PRESENTS

Cilium security audit

In collaboration with the Cilium maintainers, Open Source Technology Improvement Fund and The Linux Foundation

Authors

Adam Korczynski <adam@adalogics.com>
David Korczynski <david@adalogics.com>
Date: 13th February, 2023

Table of contents

Executive summary

In November and December 2022, Ada Logics carried out a security audit of Cilium on behalf of the Cloud Native Computing Foundation and the Open Source Technology Improvement Fund. The goal was to undertake a holistic security audit to draw on several security disciplines and evaluate and improve the security of Cilium.

Engagement objectives

The audit had the following high-level objectives:

1. **Threat model formalisation**: This part assessed the critical components of Cilium along with threat scenarios and architectural considerations relevant for Ciliums security posture.
2. **Manual code audit**: In this part, the auditing team manually audited the Cilium code base.
3. **Improve OSS-Fuzz fuzzing suite**: Cilium already had a decent fuzz test suite in place, and in this part the auditing team assessed the test coverage for possible improvements.
4. **SLSA review**: The audit team carried out a SLSA review to assess the compliance level of Cilium.

Process

Ada Logics commenced the engagement with an initial assessment of the Cilium project evaluating documentation, source code, previous vulnerabilities, maintainer and community responses to code issues and more. The purpose of this part was to get an initial understanding of how to address the hands-on part of the audit.

Next, Ada Logics drafted the threat model and began the fuzzing as well as manual auditing process. The threat model evolved in parallel with the fuzzing and manual work. Fuzzers were committed upstream ad-hoc and added to Ciliums OSS-Fuzz integration. Security concerns identified in the manual audit were shared with the Cilium team over two occasions: Once half-way through the audit and once at the end of the audit.

Finally, Ada Logics drafted the report detailing the overall engagement and shared this with the Cilium team. Together, the teams refined the report for publication.

Results and conclusions

In total 13 fuzzers were added to the test suite 3 of which were written for a critical 3rd-party dependency to Cilium. An OSS-Fuzz integration was also set up for this dependency. 22 security concerns were identified during the manual auditing.

The overall conclusion is that Cilium is a well-secured project. The audit found no critical or high severity vulnerabilities and found a lot of positives about the security of Cilium. This included both the code displaying positive security awareness as well as the maintainers having thorough understanding about the security posture of Cilium.

Project Summary

The auditors of Ada Logics were:

Name	Title	Email
Adam Korczynski	Security Engineer, Ada Logics	Adam@adalogics.com
David Korczynski	Security Researcher, Ada Logics	David@adalogics.com

The maintainers of Cilium that were involved were:

Name	Title	Email
Joe Stringer	Software Engineer, Isovalent	joe@isovalent.com
Liz Rice	Chief Open Source Officer, Isovalent	liz@isovalent.com
Mathieu Payeur Levallois	Director of Engineering, Isovalent	mpl@isovalent.com

The following facilitators of OSTIF were engaged in the audit:

Name	Title	Email
Derek Zimmer	Executive Director, OSTIF	Derek@ostif.org
Amir Montazery	Managing Director, OSTIF	Amir@ostif.org

Audit Scope

The following assets were in scope of the audit.

Cilium

Repository	https://github.com/cilium/cilium
Language	Go, C

Threat model formalisation

In this section we outline the threat model of Cilium. We first outline the core components of Cilium's architecture. Next, we specify the threat actors that could have a harmful impact on a Cilium deployment. Finally we exemplify several threat scenarios based on the observations made in the architecture overview and the specified threat actors.

The threat modelling has been conducted based on the following public resources:

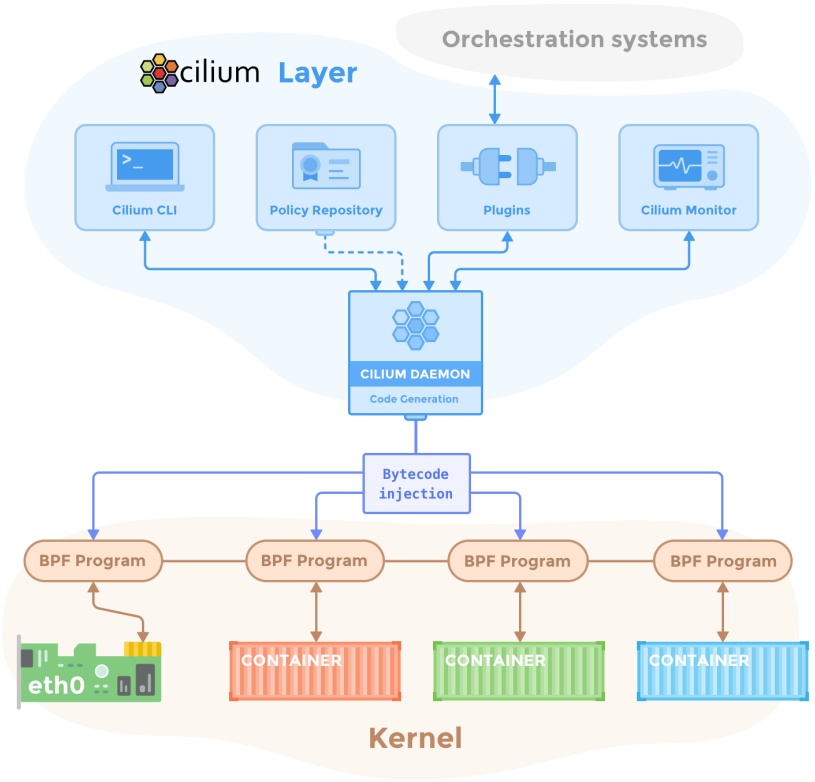
- Cilium's documentation including README files from Ciliums repository.
- Cilium's source code at https://github.com/cilium/cilium.
- 3rd-party literature, documentation and media.
- Feedback from Cilium maintainers

The intended audience for the threat model is the three target groups:

1. Security researchers who wish to contribute to the security posture of Cilium.
2. Maintainers of Cilium.
3. Users of Cilium.

It is expected that the threat model evolves over time based on both how Cilium and adoption evolves. As such, threat modelling should be seen as an ongoing effort. Future security disclosures to the Cilium security team are opportunities to evaluate the threat model of the affected components and use cases of the reported disclosure.

Cilium architecture



Daemon

The Cilium daemon runs on each node in the cluster and interacts with the container runtime and Kubernetes via plugins to set up networking and security policies. At a high-level, it accepts configuration via Kubernetes or APIs that describe networking, service load-balancing, network policies, and visibility & monitoring requirements.
The daemon listens for events from orchestration systems such as Kubernetes to learn when containers or workloads are started and stopped. It manages/creates the eBPF programs which the Linux kernel uses to control all network access in / out of those containers.

Source files

- https://github.com/cilium/cilium/tree/master/daemon

Policy repository

The policy repository has a list of policy rules that collectively make up a security policy. It is created by the Daemon when the core policy components are initialised. The Daemon is the only component that interacts with the policy repository. Some actions performed by the Daemon against the policy repository are:

- Adding new policy rules
- Deleting policy rules

CNI Plugin

Cilium interacts with Kubernetes via Kubernetes' Container Network Interface. Kubernetes starts Ciliums CNI Plugin when a pod gets scheduled or stopped on a node.

Most logic of the CNI plugin is in its `main.go`, where the `ADD`, `CHECK`, `DEL` commands are implemented.

Source files:

- https://github.com/cilium/cilium/tree/master/plugins/cilium-cni

eBPF Datapath

One of the central innovations in Cilium is the use of eBPF programs to observe and filter events in the kernel. This is achieved by having eBPF programs attached to hook points, including within the eXpress Data Path and in the traffic control subsystem. Specifically, the hooks applied by eBPF can be summarised as follows:

1. **XDP**: An early XDP BPF hook that is triggered at the earliest possible point in the network driver. The program runs when a packet is received.
2. **Traffic control (tc)**: The next eBPF program is invoked a few steps after the networking stack has carried out initial processing of the packet. At this stage, Cilium has access to the sk_buff (socket buffer).
3. **Socket hooks**: When the packet reaches Layer 4, Cilium has eBPF programs that attach to a cgroup and run on TCP events. "The socket send/recv hook runs on every send operation performed by a TCP socket. At this point the hook can inspect the message and either drop the message, send the message to the TCP layer, or redirect the message to another socket. Cilium uses this to accelerate the datapath redirects as described below."[1]

The eBPF programs are run at the kernel level and, therefore, operate at a privileged level of the system where access could theoretically impact the integrity of the system. However, eBPF programs are written in a subset of the C programming language that helps reduce the complexity of the programs, e.g. by disallowing unbounded loops to avoid deadlocks, and the impact of this is that the programs become more simple to analyse. The subset of C does not itself guarantee memory safety of the programs, however, to ensure further safety and guard against potential issues, the eBPF programs must be accepted by the eBPF verifier before being loaded into the kernel [https://docs.kernel.org/bpf/verifier.html].

[1] https://docs.cilium.io/en/stable/concepts/ebpf/intro/#:~:text=The%20socket%20send,as%20described%20below.

Threat actor enumeration

In this section we define the threat actors that we considered throughout the audit. These actors are defined pro-actively from studying the architecture of Cilium, and also by using a bottom-up approach based on findings in the Cilium code. Buttom-up in this context means when a coding issue is found we reason about what type of potential attacker can leverage such an issue.

Threat actors is a classification of a person or group of people that could actively seek to negatively impact Ciliums security posture and/or use flaws in Ciliums security posture to negatively impact users of Cilium. In an attack scenario, a threat actor answers the question: “Who would carry out this attack?”.

Actor	**Description**
Cilium contributor	Cilium is an open-source project that accepts contributions from the community. Contributors can intentionally or unintentionally commit vulnerable code to Cilium.
Cilium maintainer	A person that maintains the public Cilium code repository and is considered a gatekeeper for additions to the project.
3rd-party dependency contributors	Cilium uses open-source dependencies, many of which accept contributions from the community. A 3rd-party dependency contributor is a person who contributes code to projects in Ciliums dependency tree.
3rd-party dependency maintainer	A 3rd-party dependency maintainer is a person that manages public code repositories that Cilium depends on.
Internal user with limited privileges	An actor that has access to a host that runs Cilium, but who has lower privileges than Cilium itself.
Internal attacker	An actor that has escaped a container on a host running Cilium.
External attacker	External attacker on the internet.
Cilium administrators	Cilium administrators manage the Cilium clusters.

Threat surface enumeration

In this section we iterate through the threat surface that we considered when auditing the code. This was created iteratively as the audit progressed and the more we learned about the Cilium codebase.

Policy enforcement bugs
Cilium's policy model must be robust, secure and consistent. Users should be able to specify policies that Cilium follows. Implementation errors in the policy enforcement code can lead to issues, e.g. policies are assumed to block something they do not block. In this context we are referring to implementation issues in Cilium and not issues written in the policies themselves (for which there are suitable tools for learning and crafting policies https://networkpolicy.io). Users should be able to expect their policies to be enforced correctly by Cilium, however, implementation errors may allow an attacker to circumvent an ill-enforced policy. There are two examples of this in past advisorised https://github.com/cilium/cilium/security/advisories/GHSA-wc5v-r48v-g4vh and https://github.com/cilium/cilium/security/advisories/GHSA-c66w-hq56-4q97

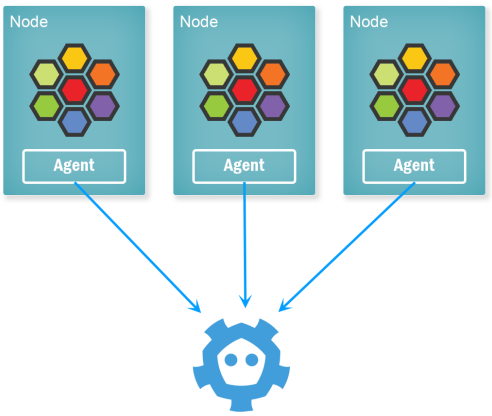
Issues in the eBPF code
The eBPF programs of Cilium are susceptible to logical issues (note, we found no such issues in the actual auditing). This includes any filtering that they may promise from a logical perspective but do not successfully enforce. For example, in the event the user specifies certain policies then these policies must be accurately handled by the eBPF programs.

The Linux kernel eBPF verifier performs an exhaustive analysis of the eBPF programs, including analysis of all possible execution paths, range analysis and more, which together ensure the safety of a given eBPF program. The verifier has been under thorough analysis, such as academic research focusing on formal verification of its range analysis [https://sanjit-bhat.github.io/assets/pdf/ebpf-verifier-range-analysis22.pdf]. The consequence of this is that the eBPF programs come with a high standard of security by default with respect to memory corruption issues. For these reasons and due to timing constraints, in this audit we do not inspect the eBPF programs for possible memory corruption issues as problems in this context should be caught by the verifier.

In general, the security of Cilium's eBPF programs are dependent on the eBPF verifier as well as the eBPF compiler toolchain. The verifier itself is complex and not limited to the same subset as eBPF programs, and has previously had issues e.g. CVE-2020-8835. Therefore, limitations in the eBPF verifier or eBPF compiler toolchain can impact the integrity of the kernel when used with software such as Cilium. Importantly in this context is that after Cilium runs on the node, the node cannot run unprivileged eBPF programs because Cilium disables this ability for the rest of the kernel run time (using `sysctl kernel.unprivileged_bpf_disabled=1`). This is done in Cilium in order to prevent the class of verifier vulnerabilities like CVE-2020-8835. This is also described here: https://docs.cilium.io/en/stable/bpf/#hardening.

