.com/volcano-sh/volcano/blob/b4f35c83ec5e028f10e457c6abe89684cb491e1b/pkg/scheduler/metrics/source/metrics_client_elasticsearch.go#L154
2. https://github.com/volcano-sh/volcano/blob/2dff5d83e975578eb6ac53436bfa349599ef68cc/pkg/scheduler/plugins/extender/extender.go#L325

`json.NewDecoder().Decode` can exhaust the host machines memory if the input buffer is large enough. After a while the machine freezes temporarily and the process then kills. The scheduler will behave in a this manner on the following lines:

1. https://github.com/volcano-sh/volcano/blob/b4f35c83ec5e028f10e457c6abe89684cb491e1b/pkg/scheduler/metrics/source/metrics_client_elasticsearch.go#L154
2. https://github.com/volcano-sh/volcano/blob/2dff5d83e975578eb6ac53436bfa349599ef68cc/pkg/scheduler/plugins/extender/extender.go#L325

Volcano has issued the following advisory for this issue: GHSA-hg79-fw4p-25p8

Resources unnecessarily run as root

Severity	Moderate
Status	Fixed
CWE	CWE-250: Execution with Unnecessary Privileges, CWE-276: Incorrect Default Permissions
id	ADA-VLCN-2025-2

`runAsNonRoot` ensures that resources run without root privileges. Volcano should approach this by setting it to true for all resources and by default and only remove as needed. We found 4 resources containers that were not using this setting which indicates that the developers have not made the resources run without root privileges by default:

1. volcano-scheduler
2. volcano-admission
3. volcano-controllers
4. volcano-dashboard

There are several risks associated with allowing a resources to run as root in the cluster. An attacker who compromises the root-privileged application has a wider attack surface to further escalate their privileges than they would if the resource did not have root privileges. In addition, some container breakout vulnerabilities will allow an attacker to obtain root privileges on the host if successfully exploited.

If an attacker compromises a Kubernetes container configured with runAsNonRoot=false, they gain root access inside the container, which dramatically increases the potential impact of the breach. With root privileges, the attacker can:

- Escalate privileges by exploiting kernel vulnerabilities or using setuid binaries to break out of the container and gain control over the host node.
- Access and modify sensitive files, both inside the container and on mounted host volumes, potentially tampering with configurations, credentials, or logs.
- Interact with the container runtime or Docker socket (if mounted), enabling them to control other containers, deploy malicious workloads, or escape into the cluster.
- Access service account tokens and secrets, allowing lateral movement to other pods or direct interaction with the Kubernetes API for further compromise.
- Bypass security controls like file permissions or audit logging, making detection and containment more difficult.

In short, a compromised root container can serve as a launchpad for full host takeover, data exfiltration, and broader cluster-level attacks, especially in poorly secured environments.

Fix PR(s):

1. https://github.com/volcano-sh/volcano/pull/4207
2. https://github.com/volcano-sh/dashboard/pull/86

Resources lack seccomp profile

Severity	Moderate
Status	Fixed
CWE	CWE-250: Execution with Unnecessary Privileges, CWE-276: Incorrect Default Permissions
id	ADA-VLCN-2025-3

Seccomp restricts the system calls (syscalls) a container can make to the host kernel. It's a layer of defense to limit what code inside a container can do, especially in case the container gets compromised.

Not setting a seccomp profile for Kubernetes containers exposes workloads to several potential security vulnerabilities, particularly around syscall abuse and kernel-level attack surfaces. As such, a missing seccomp profile is a security misconfiguration that increases the kernel attack surface and the risk from container escapes.

Kubernetes best practices for Pod hardening dictate that the Seccomp profile is either `RuntimeDefault` (recommended) or `Localhost`. All of the following containers are missing Seccomp profiles:

The following resources are missing seccompProfile:

1. Volcano-scheduler
2. Volcano-admission
3. Volcano-controllers
4. Volcano-dashboard

If an attacker compromises a Kubernetes container that is running without a seccomp profile, they can execute any available Linux system call, significantly increasing the risk of exploitation. This unrestricted syscall access allows the attacker to:

- Exploit kernel vulnerabilities using dangerous syscalls like unshare, ptrace, or clone, potentially leading to container escape and host compromise.
- Bypass security boundaries that would otherwise block risky operations (e.g., creating new namespaces or manipulating kernel memory).
- Run advanced malware or reverse shells that rely on specific syscalls often restricted by seccomp, making it easier to establish persistence or exfiltrate data.
- Avoid detection, since seccomp profiles also provide syscall filtering and logging, which help in identifying malicious behavior.

Overall, not using a seccomp profile removes a critical layer of sandboxing, making it far easier for a compromised container to escalate privileges, attack the host, or pivot deeper into the Kubernetes cluster.