Kvstore

Cilium can share data about cluster-wide state with all Cilium agents in two ways: in CRD's or in a global key-value store. The two production-grade key-value stores supported are etcd and Consul, with etcd being the most widely used. Cilium agents connect to the key-value store and obtain the data in the kv-store to effectuate the desired state across pods:



If Cilium is deployed in kv-store mode, the Cilium etcd/Consul endpoints represent an attack surface for the cluster which an attacker with local network access could attempt to exploit. If an attacker was able to compromise the kv-store instance to a degree where they could modify the entries of the store, they would de-facto have full control of the cluster.
This attack surface exists even if Cilium is not deployed in kv-store mode, in that Kubernetes per default uses etcd. As such, Cilium's kv-store mode does not expose increased attack surface but depends on Kubernetes' threat model for Cilium-specific configuration data.
Proper configuration by the Cilium user of the kv-store is critical in preventing exploitation of the added attack surface from the kv-store.
Depending on the configuration of the cluster, an attacker could utilize other properties of the kv-store deployment. If the kv-store runs in a container with root privileges, the attacker has a higher chance of escalating to other nodes in the cluster.

Supply-chain & runtime environment attacks

Cilium acts as a framework to route traffic through the Linux Kernel's eBPF runtime in cloud-native use cases. As such, the security of Kubernetes and the Linux Kernel affects Cilium users in several ways. From one side, vulnerabilities in either Kubernetes or the Linux Kernel's eBPF implementation may bring Cilium users at risk despite the root cause not stemming from Cilium itself. From another side, Cilium may not interact with these 3rd-party ecosystems - like Kubernetes the eBPF datapath - correctly which could lead to security issues.

Besides large ecosystems like Kubernetes and the Linux kernel, Cilium also depends on 3rd-party libraries for specific functionality in Cilium. A malicious attacker may find vulnerabilities in or intentionally commit vulnerable code to a 3rd-party dependency of Cilium. This could have an effect on Ciliums security posture. The attack surface through 3rd-party dependencies varies based on the extent to which the dependencies are used. For example, the library `github.com/vishvananda/netlink` is widely used across the entire cilium repository, whereas others are only used in tests.

As an example, Cilium has previously been affected by vulnerabilities in Envoy which Cilium uses to enforce certain L7 policies. These are described in detail here:

- https://github.com/cilium/cilium/security/advisories/GHSA-6hf9-972x-wff3
- https://github.com/cilium/cilium/security/advisories/GHSA-9hx8-3wfx-q2vw

We highlight here that the SLSA review given below outlines several of the practices that Cilium deploys to protect against supply chain security attacks.

Complex code assumptions regarding trusted input

Throughout the audit we found several pieces of code that alone did not constitute a security vulnerability. However, the code had certain properties where minor changes in the code base could result in security vulnerabilities without necessarily being obvious. Issue 4 and 5 are examples of this, where unbounded memory allocation happens. At the given moment we did not find these issues to be vulnerable, however, it is code that may be re-used improperly at a later stage can introduce a potential issue, where it may not be obvious that it is improperly re-used.

In essence, we consider this as an instance of complex code where assumptions of input to functions are not obvious. Consequently, it is not obvious from the code that it is indeed safe, where it could be made more obvious. A complex code base has security implications on the project - albeit indirectly. Complex code conceals vulnerabilities which in turn makes them arduous to identify in merged code and during code reviews before the code is even merged. In addition, complex code increases the overhead of contributing to the project which in turn might lead to fewer security issues being uncovered by the community.

Fuzzing

Having high fuzz test coverage is an important element of the security measures of a project. At the commencement of this audit, Cilium had carried out extensive work to integrate fuzzing in a previous fuzzing audit also performed by Ada Logics. In this security audit, Ada Logics did an assessment of where the project could improve its fuzzing. The high-level tasks that we carried out were:

Improve test coverage of Cilium
Ada Logics wrote 10 fuzzers targeting policy handling as well as use of critical dependencies.

Add fuzzing to critical 3rd-party library
Ada Logics made an initial integration on OSS-Fuzz for the extensively used `github.com/vishvananda/netlink` dependency. In addition we wrote 3 fuzzers for APIs used by Cilium. `github.com/vishvananda/netlink` has not been integrated into OSS-Fuzz at this time due to lack of upstream response.

Cilium Fuzzers written during the audit

#	Name	Package
1	`FuzzCiliumNetworkPolicyParse`	`pkg/k8s/apis/cilium.io/v2`
2	`FuzzCiliumClusterwideNetworkPolicyParse`	`pkg/k8s/apis/cilium.io/v2`
3	`FuzzTest`	`pkg/policy`
4	`FuzzRegenerateGatewayConfigs`	`pkg/egressgateway`
5	`FuzzParseCEGP`	`pkg/egressgateway`
6	`FuzzGetLookupTable`	`pkg/maglev`
7	`FuzzRoutes`	`pkg/testutils`
8	`FuzzListRules`	`pkg/testutils`
9	`FuzzMapSelectorsToIPsLocked`	`pkg/fqdn`
10	`FuzzNodeHandler`	`pkg/datapath/linux`

Fuzzer descriptions

FuzzCiliumNetworkPolicyParse
Parses a pseudo-randomized CiliumNetworkPolicy.

FuzzCiliumClusterwideNetworkPolicyParse

Parses a pseudo-randomized CiliumClusterwideNetworkPolicy.

FuzzTest
Creates a pseudo-randomized api rule and calls its resolveEgressPolicy method.

FuzzRegenerateGatewayConfigs
Creates a Manager with pseudo-random nodes and policy configs and invokes the managers regenerateGatewayConfigs() method.

FuzzParseCEGP
Parses a pseudo-randomized CiliumEgressGatewayPolicy.

FuzzGetLookupTable
Passes a pseudo-randomized map of backends to GetLookupTable().

FuzzRoutes
Passes a pseudo-randomized Route to multiple APIs in pkg/datapath/linux/route.

FuzzListRules
Passes a pseudo-randomized Rule to ListRules().

FuzzMapSelectorsToIPsLocked
Passes a pseudo-randomized map of FQDNSelectors to MapSelectorsToIPsLocked().

FuzzNodeHandler
Creates a linux node handler and a Node and passes the node to multiple methods of the linux node handler.

Netlink Fuzzers written during the audit

#	Name	Package
1	`FuzzLinkByName`	`netlink`
2	`FuzzDeserializeRoute`	`pkg/k8s/apis/cilium.io/v2`
3	`FuzzParseRawData`	`pkg/policy`

FuzzLinkByName
Passes a pseudo-random string to `LinkByName()` which returns a `Link`. Passes that `Link` to `AddrList()`.

FuzzDeserializeRoute
Passes raw data to `deserializeRoute()`.

FuzzParseRawData

Passes raw data to `parseRawData()`.

Supply chain

Scorecard

At the time of this audit, Cilium has not integrated the Scorecard project.

Scorecard is an open source tool that helps projects assess security risks in their code base. It performs a series of checks and scores the overall security posture on a scale from 1-10. If the project scores low, Scorecard will provide recommendations for remediation and mitigation.

Cilium can use Scorecard for its own repository to increase transparency on the overall security posture of the project. This will help demonstrate what Cilium is doing right, and it will make it clear to the community which actionable issues require contributions. Integrating Scorecard into Cilium would add the OpenSSF Scorecard badge to the README, where Cilium also has the "OpenSSF best practices" badge.

The Cilium community could consider requiring 3rd-party dependencies to integrate with Scorecard to increase transparency of Cilium's own supply chain. We recommend discussing this in a public Github issue. In case this is something the community would like enforced, it too could be a shared effort to bring Scorecard to Ciliums own supply chain.

More information on Scorecard can be found below:

- https://securityscorecards.dev
- https://github.com/ossf/scorecard

SLSA review

SLSA is a framework designed to mitigate threats against the software supply chain. Its purpose is to ensure integrity and prevent tampering of software artifacts. SLSA aims to ensure the integrity of the source, the build and the availability of software artifacts todefend against a series of known attack vectors of the software supply-chain[2].

Overall, Cilium is not yet SLSA 1 compliant, but with a few improvements can reach level 3.

Ada Logics follows the specification of SLSA v0.1 that is outlined here: https://slsa.dev/spec/v0.1/requirements. This version of the compliance requirements is currently in alpha and is likely to change.
Given the early stages of SLSA, projects looking to comply should consider compliance to be an on-going process with regular assessment on whether all requirements are satisfied. Even when SLSA levels are met, Cilium should work actively on maintaining the achieved

[2] https://slsa.dev/spec/v0.1/threats

levels, and we recommend that future security audits include an assessment on how Cilium performs in maintaining the achieved SLSA compliance level. In this audit, several of the requirements were assessed by Cilium maintainers, and as a consequence of this introspective work, we - Ada Logics - consider the Cilium project to be self-sustaining in progressing with compliance. A Github issue has been created here for that purpose: https://github.com/cilium/cilium/issues/22740.

Cilium performs well in the Source and Build categories and meets most requirements in these categories up to and including level 3.
The “Two-person viewed” criteria under “Source” is fulfilled, since two Cilium maintainers are involved in merging pull requests. Not all pull requests are explicitly approved by the second maintainer, and we recommend that the second maintainer involved in a pull-request review approves it before merging to avoid doubt of fulfilling this requirement.

The primary missing domain for SLSA compliance is the provenance which is not generated during releases. The https://github.com/slsa-framework/slsa-github-generator project offers a series of tools to generate SLSA level 3-compliant provenance attestation for Github-hosted projects. These tools are created to specifically satisfy the provenance-section of the three SLSA categories and will be a useful addition to Ciliums SLSA compliance. When Cilium makes the provenance available, it has achieved SLSA level 1 compliance.

SLSA compliance overview

Requirement	SLSA 1	SLSA 2	SLSA 3	SLSA 4
Source - Version controlled		✓	✓	✓
Source - Verified history			✓	✓
Source - Retained indefinitely			✓ (18 mo.)	✓
Source - Two-person reviewed				✓
Build - Scripted build	✓	✓	✓	✓
Build - Build service		✓	✓	✓
Build - Build as code			✓	✓
Build - Ephemeral environment			✓	✓

Build - Isolated			✓	✓
Build - Parameterless				✓
Build - Hermetic				⛔
Build - Reproducible				⛔
Provenance - Available	⛔	⛔	⛔	⛔
Provenance - Authenticated		⛔	⛔	⛔
Provenance - Service generated		⛔	⛔	⛔
Provenance - Non-falsifiable			⛔	⛔
Provenance - Dependencies complete				⛔
Common - Security	Not defined by SLSA requirements			
Common - Access				✓
Common - Superusers				✓

Issues

This section presents issues found during the audit. Find an up-to-date report on the status of these issues at https://github.com/cilium/cilium/issues/23121

#	Title	Severity
1	Out of bounds file read in certificate manager GetSecrets()	Low
2	Missing central documentation on using Cilium securely	Medium
3	Handlers of the Cilium Docker plugin do not limit the size of the http request body before decoding it.	Informational
4	Possible memory exhaustion from CNI template rendering	Informational
5	Possible excessive memory allocation	Low
6	Race condition in Hubble Relay Peer Manager	Low
7	Race condition in pkg/policy.Repository.LocalEndpointIdentityRemoved()	Low
8	Deprecated 3rd-party library	Informational
9	TOCTOU race condition in endpoint file move helper function	Low
10	Redundant return statements	Informational
11	Redundant imports	Informational
12	Redundant function parameters	Informational
13	TOCTOU race condition in sockops bpftoolLoad	Low
14	Level of trust for input from cloud providers is too high	Low
15	BGP configuration file is read entirely into memory	Low
16	Race condition when starting operator apiserver	Low
17	Bad code practice: Identical identifier of import and variable	Low
18	Deadlock from locked mutex	Medium
19	Possible type confusions	Low
20	Ill-defined contexts	Informational
21	Use of deprecated TLS version	Informational
22	Deprecated function calls	Low

1: Out of bounds file read in certificate manager `GetSecrets()`

Overall severity:	**Low**
ID:	ADA-CIL-1
Location:	https://github.com/cilium/cilium/blob/e4af2f9a0c28c84090c7142e241c2749a3f84ed9/pkg/crypto/certificatemanager/certificate_manager.go#L42
CWE:	• CWE-125: Out-of-bounds Read

Description

An attacker that can create a symlink in the `certPath` of a given namespace can potentially read files outside of the `certPath`.

https://github.com/cilium/cilium/blob/314ca7baeff4f568ffc0bad95124e40665b1f88c/pkg/crypto/certificatemanager/certificate_manager.go#L42

```
func (m *Manager) GetSecrets(ctx context.Context, secret *api.Secret, ns string) (string,
map[string][]byte, error) {
        ...
        nsName := filepath.Join(ns, secret.Name)
        ...
        certPath := filepath.Join(m.rootPath, nsName)
        files, ioErr := os.ReadDir(certPath)
        if ioErr == nil {
                secrets := make(map[string][]byte, len(files))
                for _, file := range files {
                        var bytes []byte

                        path := filepath.Join(certPath, file.Name())
                        bytes, ioErr = os.ReadFile(path)
                        if ioErr == nil {
                                secrets[file.Name()] = bytes
                        }
                }

                if len(secrets) == 0 && ioErr != nil {
                        return nsName, nil, ioErr
                }
                return nsName, secrets, nil
        }
        secrets, err := m.k8sClient.GetSecrets(ctx, ns, secret.Name)
        return nsName, secrets, err
}
```

The root cause of the issue is that `os.ReadFile()` resolves symlinks. For example, an attacker could create a symlink in certPath with the filename “tls.key” linking to a cert file elsewhere on the machine. m.GetSecrets would then first read all the files in certPath:

```
files, ioErr := os.ReadDir(certPath)
```

Cilium then proceeds to read filenames. Here, `path` would be correct, but Cilium would resolve a symlink. bytes could be the file contents of the file being linked to in a symlink.

```
path := filepath.Join(certPath, file.Name())
bytes, ioErr = os.ReadFile(path)
```

The file contents are stored by filename:

```
secrets[file.Name()] = bytes
```

Note that there is also a TOCTOU race condition in this part of the code. `GetSecrets()` only uses the file names in the directory and subsequently reads the files on their path, the files could theoretically be changed in the time between the invocation of `os.ReadDir()` and `os.ReadFile()`.

The ‘certPath’ is considered a privileged directory created by the privileged user. Thus, there is no escalation of privileges since the privileged user will also have access to the linked files in a symlink.”

2: Missing central documentation on using Cilium securely

Overall severity:	Medium
ID:	ADA-CIL-2
Location:	Documentation
Fix:	https://github.com/cilium/cilium/pull/23599

Description

Misconfiguration is a major security problem with misconfiguration currently ranking 5 on OWASP's top ten (2021). In cloud environments data indicates misconfiguration is the most significant reason for unnecessary security risk exposure[3]. Cilium users should be able to confidently follow industry standards in their deployments, and the documentation should provide this information. Cilium should also list whether there are limitations of the default setting from a security perspective.

Cilium's documentation is extensive and provides information on both internals, the fundamentals of eBPF as well as usage and getting started. The documentation also includes walk-throughs of using various common components that might be used in an application deployment alongside Cilium:

- How to secure gRPC: https://docs.cilium.io/en/v1.12/gettingstarted/grpc/
- Getting Started Securing Elasticsearch: https://docs.cilium.io/en/v1.12/gettingstarted/elasticsearch/
- How to Secure a Cassandra Database: https://docs.cilium.io/en/v1.12/gettingstarted/cassandra/
- Getting Started Securing Memcached: https://docs.cilium.io/en/v1.12/gettingstarted/memcached/

These are great tutorials to include in the documentation, however, users are required to read through the entire documentation to extract all security-relevant information on configuration. This is a time-consuming task, magnified by some tutorials being in a step-by-step format requiring setting up a test environment. The consequences of this may be that some CIlium users postpone extracting all documentation relevant to securely configuring Cilium, or that some users go through the documentation but fail to absorb all security-relevant information because it is scattered across the entire documentation.

We recommend creating a single-page piece of documentation that in short-form covers all known configuration points that could put users at risk if misconfigured. This will ensure that adopters can quickly and confidently go through all best-practice configurations and ensure they follow the recommended standards.
For examples of security best practice documentation, see:

[3] https://www.armosec.io/blog/what-we-learned-from-scanning-over-10k-kubernetes-clusters/

- https://istio.io/latest/docs/ops/best-practices/security/
- https://github.com/kubeedge/community/blob/master/sig-security/sig-security-audit/KubeEdge-threat-model-and-security-protection-analysis.md

3. Handlers of the Cilium Docker plugin do not limit the size of the http request body before decoding it

Overall severity:	Low
ID:	ADA-CIL-3
Location:	https://github.com/cilium/cilium/blob/a5901e562faae14b86fc02d6ed195464134e9586/plugins/cilium-docker/driver/driver.go
CWE:	• CWE-400: Uncontrolled Resource Consumption

Description

The Cilium Docker plugin does not check the size of the request body before decoding it. An http request with a large body could cause temporary DoS of the machine. Local testing demonstrated a denial of the machine of ~ 10 seconds before Go terminated.

The issue exists here:
https://github.com/cilium/cilium/blob/a5901e562faae14b86fc02d6ed195464134e9586/plugins/cilium-docker/driver/driver.go#L327

```
func (driver *driver) createNetwork(w http.ResponseWriter, r *http.Request) {
	var create api.CreateNetworkRequest
	err := json.NewDecoder(r.Body).Decode(&create)
	if err != nil {
		sendError(w, "Unable to decode JSON payload: "+err.Error(),
http.StatusBadRequest)
		return
	}
	log.WithField(logfields.Request, logfields.Repr(&create)).Debug("Network
Create Called")
	emptyResponse(w)
}
```

As well as these places:

- https://github.com/cilium/cilium/blob/a5901e562faae14b86fc02d6ed195464134e9586/plugins/cilium-docker/driver/driver.go#L340
- https://github.com/cilium/cilium/blob/a5901e562faae14b86fc02d6ed195464134e9586/plugins/cilium-docker/driver/driver.go#L360
- https://github.com/cilium/cilium/blob/a5901e562faae14b86fc02d6ed195464134e9586/plugins/cilium-docker/driver/driver.go#L436
- https://github.com/cilium/cilium/blob/a5901e562faae14b86fc02d6ed195464134e9586/plugins/cilium-docker/driver/driver.go#L456
- https://github.com/cilium/cilium/blob/a5901e562faae14b86fc02d6ed195464134e9586/plugins/cilium-docker/driver/driver.go#L474
- https://github.com/cilium/cilium/blob/a5901e562faae14b86fc02d6ed195464134e9586/plugins/cilium-docker/driver/driver.go#L513

- https://github.com/cilium/cilium/blob/a5901e562faae14b86fc02d6ed195464134e9586/plugins/cilium-docker/driver/ipam.go#L68
- https://github.com/cilium/cilium/blob/a5901e562faae14b86fc02d6ed195464134e9586/plugins/cilium-docker/driver/ipam.go#L81
- https://github.com/cilium/cilium/blob/a5901e562faae14b86fc02d6ed195464134e9586/plugins/cilium-docker/driver/ipam.go#L93
- https://github.com/cilium/cilium/blob/a5901e562faae14b86fc02d6ed195464134e9586/plugins/cilium-docker/driver/ipam.go#L142

This binary is not distributed or installed in Cilium environments. It's used for development purposes in some of the older testing infrastructure and we've been discussing getting rid of it.

4. Possible memory exhaustion from CNI template rendering

Overall severity:	Informational
ID:	ADA-CIL-4
Location:	https://github.com/cilium/cilium/blob/12b7b11e10c87dce2704d4252d22ff202a48ebc1/daemon/cmd/cni.go#L271
CWE:	• CWE-400: Uncontrolled Resource Consumption

Description

Ciliums CNI injection mechanism contains several memory allocations with no upper bounds that may end in a memory exhaustion vulnerability. We did not find a way for this to be vulnerable at the moment, however, we highlight this as a possible issue in that it may be exposed in the future. The root cause of the issue is that Cilium merges two byte slices without enforcing an upper limit on either. There are several places where this could fail, and we list these further below. First we present the dataflow of the CNI injection. The two byte slices being merged are:

1. A CNI template containing an aws CNI entry. This is the entry that gets added to the CNI configuration file.
2. The contents of the CNI configuration file.

The merging happens in `renderCNIConf()`:

```
func renderCNIConf(opts *option.DaemonConfig, confDir string) (cniConfig []byte, err
error) {

        if opts.CNIChainingMode == "aws-cni" {
                pluginConfig := renderCNITemplate(awsCNIEntry, opts)
                cniConfig, err = mergeExistingAWSCNIConfig(confDir, pluginConfig)
                if err != nil {
                        return nil, err
                }
        } else {
                ...
        }

        ...
        return cniConfig, nil
}
```

`renderCNIConf` creates the bytes that will later be added to the existing CNI configuration file. This happens in `renderCNITemplate`:

```
func renderCNITemplate(in string, opts *option.DaemonConfig) []byte {
        data := struct {
                Debug   bool
                LogFile string
```

```
        }{
                Debug:   opts.Debug,
                LogFile: opts.CNILogFile,
        }

        t := template.Must(template.New("cni").Parse(in))

        out := bytes.Buffer{}
        if err := t.Execute(&out, data); err != nil {
                panic(err) // impossible
        }
        return out.Bytes()
}
```

`renderCNITemplate` adds the values of the `Debug` and `CNILogFile` fields from the `DaemonConfig` to the `awsCNIEntry` template:

```
const awsCNIEntry = `
{
        "type": "cilium-cni",
        "enable-debug": {{.Debug | js }},
        "log-file": "{{.LogFile | js }}"
}
`
```

This template is then passed to `mergeExistingAWSCNFConfig` which reads the existing cni config file and adds the Cilium plugin:

```
func mergeExistingAWSCNIConfig(confDir string, pluginConfig []byte) ([]byte, error) {
        awsFiles := []string{"10-aws.conflist", "10-aws.conflist.cilium_bak"}
        found, err := findFile(confDir, awsFiles)
        if err != nil {
                return nil, fmt.Errorf("could not find existing AWS CNI config for
chaining %w", err)
        }

        contents, err := os.ReadFile(found)
        if err != nil {
                return nil, fmt.Errorf("failed to read existing AWS CNI config %s: %w",
found, err)
        }

        // We found the CNI configuration,
        // inject Cilium as the last chained plugin
        out, err := sjson.SetRawBytes(contents, "plugins.-1", pluginConfig)
        if err != nil {
                return nil, fmt.Errorf("failed to modify existing AWS CNI config at %s:
%w", found, err)
        }
        log.Infof("Inserting cilium in to CNI configuration file at %s", found)
        return out, nil
}
```

Finally the merged aws plugin and the existing file contents are written back to the CNI config file.

Since there are no upper bounds on the size of the template or the size of the existing CNI configuration file, Cilium may try to allocate excessive memory in the CNI merging workflow. At the moment, we have found 2 ways to manipulate the amount of allocated memory:

1. Create a DaemonConfig with a large `CNILogFile` string. This will create a large template. The failure can happen at the moment the template is created in case its size is enough to exhaust memory, or later in the CNI merging process.
2. The existing CNI configuration file is large. Once its contents are merged with the template, this may exhaust memory.

Recommendation
We acknowledge that this code does not constitute a bug at the moment. However, we recommend either leaving a comment in the code that highlights the unbounded memory allocations are in fact safe, or, add upper bounds to the template size and the existing CNI configuration file before reading it.

5: Possible excessive memory allocation

Overall severity:	Informational
ID:	ADA-CIL-5
Location:	https://github.com/cilium/cilium/blob/dac4c0691c2fd611308b28ed08b8ba2ee7f38b8c/pkg/labels/labels.go#L478
CWE:	• CWE-400: Uncontrolled Resource Consumption

Description

Cilium may be made to create a byte slice that exceeds the memory available on the machine in `labels.SortedList()`:

```
func (l Labels) SortedList() []byte {
	keys := make([]string, 0, len(l))
	for k := range l {
		keys = append(keys, k)
	}
	sort.Strings(keys)

	b := make([]byte, 0, len(keys)*2)
	buf := bytes.NewBuffer(b)
	for _, k := range keys {
		buf.Write(l[k].FormatForKVStore())
	}

	return buf.Bytes()
}
```

In similar fashion to issue 4, we did not find this to be vulnerable at this given moment, but we highlight it as a potential issue since unbounded memory allocations can lead to memory exhaustion. It is worth noting here that according to https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#syntax-and-character-set labels are constrained in size, label keys < 63 characters, but with a < 253 char potential prefix, values < 63 characters. Generously: below 400 ASCII characters, hence <400 bytes per label.