Fix PR(s):

1. https://github.com/volcano-sh/volcano/pull/4207
2. https://github.com/volcano-sh/dashboard/pull/86

Resources are not configured with seLinux

Severity	Moderate
Status	Fixed
CWE	CWE-250: Execution with Unnecessary Privileges, CWE-276: Incorrect Default Permissions
id	ADA-VLCN-2025-4

Not configuring SELinux for Kubernetes containers significantly weakens the security posture of your cluster by eliminating a critical layer of mandatory access control (MAC). Without SELinux, containers rely solely on discretionary access control mechanisms like traditional Linux file permissions, which are insufficient in protecting against privilege escalation, container breakout, or unauthorized access to sensitive resources. SELinux enforces strict policies that limit what processes can do—even if they are running as root—helping to contain compromised containers and prevent them from affecting the host or other containers. Without it, any container that gains elevated privileges or misuses mounted host paths (such as /var/run/docker.sock or /etc) can potentially tamper with the host system or escalate to full control of the node. Moreover, the absence of SELinux removes fine-grained control over inter-process communication and file access, increasing the risk of lateral movement within the cluster. This lack of isolation also hinders compliance with security standards and benchmarks, making the environment more vulnerable to sophisticated attacks.

The following are missing seLinux in their security context:

1. volcano-scheduler
2. volcano-admission
3. volcano-controllers
4. volcano-dashboard

Fix PR(s):

1. https://github.com/volcano-sh/volcano/pull/4207
2. https://github.com/volcano-sh/dashboard/pull/86

Resources may have unnecessary capabilities

Severity	Moderate
Status	Fixed
CWE	CWE-250: Execution with Unnecessary Privileges, CWE-276: Incorrect Default Permissions
id	ADA-VLCN-2025-5

In Kubernetes, capabilities in the securityContext are a way to fine-tune the privileges granted to a container process, offering more control than simply running as root or non-root. They are part of Linux's capability-based security model, which breaks down the all-powerful root privileges into smaller, more manageable pieces.

Managing Linux capabilities in Kubernetes matters because it allows you to enforce the principle of least privilege at a granular level, significantly reducing the risk of containerized workloads performing unauthorized or dangerous actions. By default, containers can inherit a broad set of capabilities that may not be necessary for their intended function, exposing the system to potential abuse if the container is compromised. For example, a container with capabilities like `CAP_NET_ADMIN` or `CAP_SYS_ADMIN` could manipulate network settings or interact with kernel-level features, increasing the likelihood of privilege escalation or host compromise. Carefully controlling which capabilities are added or dropped limits what a process can do, even if it runs as root, creating a tighter security boundary and making it much harder for attackers to exploit misconfigurations or vulnerabilities within the container or the host system.

The best practice for managing capabilities in Kubernetes is to adopt a "deny by default" approach by explicitly dropping all capabilities and only adding back those that are strictly necessary for the container's functionality. This is typically done by setting drop: ["ALL"] in the container's securityContext, which removes all Linux capabilities—even those normally granted to the root user—ensuring that the container starts with the minimal privilege set. If the application requires specific capabilities to operate, such as NET_BIND_SERVICE to bind to a low-numbered port, these can be selectively re-added using the add field. This approach significantly reduces the container's attack surface by preventing unnecessary privileges that could be exploited if the container is compromised. Implementing minimal capabilities, along with other hardening techniques like runAsNonRoot, seccomp profiles, and SELinux or AppArmor, forms a strong, layered defense strategy for securing Kubernetes workloads.

1. volcano-scheduler
2. Volcano-admission
3. Volcano-controllers
4. volcano-dashboard

Fix PR(s):

1. https://github.com/volcano-sh/volcano/pull/4207
2. https://github.com/volcano-sh/dashboard/pull/86

Resources allow privilege escalation

Severity	Moderate
Status	Fixed
CWE	CWE-250: Execution with Unnecessary Privileges, CWE-276: Incorrect Default Permissions
id	ADA-VLCN-2025-6

The `allowPrivilegeEscalation` setting in Kubernetes controls whether a process running in a container can gain more privileges than its parent process, such as through the use of `setuid` or `setgid` binaries. If this option is set to true or left unset (as it defaults to true), it opens the door for attackers or compromised applications to escalate their privileges within the container. This can lead to scenarios where a non-root process spawns a child process with elevated capabilities, potentially bypassing security controls and gaining access to sensitive resources or performing restricted operations. Such privilege escalation can be particularly dangerous when combined with other misconfigurations, such as overly permissive Linux capabilities or mounted host paths, enabling attackers to break out of the container and affect the host or other workloads. Disabling privilege escalation (allowPrivilegeEscalation: false) is a crucial defense-in-depth measure that helps enforce the principle of least privilege, preventing unauthorized access and reducing the impact of any potential security breach.