6: Race condition in Hubble Relay Peer Manager

Overall severity:	Low
ID:	ADA-CIL-6
Location:	https://github.com/cilium/cilium/blob/82c742f2e9fb65fb3fc392c7dcb3b4e22b69c650/pkg/hubble/relay/pool/manager.go#L143-L186
CWE:	• CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
Fix:	https://github.com/cilium/cilium/commit/43121d9d554604f708eb273d117b68960747812d

Description

`PeerManager.manageConnections()` creates a goroutine inside a loop and references a variable outside the goroutine. This can result in a race condition, whereby both the PeerManager and the peer name might change between they are declared and the time they are used inside the goroutine:
https://github.com/cilium/cilium/blob/82c742f2e9fb65fb3fc392c7dcb3b4e22b69c650/pkg/hubble/relay/pool/manager.go#L143-L186

```
func (m *PeerManager) manageConnections() {
	connTimer, connTimerDone := inctimer.New()
	defer connTimerDone()
	for {
		select {
		case <-m.stop:
			return
		case name := <-m.offline:
			m.mu.RLock()
			p := m.peers[name]
			m.mu.RUnlock()
			m.wg.Add(1)
			go func() {
				defer m.wg.Done()
				// a connection request has been made, make sure to attempt a connection
				m.connect(p, true)
			}()
		case <-connTimer.After(m.opts.connCheckInterval):
			m.mu.RLock()
			now := time.Now()
			for _, p := range m.peers {
				p.mu.Lock()
				if p.conn != nil {
					switch p.conn.GetState() {
					case connectivity.Connecting, connectivity.Idle, connectivity.Ready, connectivity.Shutdown:
						p.mu.Unlock()
						continue
					}
```

```
                                }
                                switch {
                                case p.nextConnAttempt.IsZero(),
p.nextConnAttempt.Before(now):
                                        p.mu.Unlock()
                                        m.wg.Add(1)
                                        go func() {
                                                defer m.wg.Done()
                                                m.connect(p, false)
                                        }()
                                default:
                                        p.mu.Unlock()
                                }
                        }
                        m.mu.RUnlock()
                }
        }
}
```

Minimal reproducer

The race condition is demonstrated with this reproducer that mimics the behaviour of `manageConnections()`. Towards the end of the program, a line with a panic is marked with yellow. This panic will only be triggered if `firstPeerName` and `secondPeerName` are different, ie. if the `p` changes before it is used inside the goroutine.

```
package main

import (
        "fmt"
        "github.com/cilium/cilium/pkg/lock"
        "sync"
)

type peer struct {
        mu   lock.Mutex
        name string
}

type m struct {
        mu    lock.RWMutex
        wg    sync.WaitGroup
        peers map[string]*peer
}

func (m *m) connect(p *peer) {
        fmt.Println(p.name)
}

func main() {
        peers := map[string]*peer{
                "peer1":   &peer{name: "peer1"},
                "peer2":   &peer{name: "peer2"},
                "peer3":   &peer{name: "peer3"},
```

```
                "peer4":   &peer{name: "peer4"},
                "peer5":   &peer{name: "peer5"},
                "peer6":   &peer{name: "peer6"},
                "peer7":   &peer{name: "peer7"},
                "peer8":   &peer{name: "peer8"},
                "peer9":   &peer{name: "peer9"},
                "peer510": &peer{name: "peer10"},
        }
        m := &m{peers: peers}
        m.mu.RLock()
        for _, p := range m.peers {
                p.mu.Lock()
                p.mu.Unlock()
                m.wg.Add(1)
                go func() {
                        defer m.wg.Done()
                        firstPeerName := fmt.Sprintf("%s", p.name)
                        secondPeerName := fmt.Sprintf("%s", p.name)
                        if secondPeerName != firstPeerName {
                                panic(fmt.Sprintf("We won the race:\n'p.name' is %s and
'firstPeerName' is %s\n", p.name, firstPeerName))
                        }
                        m.connect(p)
                }()
                fmt.Println("End of loop")
        }
        m.mu.RUnlock()
}
```

For more info, see Effective Go: goroutines.

7: Race condition in `pkg/policy.Repository.LocalEndpointIdentityRemoved()`

Overall severity:	Low
ID:	ADA-CIL-7
Location:	• https://github.com/cilium/cilium/blob/bbcadc43758b7e3c89d0ef9a39266fff1bc41849/pkg/policy/repository.go#L404 • https://github.com/cilium/cilium/blob/314ca7baeff4f568ffc0bad95124e40665b1f88c/pkg/identity/identitymanager/manager.go#L150
CWE:	• CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

Description

Another case of creating a goroutine and using a variable inside the goroutine that was declared outside of the goroutine exists in `pkg/policy.Repository.LocalEndpointIdentityRemoved()`.

`(*Repository).LocalEndpointIdentityRemoved()` creates a goroutine:

```
func (p *Repository) LocalEndpointIdentityRemoved(identity *identity.Identity) {
	go func() {
		scopedLog := log.WithField(logfields.Identity, identity)
		scopedLog.Debug("Removing identity references from policy cache")
		p.Mutex.RLock()
		wg := p.removeIdentityFromRuleCaches(identity)
		wg.Wait()
		p.Mutex.RUnlock()
		scopedLog.Debug("Finished cleaning policy cache")
	}()
}
```

`LocalEndpointIdentityRemoved()` is called here:
https://github.com/cilium/cilium/blob/master/pkg/identity/identitymanager/manager.go

```
func (idm *IdentityManager) remove(identity *identity.Identity) {

	if identity == nil {
		return
	}

	idMeta, exists := idm.identities[identity.ID]
	if !exists {
		log.WithFields(logrus.Fields{
			logfields.Identity: identity,
		}).Error("removing identity not added to the identity manager!")
```

```
                return
        }
        idMeta.refCount--
        if idMeta.refCount == 0 {
                delete(idm.identities, identity.ID)
                for o := range idm.observers {
                        o.LocalEndpointIdentityRemoved(identity)
                }
        }

}
```

The for-loop (marked with blue) loops through all observers of the identitymanager, and starts a goroutine in each iteration. This will not always start one goroutine for each observer but might start multiple goroutines for the same observer.

For more info, see Effective Go: goroutines.

8: Deprecated 3rd-party library

Overall severity:	Informational
ID:	ADA-CIL-8
Location:	• https://github.com/cilium/cilium/blob/3f6dadc91c92bdf09a5dfaedb93047dc9f764cc3/tools/tools.go#L11
CWE:	• CWE-477: Use of Obsolete Function

Description:
Cilium has in its dependency tree `github.com/gogo/protobuf` which is deprecated. We recommend discontinuing all use of deprecated dependencies.

Kubernetes also relies on this library: https://github.com/kubernetes/kubernetes/issues/96564, and since Cilium uses kubernetes libraries, Cilium cannot remove this dependency until k8s does it.

9: TOCTOU race condition in endpoint file move helper function

Overall severity:	Low
ID:	ADA-CIL-9
Location:	• https://github.com/cilium/cilium/blob/314ca7baeff4f568ffc0bad95124e40665b1f88c/pkg/endpoint/directory.go#L50
CWE:	• CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

Description

A TOCTOU race condition exists in pkg/endpoint when moving files from one directory to another.
https://github.com/cilium/cilium/blob/314ca7baeff4f568ffc0bad95124e40665b1f88c/pkg/endpoint/directory.go#L50

```
func moveNewFilesTo(oldDir, newDir string) error {
	oldFiles, err := os.ReadDir(oldDir)
	if err != nil {
		return err
	}
	newFiles, err := os.ReadDir(newDir)
	if err != nil {
		return err
	}

	for _, oldFile := range oldFiles {
		exists := false
		for _, newFile := range newFiles {
			if oldFile.Name() == newFile.Name() {
				exists = true
				break
			}
		}
		if !exists {
			os.Rename(filepath.Join(oldDir, oldFile.Name()),
filepath.Join(newDir, oldFile.Name()))
		}
	}
	return nil
}
```

This could be exploited if an attacker can replace a file in oldDir after oldDir has been read and before the file is renamed.
The issue could potentially allow users to create files in directories that they should not be able to create files in.

This issue is rated `LOW`, because an attacker with this level of access can trigger the impact regardless of the TOCTOU race condition.

To replace a file in oldDir, the adversary already needs unix permissions to create/delete files in that directory. If that is given, they can just create a file in oldDir, which will then be moved to newDir if there is no file name conflict.”

10: Redundant return statements

Overall severity:	Informational
ID:	ADA-CIL-10
Location:	Several packages
CWE:	• CWE-1041: Use of Redundant Code

Description

https://github.com/cilium/cilium/blob/314ca7baeff4f568ffc0bad95124e40665b1f88c/cilium/cmd/debuginfo.go#L346

```
func writeJSONPathToOutput(buf bytes.Buffer, path string, suffix string, jsonPath string)
{
        data := buf.Bytes()
        db := &models.DebugInfo{}
        err := db.UnmarshalBinary(data)
        if err != nil {
                fmt.Fprintf(os.Stderr, "error unmarshaling binary: %s\n", err)
        }
        jsonStr, err := command.DumpJSONToString(db, jsonPath)
        if err != nil {
                fmt.Fprintf(os.Stderr, "error printing JSON: %s\n", err)
        }

        if path == "" {
                fmt.Println(jsonStr)
                return
        }

        fileName := fileName(path, suffix)
        writeFile([]byte(jsonStr), fileName)

        fmt.Printf("%s output at %s\n", jsonpathOutput, fileName)
        return
}
```

https://github.com/cilium/cilium/blob/00e40bb41683b9b3462a94879a93841751197629/daemon/cmd/config.go#L33

```
func (h *patchConfig) configModify(params PatchConfigParams, resChan chan interface{}) {
        ...
                d.TriggerDatapathRegen(policyEnforcementChanged, "agent configuration
update")
        }

        resChan <- NewPatchConfigOK()
        return
}
```

https://github.com/cilium/cilium/blob/de82f8c1a3cb92cde0f37fe627aaf1c313a37caf/daemon/cmd/policy.go#L430

```
func (d *Daemon) policyAdd(sourceRules policyAPI.Rules, opts *policy.AddOptions, resChan
chan interface{}) {
```

```
...
_, err = d.policy.RuleReactionQueue.Enqueue(ev)
	if err != nil {
		log.WithError(err).WithField(logfields.PolicyRevision,
newRev).Error("enqueue of RuleReactionEvent failed")
	}

	return
}
```

https://github.com/cilium/cilium/blob/de82f8c1a3cb92cde0f37fe627aaf1c313a37caf/daemon/cmd/policy.go#L644

```
func (d *Daemon) policyDelete(labels labels.LabelArray, res chan interface{}) {
if _, err := d.policy.RuleReactionQueue.Enqueue(ev); err != nil {
		log.WithError(err).WithField(logfields.PolicyRevision, rev).Error("enqueue
of RuleReactionEvent failed")
	}
	if err := d.SendNotification(monitorAPI.PolicyDeleteMessage(deleted,
labels.GetModel(), rev)); err != nil {
		log.WithError(err).WithField(logfields.PolicyRevision, rev).Warn("Failed
to send policy update as monitor notification")
	}

	return
}
```

https://github.com/cilium/cilium/blob/b1aa5374acf77a598e8f5a6d69cbc53122812d62/daemon/cmd/status.go#L1093

```
func (d *Daemon) startStatusCollector(cleaner *daemonCleanup) {
	...
	cleaner.cleanupFuncs.Add(func() {
		// If the KVstore state is not OK, print help for user.
		if d.statusResponse.Kvstore != nil &&
			d.statusResponse.Kvstore.State != models.StatusStateOk {
			helpMsg := "cilium-agent depends on the availability of
cilium-operator/etcd-cluster. " +
				"Check if the cilium-operator pod and etcd-cluster are
running and do not have any " +
				"warnings or error messages."
			log.WithFields(logrus.Fields{
				"status":              d.statusResponse.Kvstore.Msg,
				logfields.HelpMessage: helpMsg,
			}).Error("KVStore state not OK")

		}
	})
	return
}
```

https://github.com/cilium/cilium/blob/c147bbd3211f0ef19eb13f4daab07e100e6b72d5/operator/pkg/ciliumendpointslice/endpointslice.go#L173

```
func syncCESsInLocalCache(cesStore cache.Store, manager operations) {
	...
	log.Debug("Successfully synced all CESs locally")
```

```
        return
}
```

https://github.com/cilium/cilium/blob/c147bbd3211f0ef19eb13f4daab07e100e6b72d5/operator/pkg/ciliumendpointslice/endpointslice.go#L196

```
func (c *CiliumEndpointSliceController) Run(ces cache.Indexer, stopCh chan struct{}) {
        ...
        go wait.Until(c.worker, c.workerLoopPeriod, stopCh)

        go func() {
                defer utilruntime.HandleCrash()
        }()

        <-stopCh

        return
}
```

https://github.com/cilium/cilium/blob/c147bbd3211f0ef19eb13f4daab07e100e6b72d5/operator/pkg/ciliumendpointslice/endpointslice.go#L217

```
func syncCESsInLocalCache(cesStore cache.Store, manager operations) {
        ...
        log.Debug("Successfully synced all CESs locally")
        return
}
```

https://github.com/cilium/cilium/blob/e4ea0fa10af293b24fbe6b26307ec2709bb856e6/operator/pkg/ciliumendpointslice/manager.go#L168

```
func (c *cesMgr) addCEPtoCES(cep *cilium_v2.CoreCiliumEndpoint, ces *cesTracker) {
        ...
        // Increment the cepInsert counter
        ces.cepInserted += 1
        c.insertCESInWorkQueue(ces, DefaultCESSyncTime)
        return
}
```

https://github.com/cilium/cilium/blob/e4ea0fa10af293b24fbe6b26307ec2709bb856e6/operator/pkg/ciliumendpointslice/manager.go#L367

```
func (c *cesMgr) RemoveCEPFromCache(cepName string, baseDelay time.Duration) {
        ...
        } else {
                log.WithFields(logrus.Fields{
                        logfields.CEPName: cepName,
                }).Info("Attempted to retrieve non-existent CES, skip processing.")
        }

        return
}
```

https://github.com/cilium/cilium/blob/0b7919a9d8138a42dc82f3861372c665970fd22c/pkg/alibabacloud/eni/node.go#L86

```
func (n *Node) PopulateStatusFields(resource *v2.CiliumNode) {
	resource.Status.AlibabaCloud.ENIs = map[string]eniTypes.ENI{}

	n.manager.ForeachInstance(n.node.InstanceID(),
		func(instanceID, interfaceID string, rev ipamTypes.InterfaceRevision)
error {
			e, ok := rev.Resource.(*eniTypes.ENI)
			if ok {
				resource.Status.AlibabaCloud.ENIs[interfaceID] =
*e.DeepCopy()
			}
			return nil
		})

	return
}
```

https://github.com/cilium/cilium/blob/c5cbf403dbe355ecbb80dfc8d7a8ed4da45c43bd/pkg/aws/eni/node.go#L105

```
func (n *Node) PopulateStatusFields(k8sObj *v2.CiliumNode) {
	k8sObj.Status.ENI.ENIs = map[string]eniTypes.ENI{}

	n.manager.ForeachInstance(n.node.InstanceID(),
		func(instanceID, interfaceID string, rev ipamTypes.InterfaceRevision)
error {
			e, ok := rev.Resource.(*eniTypes.ENI)
			if ok {
				k8sObj.Status.ENI.ENIs[interfaceID] = *e.DeepCopy()
			}
			return nil
		})

	return
}
```

https://github.com/cilium/cilium/blob/f3a4c4d204cf84af3d40f4782aa68e7c2da98440/pkg/endpoint/events.go#L68

```
func (ev *EndpointRegenerationEvent) Handle(res chan interface{}) {
	...
	res <- &EndpointRegenerationResult{
		err: err,
	}
	return
}
```

https://github.com/cilium/cilium/blob/f3a4c4d204cf84af3d40f4782aa68e7c2da98440/pkg/endpoint/events.go#L185

```
func (ev *EndpointNoTrackEvent) Handle(res chan interface{}) {
	...
	res <- &EndpointRegenerationResult{
		err: nil,
```

```
        }
        return
}
```

https://github.com/cilium/cilium/blob/f3a4c4d204cf84af3d40f4782aa68e7c2da98440/pkg/endpoint/events.go#L251

```
func (ev *EndpointPolicyVisibilityEvent) Handle(res chan interface{}) {
        ...
        e.visibilityPolicy = nvp
        res <- &EndpointRegenerationResult{
                err: nil,
        }
        return
}
```

https://github.com/cilium/cilium/blob/c112a3e59dcc988cf3a9901b48bce9583cfb3581/pkg/ipam/allocator/podcidr/podcidr.go#L420

```
func (n *NodesPodCIDRManager) Delete(node *v2.CiliumNode) {
        ...
        n.ciliumNodesToK8s[node.Name] = &ciliumNodeK8sOp{
                op: k8sOpDelete,
        }
        n.k8sReSync.Trigger()
        return
}
```

https://github.com/cilium/cilium/blob/295d32957db4defae2c1dc594ff9b284c6513d4f/pkg/maps/lbmap/ipv4.go#L224

```
func (in *pad2uint8) DeepCopyInto(out *pad2uint8) {
        copy(out[:], in[:])
        return
}
```

https://github.com/cilium/cilium/blob/d5227f1baed6eb56865dc08275e6560bfff27cce/pkg/maps/lxcmap/lxcmap.go#L113

```
func (in *pad4uint32) DeepCopyInto(out *pad4uint32) {
        copy(out[:], in[:])
        return
}
```

https://github.com/cilium/cilium/blob/314ca7baeff4f568ffc0bad95124e40665b1f88c/pkg/tuple/tuple.go#L42

```
func (in *buff256uint8) DeepCopyInto(out *buff256uint8) {
        copy(out[:], in[:])
        return
}
```

https://github.com/cilium/cilium/blob/5fc05ac07597ed651a8ccb2e59dc73691ec2caec/pkg/types/ipv4.go#L32

```
func (v4 *IPv4) DeepCopyInto(out *IPv4) {
        copy(out[:], v4[:])
        return
```

```
}
```

https://github.com/cilium/cilium/blob/5fc05ac07597ed651a8ccb2e59dc73691ec2caec/pkg/types/ipv6.go#L30

```
func (v6 *IPv6) DeepCopyInto(out *IPv6) {
        copy(out[:], v6[:])
        return
}
```

https://github.com/cilium/cilium/blob/314ca7baeff4f568ffc0bad95124e40665b1f88c/pkg/types/macaddr.go#L24

```
func (addr *MACAddr) DeepCopyInto(out *MACAddr) {
        copy(out[:], addr[:])
        return
}
```

11: Redundant imports

Overall severity:	Informational
ID:	ADA-CIL-11
Location:	Several packages
CWE:	• CWE-1041: Use of Redundant Code

Description

Cilium imports the same library twice in the same file these places of the source tree:

File	Double import
`daemon/cmd/daemon.go`	`github.com/cilium/cilium/pkg/k8s/client`
`daemon/cmd/daemon_main.go`	`github.com/cilium/cilium/pkg/wireguard/agent`
`operator/cmd/flags.go`	`github.com/cilium/cilium/pkg/option`
`operator/pkg/ciliumendpointslice/endpointslice.go`	`github.com/cilium/cilium/pkg/k8s/apis/cilium.io/v2alpha1`
`pkg/k8s/watchers/cilium_endpoint_slice.go`	`github.com/cilium/cilium/pkg/k8s/apis/cilium.io/v2alpha1`
`pkg/k8s/watchers/cilium_network_policy.go`	`github.com/cilium/cilium/pkg/k8s/utils`
`pkg/nodediscovery/nodediscovery.go`	`github.com/cilium/cilium/pkg/node/types`
`pkg/redirectpolicy/redirectpolicy.go`	`github.com/cilium/cilium/pkg/loadbalancer`
`pkg/service/service.go`	`github.com/cilium/cilium/pkg/datapath/types`

12: Redundant function parameters

Overall severity:	Informational
ID:	ADA-CIL-12
Location:	Several packages
CWE	• CWE-1041: Use of Redundant Code

Description
This issue lists the places in which Cilium passes a function parameter that remains unused in the function body rendering the parameter redundant.

Latest commit of this check: `b8a6791299083d9888819d03f458ecd1942abf81`

File	API	Param name
`api/v1/server/configure_cilium_api.go`	`configureServer`	`scheme, addr`
`daemon/cmd/ciliumendpoints.go`	`(*Daemon).deleteCiliumEndpoint`	`eps`
`daemon/cmd/daemon_main.go`	`(*Daemon).instantiateBGPControlPlane`	`ctx`
`daemon/cmd/hubble.go`	`getHubbleEventBufferCapacity`	`logger`
`operator/pkg/lbipam/lbipam.go`	`(*LBIPAM).poolOnUpsert`	`k`
`operator/pkg/lbipam/lbipam.go`	`(*LBIPAM).poolOnDelete`	`k`
`operator/pkg/lbipam/lbipam.go`	`(*LBIPAM).svcOnUpsert`	`k`
`operator/pkg/lbipam/lbipam.go`	`(*LBIPAM).svcOnDelete`	`k`
`pkg/alibabacloud/eni/node.go`	`(*Node).getSecurityGroupIDs`	`ctx`
`pkg/aws/eni/migration.go`	`(*InterfaceDB).fetchFromK8s`	`name`
`pkg/aws/eni/node.go`	`(*Node).getSecurityGroupIDs`	`ctx`
`pkg/bgp/manager/metallb.go`	`newMetalLBController`	`ctx`
`pkg/bgp/speaker/metallb.go`	`newMetalLBSpeaker`	`ctx`
`pkg/bgpv1/gobgp/routermgr.go`	`(*BGPRouterManager).withdraw`	`ctx`
`pkg/bgpv1/gobgp/workdiff.go`	`(*reconcileDiff).withdrawDiff`	`policy`
`pkg/bpf/map_linux.go`	`(*Map).deleteAllMapEvent`	`err`
`pkg/bpf/map_register_linux.go`	`unregisterMap`	`m`
`pkg/datapath/ipcache/listener.go`	`(*BPFListener).garbageCollect`	`ctx`
`pkg/datapath/iptables/iptables.go`	`(*IptablesManager).installMasqueradeRules`	`ifName`

pkg/datapath/linux/routing/migrate.go	(*migrator).copyRoutes	from
pkg/datapath/loader/loader.go	(*Loader).replaceNetworkDatapath	interfaces
pkg/egressgateway/net.go	addEgressIpRule	egressIP
pkg/ipam/allocator/podcidr/podcidr.go	getCIDRAllocatorsInfo	netTypes
pkg/ipam/node_manager.go	(*NodeManager).resyncNode	ctx
pkg/k8s/apis/cilium.io/client/register.go	waitForV1CRD	crdName
pkg/k8s/apis/cilium.io/client/v1beta1.go	waitForV1beta1CRD	crdName
pkg/k8s/identitybackend/identity.go	(*crdBackend).get	ctx
pkg/k8s/identitybackend/identity.go	(*crdBackend).getById	ctx
pkg/k8s/watchers/cilium_clusterwide_envoy_config.go	(*K8sWatcher).ciliumClusterwideEnvoyConfigInit	clientset
pkg/k8s/watchers/endpoint.go	(*K8sWatcher).updateK8sEndpointV1	oldEP
pkg/k8s/watchers/service.go	(*K8sWatcher).updateK8sServiceV1	oldSvc
pkg/k8s/watchers/watcher.go	(*K8sWatcher).enableK8sWatchers	ctx
pkg/k8s/watchers/watcher.go	(*K8sWatcher).delK8sSVCs	se
pkg/monitor/agent/agent.go	(*Agent).processPerfRecord	scopedLog
pkg/monitor/api/drop.go	extendedReason	reason
pkg/monitor/format/format.go	(*MonitorFormatter).policyVerdictEvents	prefix
pkg/monitor/format/format.go	(*MonitorFormatter).recorderCaptureEvents	prefix
pkg/monitor/format/format.go	(*MonitorFormatter).logRecordEvents	prefix
pkg/monitor/format/format.go	(*MonitorFormatter).agentEvents	prefix
pkg/node/address.go	chooseHostIPsToRestore	ipv6
pkg/pidfile/pidfile.go	kill	pidfile
pkg/policy/selectorcache.go	(*selectorManager).removeUser	dnsProxy
pkg/proxy/dns.go	(*dnsRedirect).setRules	wg
proxylib/proxylib/policymap.go	newPortNetworkPolicyRules	port

13: TOCTOU race condition in sockops `bpftoolLoad`

Overall severity:	Low
ID:	ADA-CIL-13
Location:	`pkg/sockops`
CWE:	• CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

Description

A TOCTOU race condition exists in sockops `bpftoolLoad`.

https://github.com/cilium/cilium/blob/473e75f4d49b6e47880833e041c946b5446a145d/pkg/sockops/sockops.go#L100

```
func bpftoolLoad(bpfObject string, bpfFsFile string) error {
	...
	maps, err := os.ReadDir(filepath.Join(bpf.GetMapRoot(), "/tc/globals/"))
	if err != nil {
		return err
	}

	for _, f := range maps {
		// Ignore all backing files
		if strings.HasPrefix(f.Name(), "..") {
			continue
		}

		use := func() bool {
			for _, n := range sockopsMaps {
				if f.Name() == n {
					return true
				}
			}
			return false
		}()

		if !use {
			continue
		}

		mapString := []string{"map", "name", f.Name(), "pinned",
filepath.Join(bpf.GetMapRoot(), "/tc/globals/", f.Name())}
		mapArgList = append(mapArgList, mapString...)
	}

	args := []string{"-m", "prog", "load", bpfObject, bpffs}
	args = append(args, mapArgList...)
	log.WithFields(logrus.Fields{
		"bpftool": prog,
		"args":    args,
	}).Debug("Load BPF Object:")
	out, err := exec.Command(prog, args...).CombinedOutput()
	if err != nil {
		return fmt.Errorf("Failed to load %s: %s: %s", bpfObject, err, out)
```

```
        }
        return nil
}
```

In this API there is a check whether a map should be used. If it should, then the map is referenced by name and passed to the bpftool. A race condition exists in that the map could be replaced after the check has occurred and before bpftool is invoked.

The specific part that has the race condition is this:

```
        use := func() bool {
                for _, n := range sockopsMaps {
                        if f.Name() == n {
                                return true
                        }
                }
                return false
        }()

        if !use {
                continue
        }

        mapString := []string{"map", "name", f.Name(), "pinned",
filepath.Join(bpf.GetMapRoot(), "/tc/globals/", f.Name())}
        mapArgList = append(mapArgList, mapString...)
```

The level of exploitability of this issue is low but is included here to highlight that there currently is not a guarantee that the map that Cilium loads is the map that it - or the user - expects to load.

14: Level of trust for input from cloud providers is too high

Overall severity:	Low
ID:	ADA-CIL-14
Location:	• `pkg/azure` • `pkg/alibabacloud` • `pkg/aws`
CWE:	• CWE-1041: Use of Redundant Code
Fix:	https://github.com/cilium/cilium/pull/22602

Description

When Cilium fetches metadata from a 3rd party, Azure IMS, Alibaba Cloud, AWS, the response is read entirely into memory without enforcing an upper limit. Since Cilium does not control the behavior of these 3rd-party APIs, it cannot ensure that the size of the response will always be within reasonable limits. If an attacker finds a way to generate a response that contains a large body, a Denial-of-Service scenario would exist, when Cilium reads the entire response into memory.