1. volcano-scheduler
2. volcano-admission
3. volcano-controllers
4. volcano-dashboard

We recommend disallowing privilege escalation by default and instead let users enable it if they so require in their particular deployments. By disallowing privilege escalation by default, Volcano is hardened with best practices in its default installation.

Fix PR(s):

1. https://github.com/volcano-sh/volcano/pull/4207
2. https://github.com/volcano-sh/dashboard/pull/86

Servers lack header timeouts

Severity	Low
Status	Fixed
CWE	CWE-693: Protection Mechanism Failure
id	ADA-VLCN-2025-7

Five of Volcanos servers lack timeouts for HTTP headers. In two of these cases, the serve function has no support for setting timeouts, and in three of the cases, the servers are not configured with header timeouts. The same is the case for the Volcano health server.

Timeouts are a hardening mechanism that prevent a request from taking excessive time to process. A request that takes long for Volcano to process could hog the server and prevent it from processing requests from other users. As such, this prevents Denial of Service attacks in multi-tenant environments which Volcano is. However, servers can also be exposed to users with privileges that are limited to only send requests to a vulnerable server, and in those cases, such limited privileges can allow the attacker to deny the server from other users. An attacker in this scenario could be not just a user, but an intruder who has managed to manifest themselves in the system.

Configuring HTTP servers is a hardening recommendation; attackers still need to find a way to make Volcanos HTTP servers take a long time to process a request which there may not be at all times. Nonetheless, Volcano may add such behavior in the future, and we recommend preventing vulnerabilities for such cases by limiting processing time for a single request.

The following HTTP servers lack configured timeout:

1: Socket

https://github.com/volcano-sh/volcano/blob/690b4b03638f2ee956ae94dfedf1c57f556963ee/pkg/util/socket.go#L203-L205

```
server := http.Server{
    Handler: m,
}
```

Issue: Lacks `ReadHeaderTimeout` Documentation on `ReadHeaderTimeout`: https://pkg.go.dev/net/http#Server

2: webhook-manager app server:

https://github.com/volcano-sh/volcano/blob/690b4b03638f2ee956ae94dfedf1c57f556963ee/cmd/webhook-manager/app/server.go#L110-L113

```
server := &http.Server{
    Addr:      config.ListenAddress + ":" + strconv.Itoa(config.Port),
    TLSConfig: configTLS(config, restConfig),
}
```

3: Agent:

https://github.com/volcano-sh/volcano/blob/690b4b03638f2ee956ae94dfedf1c57f556963ee/cmd/agent/app/app.go#L83-L86

```
s := &http.Server{
    Addr:    net.JoinHostPort(address, strconv.Itoa(port)),
    Handler: mux,
}
```

Issue: Lacks `ReadHeaderTimeout` Documentation on `ReadHeaderTimeout`: https://pkg.go.dev/net/http#Server

4: controller-manager metrics

https://github.com/volcano-sh/volcano/blob/690b4b03638f2ee956ae94dfedf1c57f556963ee/cmd/controller-manager/app/server.go#L59-L64

```
if opt.EnableMetrics {
    go func() {
        http.Handle("/metrics", commonutil.PromHandler())
        klog.Fatalf("Prometheus Http Server failed %s", http.ListenAndServe(opt.
            ListenAddress, nil))
    }()
}
```

Issue: `net/http.Handle` does not support setting timeouts

5: Scheduler metrics:

https://github.com/volcano-sh/volcano/blob/690b4b03638f2ee956ae94dfedf1c57f556963ee/cmd/scheduler/app/server.go#L72-L77

```
if opt.EnableMetrics {
    go func() {
        http.Handle("/metrics", commonutil.PromHandler())
        klog.Fatalf("Prometheus Http Server failed %s", http.ListenAndServe(opt.
            ListenAddress, nil))
    }()
}
```

6: Health server

https://github.com/volcano-sh/apis/blob/2d1b261f52f1b0d3c6c345484ce25ff1332cfe97/pkg/apis/helpers/helpers.go#L202-L206

```
server := &http.Server{
    Addr:           listener.Addr().String(),
    Handler:        pathRecorderMux,
    MaxHeaderBytes: 1 << 20,
}
```

Issue: Lacks `ReadHeaderTimeout` Documentation on `ReadHeaderTimeout`: https://pkg.go.dev/net/http#Server

Fix PR(s):

1. https://github.com/volcano-sh/volcano/pull/4208

Scheduler exposes profiling endpoints by default

Severity	Low
Status	Fixed
CWE	CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
id	ADA-VLCN-2025-8

Volcano Scheduler exposes the profiling endpoints by default to any user with access to the localhost. Profiling data in sensitive and should be protected in a production setting, whereas allowing users to opt in to enable in for development is good practice. As of currently, an unauthenticated user with limited privileged but a foothold on localhost can view the schedulers profiling data.