The scenario exists the following places:

Azure IMS

https://github.com/cilium/cilium/blob/314ca7baeff4f568ffc0bad95124e40665b1f88c/pkg/azure/api/metadata.go#L39

```
func getMetadataString(ctx context.Context, path string) (string, error) {
	client := &http.Client{
		Timeout: time.Second * 10,
	}
	url := fmt.Sprintf("%s/%s", metadataURL, path)
	req, err := http.NewRequestWithContext(ctx, http.MethodGet, url, nil)
	if err != nil {
		return "", nil
	}

	query := req.URL.Query()
	query.Add("api-version", metadataAPIVersion)
	query.Add("format", "text")

	req.URL.RawQuery = query.Encode()
	req.Header.Add("Metadata", "true")

	resp, err := client.Do(req)
	if err != nil {
		return "", err
	}
	defer func() {
```

```
                if err := resp.Body.Close(); err != nil {
                        log.WithError(err).Errorf("Failed to close body for request %s",
url)
                }
        }()

        respBytes, err := io.ReadAll(resp.Body)
        if err != nil {
                return "", err
        }

        return string(respBytes), nil
}
```

Alibaba Cloud

https://github.com/cilium/cilium/blob/181b030b0dd868091cc00d0bd8b1ce40688d63ae/pkg/alibabacloud/metadata/metadata.go#L50

```
func getMetadata(ctx context.Context, path string) (string, error) {
        client := &http.Client{
                Timeout: time.Second * 10,
        }
        url := fmt.Sprintf("%s/%s", metadataURL, path)
        req, err := http.NewRequestWithContext(ctx, http.MethodGet, url, nil)
        if err != nil {
                return "", err
        }

        resp, err := client.Do(req)
        if err != nil {
                return "", err
        }

        if resp.StatusCode != http.StatusOK {
                return "", fmt.Errorf("metadata service returned status code %d",
resp.StatusCode)
        }

        defer resp.Body.Close()
        respBytes, err := io.ReadAll(resp.Body)
        if err != nil {
                return "", err
        }

        return string(respBytes), nil
}
```

AWS

https://github.com/cilium/cilium/blob/c5cbf403dbe355ecbb80dfc8d7a8ed4da45c43bd/pkg/aws/metadata/metadata.go#L24

```
func getMetadata(client *imds.Client, path string) (string, error) {
        res, err := client.GetMetadata(context.TODO(), &imds.GetMetadataInput{
                Path: path,
```

```
        })
        if err != nil {
                return "", fmt.Errorf("unable to retrieve AWS metadata %s: %w", path, err)
        }

        defer res.Content.Close()
        value, err := io.ReadAll(res.Content)
        if err != nil {
                return "", fmt.Errorf("unable to read response content for AWS metadata
%q: %w", path, err)
        }

        return string(value), err
}
```

15: BGP configuration file is read entirely into memory

Overall severity	Low
ID:	ADA-CIL-15
Location:	`pkg/bgp`
CWE:	• CWE-1041: Use of Redundant Code
Fix:	https://github.com/cilium/cilium/pull/22602

Description

The BGP config parser reads a config file entirely into memory. This could create a scenario whereby a malicious user intentionally - or a genuine user unintentionally - could parse a config file that is larger than the available memory on the memory creating Denial-of-Service of the machine.

https://github.com/cilium/cilium/blob/314ca7baeff4f568ffc0bad95124e40665b1f88c/pkg/bgp/config/config.go#L16

```
func Parse(r io.Reader) (*metallbcfg.Config, error) {
	buf, err := io.ReadAll(r)
	if err != nil {
		return nil, fmt.Errorf("failed to read MetalLB config: %w", err)
	}
	config, err := metallbcfg.Parse(buf)
	if err != nil {
		return nil, fmt.Errorf("failed to parse MetalLB config: %w", err)
	}
	return config, nil
}
```

16: Race condition when starting operator apiserver

Overall severity:	Low
ID:	ADA-CIL-16
Location:	Operator apiserver
CWE:	• CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

Description

A race condition exists when starting the oprerator apiserver. `StartServer()` loops through all addresses in `s.listenAddres` and starts a goroutine in each iteration. Each goroutine refers to addr which is not passed to the goroutine. As such, the addr in each goroutine may not be the addr of the given loop iteration.

This is purely a cosmetic issue with the highest impact of causing confusion when going through the logs.

```
func (s *Server) StartServer() error {
	errs := make(chan error, 1)
	nServers := 0

	// Since we are opening this on localhost only, we need to make sure
	// we can open for both v4 and v6 localhost. In case the user is running
	// v4-only or v6-only.
	for _, addr := range s.listenAddrs {
		if addr == "" {
			continue
		}
		nServers++

		mux := http.NewServeMux()

		// Index handler is the the handler for Open-API router.
		mux.Handle("/", s.Server.GetHandler())
		// Create a custom handler for /healthz as an alias to /v1/healthz. A http
mux
		// is required for this because open-api spec does not allow multiple base
paths
		// to be specified.
		mux.HandleFunc("/healthz", func(rw http.ResponseWriter, _ *http.Request) {
			resp := s.healthzHandler.Handle(operator.GetHealthzParams{})
			resp.WriteResponse(rw, runtime.TextProducer())
		})

		srv := &http.Server{
			Addr:    addr,
			Handler: mux,
		}
```

```
		errCh := make(chan error, 1)

		lc := net.ListenConfig{Control: setsockoptReuseAddrAndPort}
		ln, err := lc.Listen(context.Background(), "tcp", addr)
		if err != nil {
			log.WithError(err).Fatalf("Unable to listen on %s for healthz
apiserver", addr)
		}

		go func() {
			err := srv.Serve(ln)
			if err != nil {
				// If the error is due to the server being shutdown, then
send nil to
				// the server errors channel.
				if errors.Is(err, http.ErrServerClosed) {
					log.WithField("address", addr).Debug("Operator API
server closed")
					errs <- nil
				} else {
					errCh <- err
					errs <- err
				}
			}
		}()

		go func() {
			select {
			case <-s.shutdownSignal:
				if err := srv.Shutdown(context.Background()); err != nil {
					log.WithError(err).Error("apiserver shutdown")
				}
			case err := <-errCh:
				log.WithError(err).Warn("Unable to start operator API
server")
			}
		}()

		log.Infof("Starting apiserver on address %s", addr)
	}

	var retErr error
	for err := range errs {
		if err != nil {
			retErr = err
		}

		nServers--
		if nServers == 0 {
			return retErr
		}
	}

	return nil
}
```

17: Bad code practice: Identical identifier of import and variable

Overall severity:	Low
ID:	ADA-CIL-17
Location:	`pkg/egressgateway`
CWE:	• CWE-1109: Use of Same Variable for Multiple Purposes

Description
Overwriting import identifiers could result in undefined behavior and should be avoided. In the following part of Cilium, a variable is assigned to an identifier that also refers to an imported package.

https://github.com/cilium/cilium/blob/710297f229480bbd4fc52f39ea68a6eeb333c9d4/pkg/egressgateway/manager.go#L80

```
func (manager *Manager) getIdentityLabels(securityIdentity uint32) (labels.Labels, error)
{
	identityCtx, cancel := context.WithTimeout(context.Background(),
option.Config.KVstoreConnectivityTimeout)
	defer cancel()
	if err := manager.identityAllocator.WaitForInitialGlobalIdentities(identityCtx);
err != nil {
		return nil, fmt.Errorf("failed to wait for initial global identities: %v",
err)
	}

	identity := manager.identityAllocator.LookupIdentityByID(identityCtx,
identity.NumericIdentity(securityIdentity))
	if identity == nil {
		return nil, fmt.Errorf("identity %d not found", securityIdentity)
	}
	return identity.Labels, nil
}
```

Where this package is imported: `github.com/cilium/cilium/pkg/identity`.

Recommendations
Change the variable identifier.

18: Deadlock from locked mutex

Overall severity:	Medium
ID:	ADA-CIL-18
Location:	`pkg/envoy`
CWE:	• CWE-667: Improper Locking
Fix:	https://github.com/cilium/cilium/pull/23077

Description

Go has two common tools when dealing with concurrency: Mutual exclusion - also known as mutex, and channels. A mutex is a low-level tool to protect against race conditions. A mutex can be either locked or unlocked. A mutex can perform two operations: Lock and unlock. Each operation is atomic, meaning that a process has to wait for a locked process to be unlocked until it itself can lock. If a process fails to unlock, other processes may wait for an unlock that does not happen resulting in a deadlock.

Cilium has a deadlock error in the envoy package from a missing mutex unlock before returning in case of an invalid `listenerConfig`:
https://github.com/cilium/cilium/blob/79c6f5725372b52c9877a9bde1249da039948649/pkg/envoy/server.go#L576

```
if err := listenerConfig.Validate(); err != nil {
                log.Errorf("Envoy: Could not validate Listener (%s): %s", err,
listenerConfig.String())
                return
        }
```

Recommendation

Unlock the `s.mutex` before returning.

19: Possible type confusions

Overall severity:	Low
ID:	ADA-CIL-19
Location:	Several packages
CWE:	• CWE-704: Incorrect Type Conversion or Cast • CWE-843: Access of Resource Using Incompatible Type ('Type Confusion')

Description

Type confusions occur when a variable is assumed to be of a type that it is not. They are usually more severe in memory-unsafe languages than in memory-safe languages like Go and are recoverable in Go which only rarely makes them critical. However, there have been previous cases of type confusions in open source Go source code having security implications such as GHSA-qq97-vm5h-rrhg. In addition, Cilium has had issues in the past with panics from type confusions: https://github.com/cilium/cilium/pull/171.

Ideally all type assertions should either be checked or a unit test should demonstrate that the given cast is safe. Checking all casts would ensure that all casts are safe, but since this may unnecessarily bloat the production code base this may not be the best avenue. Instead, a unit test could catch type confusions from being introduced from unchecked type assertions when the code base changes.

Below we list the type assertions identified during this audit.

https://github.com/cilium/cilium/blob/fd50b8d3b9684e0e88139e5776bd68ef15a344d0/cilium/cmd/bpf_ct_list.go#L122-L162

```
func dumpCt(maps []interface{}, args ...interface{}) {
	entries := make([]ctmap.CtMapRecord, 0)
	eID := args[0]

	for _, m := range maps {
		path, err := m.(ctmap.CtMap).Path()
		if err == nil {
			err = m.(ctmap.CtMap).Open()
		}
		if err != nil {
			if os.IsNotExist(err) {
				msg := "Unable to open %s: %s."
				if eID.(string) != "global" {
					msg = "Unable to open %s: %s: please try using
\"cilium bpf ct list global\"."
				}
				fmt.Fprintf(os.Stderr, msg+" Skipping.\n", path, err)
				continue
```

```
                        }
                        Fatalf("Unable to open %s: %s", path, err)
                }
                defer m.(ctmap.CtMap).Close()
                // Plain output prints immediately, JSON/YAML output holds until it
                // collected values from all maps to have one consistent object
                if command.OutputOption() {
                        callback := func(key bpf.MapKey, value bpf.MapValue) {
                                record := ctmap.CtMapRecord{Key: key.(ctmap.CtKey), Value:
*value.(*ctmap.CtEntry)}
                                entries = append(entries, record)
                        }
                        if err = m.(ctmap.CtMap).DumpWithCallback(callback); err != nil {
                                Fatalf("Error while collecting BPF map entries: %s", err)
                        }
                } else {
                        doDumpEntries(m.(ctmap.CtMap))
                }
        }
        if command.OutputOption() {
                if err := command.PrintOutput(entries); err != nil {
                        os.Exit(1)
                }
        }
}
```

https://github.com/cilium/cilium/blob/d5227f1baed6eb56865dc08275e6560bfff27cce/cilium/cmd/bpf_ipcache_get.go#L52

```
                v := value.([]string)
                if len(v) == 0 {
                        fmt.Printf("Unable to retrieve identity for LPM entry %s\n", arg)
                        os.Exit(1)
                }
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/cilium/cmd/bpf_lb_list.go#L71-L88

```
        parseBackendEntry := func(key bpf.MapKey, value bpf.MapValue) {
                id := key.(lbmap.BackendKey).GetID()
                backendMap[id] = value.DeepCopyMapValue().(lbmap.BackendValue).ToHost()
        }
        if err := lbmap.Backend4MapV3.DumpWithCallbackIfExists(parseBackendEntry); err !=
nil {
                Fatalf("Unable to dump IPv4 backends table: %s", err)
        }
        if err := lbmap.Backend6MapV3.DumpWithCallbackIfExists(parseBackendEntry); err !=
nil {
                Fatalf("Unable to dump IPv6 backends table: %s", err)
        }

        parseSVCEntry := func(key bpf.MapKey, value bpf.MapValue) {
                var entry string
```

```
                svcKey := key.(lbmap.ServiceKey)
                svcVal := value.(lbmap.ServiceValue).ToHost()
                svc := svcKey.String()
                svcKey = svcKey.ToHost()
```

https://github.com/cilium/cilium/blob/fd50b8d3b9684e0e88139e5776bd68ef15a344d0/cilium/cmd/bpf_nat_list.go#L60-L68

```
                if command.OutputOption() {
                        callback := func(key bpf.MapKey, value bpf.MapValue) {
                                record := nat.NatMapRecord{Key: key.(nat.NatKey), Value:
value.(nat.NatEntry)}
                                entries = append(entries, record)
                        }
                        if err = m.(nat.NatMap).DumpWithCallback(callback); err != nil {
                                Fatalf("Error while collecting BPF map entries: %s", err)
                        }
                } else {
                        out, err := m.(nat.NatMap).DumpEntries()
```

https://github.com/cilium/cilium/blob/e4d9dd21cfcb396bfe38893547bfe6ed578f7292/cilium/cmd/bpf_recorder_list.go#L57-L65

```
                if command.OutputOption() {
                        callback := func(key bpf.MapKey, value bpf.MapValue) {
                                record := recorder.MapRecord{Key:
key.(recorder.RecorderKey), Value: value.(recorder.RecorderEntry)}
                                entries = append(entries, record)
                        }
                        if err = m.DumpWithCallback(callback); err != nil {
                                Fatalf("Error while collecting BPF map entries: %s", err)
                        }
                } else {
```

https://github.com/cilium/cilium/blob/677750f8a3f098be7da6bb7b5a94c240e6633b36/operator/cmd/cilium_node.go#L328

```
        if _, ok := key.(ciliumNodeManagerQueueSyncedKey); ok {
                close(s.ciliumNodeManagerQueueSynced)
                return true
        }

        err := syncHandler(key.(string))
        if err == nil {
                // If err is nil we can forget it from the queue, if it is not nil
                // the queue handler will retry to process this key until it succeeds.
                queue.Forget(key)
                return true
        }
```

https://github.com/cilium/cilium/blob/c147bbd3211f0ef19eb13f4daab07e100e6b72d5/operator/pkg/ciliumendpointslice/endpointslice.go#L203

```
func syncCESsInLocalCache(cesStore cache.Store, manager operations) {
	for _, obj := range cesStore.List() {
		ces := obj.(*v2alpha1.CiliumEndpointSlice)
		// If CES is already cached locally, do nothing.
```

https://github.com/cilium/cilium/blob/c147bbd3211f0ef19eb13f4daab07e100e6b72d5/operator/pkg/ciliumendpointslice/endpointslice.go#L236

```
func (c *CiliumEndpointSliceController) processNextWorkItem() bool {
	cKey, quit := c.queue.Get()
	if quit {
		return false
	}
	defer c.queue.Done(cKey)

	err := c.syncCES(cKey.(string))
	c.handleErr(err, cKey)

	return true
}
```

https://github.com/cilium/cilium/blob/c147bbd3211f0ef19eb13f4daab07e100e6b72d5/operator/pkg/ciliumendpointslice/endpointslice.go#L278

```
	obj, exists, err := c.ciliumEndpointSliceStore.GetByKey(key)
	if err == nil && exists {
		ces := obj.(*v2alpha1.CiliumEndpointSlice)
		// Delete the CES, only if CEP count is zero in local copy of CES and
api-server copy of CES,
		// else Update the CES
		if len(ces.Endpoints) == 0 && c.Manager.getCEPCountInCES(key) == 0 {
			if err := c.reconciler.reconcileCESDelete(key); err != nil {
				return err
			}
		} else {
			if err := c.reconciler.reconcileCESUpdate(key); err != nil {
				return err
			}
		}
	}
```

https://github.com/cilium/cilium/blob/e3dfed84dc69990bbb68e9a181e984ec83dbe233/operator/pkg/gateway-api/controller.go#L191-L222

```
func onlyStatusChanged() predicate.Predicate {
	option := cmpopts.IgnoreFields(metav1.Condition{}, "LastTransitionTime")
	return predicate.Funcs{
		UpdateFunc: func(e event.UpdateEvent) bool {
			switch e.ObjectOld.(type) {
			case *gatewayv1beta1.GatewayClass:
				o, _ := e.ObjectOld.(*gatewayv1beta1.GatewayClass)
				n, ok := e.ObjectNew.(*gatewayv1beta1.GatewayClass)
				if !ok {
					return false
				}
```

```
                        return !cmp.Equal(o.Status, n.Status, option)
                case *gatewayv1beta1.Gateway:
                        o, _ := e.ObjectOld.(*gatewayv1beta1.Gateway)
                        n, ok := e.ObjectNew.(*gatewayv1beta1.Gateway)
                        if !ok {
                                return false
                        }
                        return !cmp.Equal(o.Status, n.Status, option)
                case *gatewayv1beta1.HTTPRoute:
                        o, _ := e.ObjectOld.(*gatewayv1beta1.HTTPRoute)
                        n, ok := e.ObjectNew.(*gatewayv1beta1.HTTPRoute)
                        if !ok {
                                return false
                        }
                        return !cmp.Equal(o.Status, n.Status, option)
                default:
                        return false
                }
        },
    }
}
```

https://github.com/cilium/cilium/blob/d0a43aa05ee39b884ff9e87318e2a6aebd5ca98f/operator/pkg/gateway-api/httproute.go#L44-L46

```
        func(rawObj client.Object) []string {
                hr, ok := rawObj.(*gatewayv1beta1.HTTPRoute)
                if !ok {
```

https://github.com/cilium/cilium/blob/d0a43aa05ee39b884ff9e87318e2a6aebd5ca98f/operator/pkg/gateway-api/httproute.go#L70-L72

```
        func(rawObj client.Object) []string {
                hr := rawObj.(*gatewayv1beta1.HTTPRoute)
                var gateways []string
```

https://github.com/cilium/cilium/blob/8df3d1e320da86b75f07f65adcd185299a5b6d9c/operator/pkg/lbipam/lbipam.go#L398-L401

```
        for i := 0; i < poolRetryQueue.Len(); i++ {
                retryInt, _ := poolRetryQueue.Get()
                retry := retryInt.(*retry)
```

https://github.com/cilium/cilium/blob/6c98f152ad9e9d9882bc02840474dd39c04bd1e0/operator/watchers/cilium_endpoint.go#L200-L204

```
        if !exists {
                return nil, false, nil
        }
        cep := item.(*cilium_api_v2.CiliumEndpoint)
        return cep, exists, nil
```

https://github.com/cilium/cilium/blob/af61d36f5f20a7f3b07b8430a400073eb20c411c/operator/watchers/node_taint.go#L71

```
        success := checkAndMarkNode(c, nodeGetter, key.(string), mno)
```

https://github.com/cilium/cilium/blob/af61d36f5f20a7f3b07b8430a400073eb20c411c/operator/watchers/node_taint.go#L161

```
        podInterface, exists, err := ciliumPodsStore.GetByKey(key.(string))
```

https://github.com/cilium/cilium/blob/af61d36f5f20a7f3b07b8430a400073eb20c411c/operator/watchers/node_taint.go#L170

```
        pod := podInterface.(*slim_corev1.Pod)
```

https://github.com/cilium/cilium/blob/af61d36f5f20a7f3b07b8430a400073eb20c411c/operator/watchers/node_taint.go#L198

```
                ciliumPod := ciliumPodInterface.(*slim_corev1.Pod)
```

https://github.com/cilium/cilium/blob/af61d36f5f20a7f3b07b8430a400073eb20c411c/operator/watchers/pod.go#L43

```
        pod := obj.(*slim_corev1.Pod)
```

https://github.com/cilium/cilium/blob/d5227f1baed6eb56865dc08275e6560bfff27cce/pkg/datapath/ipcache/listener.go#L192

```
                k := key.(*ipcacheMap.Key)
```

https://github.com/cilium/cilium/blob/e4d68f84a7c957bc9c6b6fea3bfda3151f830629/pkg/datapath/loader/hash.go#L77

```
        state, err := d.Hash.(encoding.BinaryMarshaler).MarshalBinary()
```

https://github.com/cilium/cilium/blob/e4d68f84a7c957bc9c6b6fea3bfda3151f830629/pkg/datapath/loader/hash.go#L82

```
        if err := newDatapathHash.(encoding.BinaryUnmarshaler).UnmarshalBinary(state);
err != nil {
```

https://github.com/cilium/cilium/blob/be6d746ae29fc733bc3a58e2afcc79e77c35485b/pkg/endpoint/policy.go#L607

```
                        regenResult := result.(*EndpointRegenerationResult)
```

https://github.com/cilium/cilium/blob/79c6f5725372b52c9877a9bde1249da039948649/pkg/envoy/server.go#L1778

```
                networkPolicy := res.(*cilium.NetworkPolicy)
```

https://github.com/cilium/cilium/blob/83f82482f9c831de712a14c9adc72caa0bda3dc3/pkg/envoy/xds/server.go#L274

```
                req := recv.Interface().(*envoy_service_discovery.DiscoveryRequest)
```

https://github.com/cilium/cilium/blob/83f82482f9c831de712a14c9adc72caa0bda3dc3/pkg/envoy/xds/server.go#L387

```
                resp := recv.Interface().(*VersionedResources)
```

https://github.com/cilium/cilium/blob/e6ce5c17afc21448fa9e69a7a36eb41814532256/pkg/hubble/relay/queue/priority_queue.go#L48

```
        resp := heap.Pop(&pq.h).(*observerpb.GetFlowsResponse)
```

https://github.com/cilium/cilium/blob/e6ce5c17afc21448fa9e69a7a36eb41814532256/pkg/hubble/relay/queue/priority_queue.go#L91

```
        resp := x.(*observerpb.GetFlowsResponse)
```

https://github.com/cilium/cilium/blob/69e4c6974891c161300889596f2029e489dded02/pkg/k8s/init.go#L102

```
                typesNode := nodeInterface.(*slim_corev1.Node)
```

https://github.com/cilium/cilium/blob/09f72f77b460bfcb0eef09c4552026ec2e45fe65/pkg/k8s/resource/resource.go#L254

```
                        entry := raw.(queueEntry)
```

https://github.com/cilium/cilium/blob/183726c8da72d899ef9c205bb7f80d1a2577c099/pkg/maps/ctmap/ctmap.go#L285

```
                key := k.(CtKey)
```

https://github.com/cilium/cilium/blob/183726c8da72d899ef9c205bb7f80d1a2577c099/pkg/maps/ctmap/ctmap.go#L289

```
                value := v.(*CtEntry)
```

https://github.com/cilium/cilium/blob/183726c8da72d899ef9c205bb7f80d1a2577c099/pkg/maps/ctmap/ctmap.go#L361

```
                entry := value.(*CtEntry)
```

https://github.com/cilium/cilium/blob/183726c8da72d899ef9c205bb7f80d1a2577c099/pkg/maps/ctmap/ctmap.go#L445

```
                entry := value.(*CtEntry)
```

https://github.com/cilium/cilium/blob/183726c8da72d899ef9c205bb7f80d1a2577c099/pkg/maps/ctmap/ctmap.go#L599-L600

```
                natKey := key.(nat.NatKey)
                natVal := value.(nat.NatEntry)
```

https://github.com/cilium/cilium/blob/183726c8da72d899ef9c205bb7f80d1a2577c099/pkg/maps/ctmap/utils.go#L37

```
                natKey := k.(*nat.NatKey6)
```

https://github.com/cilium/cilium/blob/183726c8da72d899ef9c205bb7f80d1a2577c099/pkg/maps/ctmap/utils.go#L60

```
                natVal := v.(*nat.NatEntry4)
```

https://github.com/cilium/cilium/blob/183726c8da72d899ef9c205bb7f80d1a2577c099/pkg/maps/ctmap/utils.go#L78-L79

```
                natKey := k.(*nat.NatKey6)
                natVal := v.(*nat.NatEntry6)
```

https://github.com/cilium/cilium/blob/183726c8da72d899ef9c205bb7f80d1a2577c099/pkg/maps/ctmap/utils.go#L118

```
                natKey := k.(*nat.NatKey6)
```

https://github.com/cilium/cilium/blob/58e4081c3c2a6e052d7c5e3443be33dd6ae5024b/pkg/maps/egressmap/policy.go#L176-L177

```
                        key := k.(*EgressPolicyKey4)
                        value := v.(*EgressPolicyVal4)
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/ipv4.go#L242

```
        vHost := v.ToHost().(*RevNat4Value)
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/ipv4.go#L288

```
        kHost := k.ToHost().(*Service4Key)
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/ipv4.go#L342

```
        sHost := s.ToHost().(*Service4Value)
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/ipv4.go#L452

```
        vHost := v.ToHost().(*Backend4Value)
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/ipv4.go#L516

```
        vHost := v.ToHost().(*Backend4ValueV3)
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/ipv6.go#L109

```
        vHost := v.ToHost().(*RevNat6Value)
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/ipv6.go#L153

```
        kHost := k.ToHost().(*Service6Key)
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/ipv6.go#L207

```
        sHost := s.ToHost().(*Service6Value)
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/ipv6.go#L316

```
        vHost := v.ToHost().(*Backend6Value)
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/lbmap.go#L83

```
        svcVal := svcKey.NewValue().(ServiceValue)
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/lbmap.go#L120

```
        zeroValue := svcKey.NewValue().(ServiceValue)
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/lbmap.go#L336

```
                matchKey := key.DeepCopyMapKey().(*AffinityMatchKey).ToHost()
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/lbmap.go#L357

```
                k := key.(SourceRangeKey).ToHost()
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/lbmap.go#L430-L438