The profiling server is instantiated with the following import:

https://github.com/volcano-sh/volcano/blob/690b4b03638f2ee956ae94dfedf1c57f556963ee/cmd/scheduler/main.go#L33

```
        _ "net/http/pprof"
```

Instead of enabling enabling as such, we recommend only allowing it if a given option is enabled when starting the scheduler.

Fix PR(s):

1. https://github.com/volcano-sh/volcano/pull/4173

Missing warning when Volcano is configured to skip SSL certificate verification

Severity	Low
Status	Fixed
CWE	CWE-778: Insufficient Logging
id	ADA-VLCN-2025-9

Volcano.sh allows users to skip SSL certificate verification when using the Prometheus and Elastic metrics clients. The option is disabled by default and users are required to enable it to see the effects which constitutes best practices. However, as a matter of hardening, we recommend that Volcano logs with warning level that the feature should not be used in production. As such, if users enable it, they should see a warning log entry.

The Prometheus metrics client lacks a warning here:

https://github.com/volcano-sh/volcano/blob/b9793c4c4cc2531ab4cd3945b178164637b49e73/pkg/scheduler/metrics/source/metrics_client_prometheus.go#L59-L68

```go
func (p *PrometheusMetricsClient) NodeMetricsAvg(ctx context.Context, nodeName string)
    (*NodeMetrics, error) {
    klog.V(4).Infof("Get node metrics from Prometheus: %s", p.address)
    var client api.Client
    var err error
    insecureSkipVerify := p.conf["tls.insecureSkipVerify"] == "true"
    tr := &http.Transport{
        TLSClientConfig: &tls.Config{
            InsecureSkipVerify: insecureSkipVerify,
        },
    }
```

The ElasticSearch client lacks a warning here:

https://github.com/volcano-sh/volcano/blob/b9793c4c4cc2531ab4cd3945b178164637b49e73/pkg/scheduler/metrics/source/metrics_client_elasticsearch.go#L67-L77

```go
    insecureSkipVerify := conf["tls.insecureSkipVerify"] == "true"
    e.es, err = elasticsearch.NewClient(elasticsearch.Config{
        Addresses: []string{address},
        Username:  conf["elasticsearch.username"],
        Password:  conf["elasticsearch.password"],
        Transport: &http.Transport{
            TLSClientConfig: &tls.Config{
                InsecureSkipVerify: insecureSkipVerify,
            },
        },
    })
```

Fix PR(s):

1. https://github.com/volcano-sh/volcano/pull/4211

Source files are stored in repository as executables

Severity	Informational
Status	Fixed
CWE	CWE-276: Incorrect Default Permissions
id	ADA-VLCN-2025-10

Volcano.sh stores some of its source files as executable binaries in its source tree. This is a sub-optimal practice that could lead to unwanted risks. We did not find any method this could be exploited at this audit, but Volcano should avoid this practice altogether. We found the following files to have executable permissions:

1. `pkg/controllers/jobtemplate/constant.go`
2. `pkg/controllers/jobtemplate/jobtemplate_controller_handler_test.go`
3. `pkg/controllers/jobtemplate/jobtemplate_controller_action.go`
4. `pkg/controllers/jobtemplate/jobtemplate_controller_util.go`
5. `pkg/controllers/jobtemplate/jobtemplate_controller_handler.go`
6. `pkg/controllers/jobtemplate/jobtemplate_controller_action_test.go`
7. `pkg/controllers/jobtemplate/jobTemplate_controller_util_test.go`
8. `pkg/controllers/jobtemplate/jobtemplate_controller.go`
9. `pkg/controllers/jobflow/constant.go`
10. `pkg/controllers/jobflow/jobflow_controller_handler.go`
11. `pkg/controllers/jobflow/jobflow_controller_util_test.go`
12. `pkg/controllers/jobflow/jobflow_controller_action_test.go`
13. `pkg/controllers/jobflow/state/pending.go`
14. `pkg/controllers/jobflow/state/running.go`
15. `pkg/controllers/jobflow/state/factory.go`
16. `pkg/controllers/jobflow/state/terminating.go`
17. `pkg/controllers/jobflow/state/failed.go`
18. `pkg/controllers/jobflow/state/succeed.go`
19. `pkg/controllers/jobflow/jobflow_controller_action.go`
20. `pkg/controllers/jobflow/jobflow_controller.go`
21. `pkg/controllers/jobflow/jobflow_controller_util.go`
22. `pkg/controllers/jobflow/jobflow_controller_handler_test.go`
23. `pkg/webhooks/admission/pods/mutate/annotation.go`
24. `pkg/webhooks/admission/pods/mutate/factory.go`
25. `pkg/webhooks/admission/pods/mutate/namespace.go`

Fix PR(s):

1. https://github.com/volcano-sh/volcano/pull/4171