```
        parseBackendEntries := func(key bpf.MapKey, value bpf.MapValue) {
                backendKey := key.(BackendKey)
                backendValue := value.DeepCopyMapValue().(BackendValue).ToHost()
                backendValueMap[backendKey.GetID()] = backendValue
        }

        parseSVCEntries := func(key bpf.MapKey, value bpf.MapValue) {
                svcKey := key.DeepCopyMapKey().(ServiceKey).ToHost()
                svcValue := value.DeepCopyMapValue().(ServiceValue).ToHost()
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/lbmap.go#L514-L515

```
                backendKey := key.(BackendKey)
                backendValue := value.DeepCopyMapValue().(BackendValue).ToHost()
```

https://github.com/cilium/cilium/blob/c6e53ea42bccbe8507e7df0f49319a2a853c054e/pkg/maps/lbmap/lbmap.go#L564

```
        zeroValue := fe.NewValue().(ServiceValue)
```

https://github.com/cilium/cilium/blob/d5227f1baed6eb56865dc08275e6560bfff27cce/pkg/maps/lbmap/source_range.go#L52

```
        kHost := k.ToHost().(*SourceRangeKey4)
```

https://github.com/cilium/cilium/blob/d5227f1baed6eb56865dc08275e6560bfff27cce/pkg/maps/lbmap/source_range.go#L96

```
        kHost := k.ToHost().(*SourceRangeKey6)
```

https://github.com/cilium/cilium/blob/314ca7baeff4f568ffc0bad95124e40665b1f88c/pkg/maps/metricsmap/metricsmap.go#L90-L91

```
                key := k.(*Key)
                values := v.(*Values)
```

https://github.com/cilium/cilium/blob/d5227f1baed6eb56865dc08275e6560bfff27cce/pkg/maps/nat/nat.go#L131-L138

```
        cb := func(k bpf.MapKey, v bpf.MapValue) {
                key := k.(NatKey)
                if !key.ToHost().Dump(&sb, false) {
                        return
                }
```

```
                val := v.(NatEntry)
                sb.WriteString(val.ToHost().Dump(key, nsecStart))
        }
```

https://github.com/cilium/cilium/blob/d1d8e7a35b35d3420a33251767ae2696d664da53/pkg/maps/policymap/policymap.go#L375-L381

```
        cb := func(key bpf.MapKey, value bpf.MapValue) {
                eDump := PolicyEntryDump{
                        Key:         *key.DeepCopyMapKey().(*PolicyKey),
                        PolicyEntry: *value.DeepCopyMapValue().(*PolicyEntry),
                }
                entries = append(entries, eDump)
        }
```

https://github.com/cilium/cilium/blob/314ca7baeff4f568ffc0bad95124e40665b1f88c/pkg/maps/recorder/recorder.go#L60-L65

```
        cb := func(k bpf.MapKey, v bpf.MapValue) {
                key := k.(RecorderKey)
                key.ToHost().Dump(&sb)
                val := v.(RecorderEntry)
                val.Dump(&sb)
        }
```

https://github.com/cilium/cilium/blob/58e4081c3c2a6e052d7c5e3443be33dd6ae5024b/pkg/maps/srv6map/policy.go#L158-L167

```
                func(k, v interface{}) {
                        key4 := k.(*PolicyKey4)
                        key := PolicyKey{
                                VRFID:    key4.VRFID,
                                DestCIDR: key4.getDestCIDR(),
                        }
                        value := v.(*PolicyValue)

                        cb(&key, value)
                })
```

https://github.com/cilium/cilium/blob/58e4081c3c2a6e052d7c5e3443be33dd6ae5024b/pkg/maps/srv6map/policy.go#L174-L183

```
                func(k, v interface{}) {
                        key6 := k.(*PolicyKey6)
                        key := PolicyKey{
                                VRFID:    key6.VRFID,
                                DestCIDR: key6.getDestCIDR(),
                        }
                        value := v.(*PolicyValue)

                        cb(&key, value)
                })
```

https://github.com/cilium/cilium/blob/9c07b75282546500e26da55c46b7cf9c110205f0/pkg/maps/srv6map/sid.go#L96-L97

```
                key := k.(*SIDKey)
                value := v.(*SIDValue)
```

https://github.com/cilium/cilium/blob/fe9baa95eadc9ab678af619063ba443a4abcafbd/pkg/maps/srv6map/state.go#L92-L103

```
        func(k, v interface{}) {
                key4 := k.(*StateKey4)
                srcIP := key4.InnerSrc.IP()
                dstIP := key4.InnerDst.IP()
                key := StateKey{
                        InnerSrc: &srcIP,
                        InnerDst: &dstIP,
                }
                value := v.(*StateValue)

                cb(&key, value)
        })
```

https://github.com/cilium/cilium/blob/fe9baa95eadc9ab678af619063ba443a4abcafbd/pkg/maps/srv6map/state.go#L110-L121

```
        func(k, v interface{}) {
                key6 := k.(*StateKey6)
                srcIP := key6.InnerSrc.IP()
                dstIP := key6.InnerDst.IP()
                key := StateKey{
                        InnerSrc: &srcIP,
                        InnerDst: &dstIP,
                }
                value := v.(*StateValue)

                cb(&key, value)
        })
```

https://github.com/cilium/cilium/blob/58e4081c3c2a6e052d7c5e3443be33dd6ae5024b/pkg/maps/srv6map/vrf.go#L158-L168

```
        func(k, v interface{}) {
                key4 := k.(*VRFKey4)
                srcIP := key4.SourceIP.IP()
                key := VRFKey{
                        SourceIP: &srcIP,
                        DestCIDR: key4.getDestCIDR(),
                }
                value := v.(*VRFValue)

                cb(&key, value)
        })
```

https://github.com/cilium/cilium/blob/58e4081c3c2a6e052d7c5e3443be33dd6ae5024b/pkg/maps/srv6map/vrf.go#L175-L185

```
		func(k, v interface{}) {
			key6 := k.(*VRFKey6)
			srcIP := key6.SourceIP.IP()
			key := VRFKey{
				SourceIP: &srcIP,
				DestCIDR: key6.getDestCIDR(),
			}
			value := v.(*VRFValue)

			cb(&key, value)
		})
```

https://github.com/cilium/cilium/blob/da9d6a0167e1cede54480dbe832ef0ef5dca3aa8/pkg/nodediscovery/nodediscovery.go#L456

```
	typesNode := nodeInterface.(*k8sTypes.Node)
```

20: Ill-defined contexts

Overall severity:	Informational
ID:	ADA-CIL-20
Location:	Several packages

Description
At the time of the audit, Cilium has 149 cases of ill-defined contexts - `context.TODO()`. To reproduce:

```
git clone https://github.com/cilium/cilium --depth=1
cd cilium
rm -r test
rm -r vendor
grep -r “context\.TODO” --exclude=*_test.go
```

The Golang docs states about `context.TODO()`:

“*TODO returns a non-nil, empty Context. Code should use context.TODO when it's unclear which Context to use or it is not yet available (because the surrounding function has not yet been extended to accept a Context parameter).*”

As such, ill-defined contexts should be avoided in Ciliums production code.

Recommendation
A best effort of using well-defined contexts over time.

21: Use of deprecated TLS version

Overall severity:	**Informational**
ID:	ADA-CIL-21
Location:	`pkg/crypto/certloader`
CWE	• CWE-326: Inadequate Encryption Strength • CWE-327: Use of a Broken or Risky Cryptographic Algorithm • CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')
Fix	https://github.com/cilium/cilium/commit/ca890a4938f765be78db19ac77916eff1be66a3f

Description

By default a `crypto/tls.Config` sets the minimum accepted TLS version to 1.0 when acting as a server. From the Golang docs on `crypto/tls.Config`:

"*// By default, TLS 1.2 is currently used as the minimum when acting as a*
// client, and TLS 1.0 when acting as a server. TLS 1.0 is the minimum
// supported by this package, both as a client and as a server."

This is an issue in case a minimum TLS version is not specified in the Config. TLS 1.0 and TLS 1.1 are formally deprecated and a number of known attacks are known to exploit weaknesses of the protocols. For example, TLS 1.0 and TLS 1.0 rely on MD5 and SHA-1 which makes the protocols vulnerable to downgrade attacks. Authentication of handshakes is done based on SHA-1 which increases the chance for an attacker to carry out a MITM attack by impersonating a server.

https://github.com/cilium/cilium/blob/314ca7baeff4f568ffc0bad95124e40665b1f88c/pkg/crypto/certloader/server.go#L102

```
func (c *WatchedServerConfig) ServerConfig(base *tls.Config) *tls.Config {
	// We return a tls.Config having only the GetConfigForClient member set.
	// When a client initialize a TLS handshake, this function will be called
	// and the tls.Config returned by GetConfigForClient will be used. This
	// mechanism allow us to reload the certificates transparently between two
	// clients connections without having to restart the server.
	// See also the discussion at https://github.com/golang/go/issues/16066.
	return &tls.Config{
		GetConfigForClient: func(_ *tls.ClientHelloInfo) (*tls.Config, error) {
			keypair, caCertPool := c.KeypairAndCACertPool()
			tlsConfig := base.Clone()
			tlsConfig.Certificates = []tls.Certificate{*keypair}
			if c.IsMutualTLS() {
				// We've been configured to serve mTLS, so setup the
ClientCAs
				// accordingly.
				tlsConfig.ClientCAs = caCertPool
```

```
                                // The caller may have its own desire about the handshake
                                // ClientAuthType. We honor it unless its tls.NoClientCert
(the
                                // default zero value) as we are configured to serve mTLS.
                                if tlsConfig.ClientAuth == tls.NoClientCert {
                                        tlsConfig.ClientAuth =
tls.RequireAndVerifyClientCert
                                }
                        }
                        c.log.WithField("keypair-sn", keypairId(keypair)).
                                Debugf("Server tls handshake")
                        return tlsConfig, nil
                },
        }
}
```

Recommendations

Specify the minimum accepted TLS version in the `crypto/tls.Config`.

The Cilium maintainers triaged this issue and found that the existing users of this library code ensured a newer TLS version was used. This included Hubble clients and servers. The library code itself did not previously enforce a new enough TLS version.

22: Deprecated function calls

Overall severity:	Low
ID:	ADA-CIL-22
Location:	Several packages
CWE:	• CWE-477: Use of Obsolete Function

Description
Deprecated 3rd-party APIs may be abandoned to a degree where patches to known security issues are not applied. Cilium makes calls to deprecated APIs in several places throughout the codebase.

We recommend creating a strategy to replace deprecated APIs over time. It could be shared in a public Github issue to get support from the community.

At commit `398cf5e051c49a46941d1efedf9659740d80f52c`, these are the uses of deprecated APIs that need addressing:

Location	Deprecated API call
`api/v1/health/server/server.go:316:3`	`httpsServer.TLSConfig.BuildNameToCertificate`
`api/v1/operator/server/server.go:317:3`	`httpsServer.TLSConfig.BuildNameToCertificate`
`api/v1/server/server.go:316:3`	`httpsServer.TLSConfig.BuildNameToCertificate`
`daemon/cmd/status.go:256:29`	`strings.Title`
`daemon/cmd/status.go:266:37`	`strings.Title`
`operator/metrics/metrics.go:139:24`	`prometheus.NewProcessCollector`
`pkg/aws/ec2/ec2.go:79:25`	`aws.EndpointResolverFunc`
`pkg/envoy/accesslog_server.go:167:14`	`pblog.Method`
`pkg/envoy/accesslog_server.go:168:18`	`pblog.Status`
`pkg/envoy/accesslog_server.go:169:23`	`pblog.Scheme`
`pkg/envoy/accesslog_server.go:169:37`	`pblog.Host`
`pkg/envoy/accesslog_server.go:169:49`	`pblog.Path`
`pkg/envoy/accesslog_server.go:170:26`	`pblog.HttpProtocol`
`pkg/envoy/accesslog_server.go:171:32`	`pblog.Headers`

pkg/envoy/sort.go:221:8	m1.GetExactMatch
pkg/envoy/sort.go:222:8	m2.GetExactMatch
pkg/envoy/sort.go:230:10	m1.GetSafeRegexMatch
pkg/envoy/sort.go:231:10	m2.GetSafeRegexMatch
pkg/envoy/sort.go:275:7	m1.GetPrefixMatch
pkg/envoy/sort.go:276:7	m2.GetPrefixMatch
pkg/envoy/sort.go:284:7	m1.GetSuffixMatch
pkg/envoy/sort.go:285:7	m2.GetSuffixMatch
pkg/health/client/client.go:59:3	tr.Dial
pkg/health/client/client.go:64:3	tr.Dial
pkg/hubble/metrics/drop/handler.go:56:62	flow.GetDropReason
pkg/hubble/parser/seven/parser.go:125:2	decoded.DropReason
pkg/hubble/parser/seven/parser.go:137:2	decoded.Reply
pkg/hubble/parser/seven/parser.go:144:2	decoded.Summary
pkg/hubble/parser/sock/parser.go:118:2	decoded.Summary
pkg/hubble/parser/threefour/parser.go:193:2	decoded.DropReason
pkg/hubble/parser/threefour/parser.go:194:41	decoded.DropReason
pkg/hubble/parser/threefour/parser.go:205:2	decoded.Reply
pkg/hubble/parser/threefour/parser.go:214:2	decoded.Summary
pkg/hubble/peer/types/client.go:56:45	grpc.WithInsecure
pkg/hubble/relay/pool/client.go:53:23	grpc.WithInsecure
pkg/hubble/relay/pool/option.go:29:4	grpc.WithInsecure
pkg/k8s/informer/informer.go:86:10	cache.NewDeltaFIFO
pkg/k8s/slim/k8s/apis/labels/selector.go:270:32	sets.String
pkg/k8s/slim/k8s/apis/labels/selector.go:271:9	sets.String
pkg/k8s/slim/k8s/apis/labels/selector.go:689:13	sets.String
pkg/k8s/slim/k8s/apis/labels/selector.go:755:33	sets.String
pkg/k8s/slim/k8s/apis/labels/selector.go:781:42	sets.String
pkg/k8s/slim/k8s/apis/labels/selector.go:817:37	sets.String
proxylib/npds/client.go:137:35	grpc.WithInsecure