ADA LOGICS

Cert-Manager Fuzzing Audit 2025

Cert-Manager Fuzzing Audit Report

In collaboration with the Cloud Native

Computing Foundation and the Cert-Manager Maintainers

Adam Korczynski, David Korczynski

14th February 2025

Contents

CNCF security and fuzzing audits

This report details the work Ada Logics did during our fuzzing audit of Cert-Manager. The audit was commissioned by the Cloud Native Computing Foundation (CNCF) and is part of the CNCFs efforts to secure the software in the CNCF landscape. Demonstrating and ensuring the security of these software packages is vital for the CNCF ecosystem, and the CNCF continues to use state-of-the-art techniques to secure its projects as well as carry out manual audits. Over the last few years, CNCF has been investing in security audits, fuzzing, and software supply chain security, which have helped proactively discover and fix hundreds of security issues and risks.

Fuzzing is a technique for testing for security and reliability issues in software. It has gained popularity due to its practical efficiency. The CNCFs efforts have resulted in more than twenty projects integrating fuzzing into their development pipeline through a series of dedicated fuzzing audits. In total, more than 350 bugs have been found through fuzzing of CNCF projects. The fuzzing efforts have focused on enabling continuous fuzzing to ensure continued security analysis, which is done by way of the open-source fuzzing project OSS-Fuzz.

CNCF continues work in this space. The focus for future work is integrating fuzzing into more projects, enabling sustainable fuzzer maintenance, increasing maintainer involvement and enabling fuzzing to find more vulnerabilities. Maintainers who are interested in getting fuzzing integrated into their projects or have questions about fuzzing are encouraged to visit the dedicated cncf-fuzzing repository https://github.com/cncf/cncf-fuzzing where questions and queries are welcome.

About Ada Logics

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

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

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

Ada Logics helps some of the most exposed organisations secure their software, analyse their code and increase security automation and assurance, and if you would like to consider working with us please reach out to us via our website. We write about our work on our blog. You can also follow Ada Logics on Linkedin, Twitter and Youtube.

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

Audit contacts

Contact	Role	Organisation	Email
Adam Korczynski	Auditor	Ada Logics Ltd	adam@adalogics.com
David Korczynski	Auditor	Ada Logics Ltd	david@adalogics.com
Ashley Davis	cert-manager maintainer	Venafi	
Tim Ramlot	cert-manager maintainer	Venafi	
Maël Valais	cert-manager maintainer	Venafi	
Richard Wall	cert-manager maintainer	Venafi	
Adam Talbot	cert-manager maintainer	Venafi	

Executive summary

In late 2024 and early 2025, Ada Logics worked on setting up a fuzzing suite for Cert-Manager. When this engagement started, Cert-Manager was already integrated into OSS-Fuzz from its previous security audit, where the auditors - also Ada Logics - set up continuous fuzzing with an initial set of fuzzers. The goal of the current fuzzing audit was to improve upon this initial integration and identify meaningful entrypoint to test by way of fuzzing. To do that, Ada Logics manually reviewed the code to identify areas where Cert-Manager processes untrusted input, and we identified the high-value targets by following Cert-Manager's threat model. The outcome was a set of fuzz tests for Cert-Managers controllers that have a wide call tree and process untrusted Kubernetes resources.

We carried out this work in collaboration with the Cert-Manager maintainers, which allowed us to efficiently get the fuzz tests checked into Cert-Managers source tree. This is the best way to maintain fuzzers in general, as the maintainers have the highest degree of control of the fuzzing suite.

We wrote 9 fuzz tests during the audit and integrated them all into Cert-Managers OSS-Fuzz integration so that they run continuously after the audit.

With the completion of this audit, Cert-Manager maintains its fuzzers in its own source tree and has a variety of fuzzers, from e2e tests to tests that target complex parsing and processing routines. All its fuzz tests run continuously on OSS-Fuzz, and Cert-Managers security team has demonstrated a healthy response to OSS-Fuzz findings by triaging and fixing crashes reported by OSS-Fuzz in a timely manner.

Cert-Manager fuzzing

In this section, we detail the Cert-Manager fuzzing suite. We first present a high-level view of the overall fuzzing architecture and how it supports running the fuzzers continuously. We then enumerate the fuzzers we wrote during the audit.

Architecture

A central component in Cert-Managers fuzzing infrastructure is its integration into OSS-Fuzz. The Cert-Manager source code and Cert-Managers fuzz tests are the two core elements that OSS-Fuzz uses to fuzz test Cert-Manager. The following diagram gives an overview of how OSS-Fuzz uses these two elements and what happens when an issue is found/fixed.

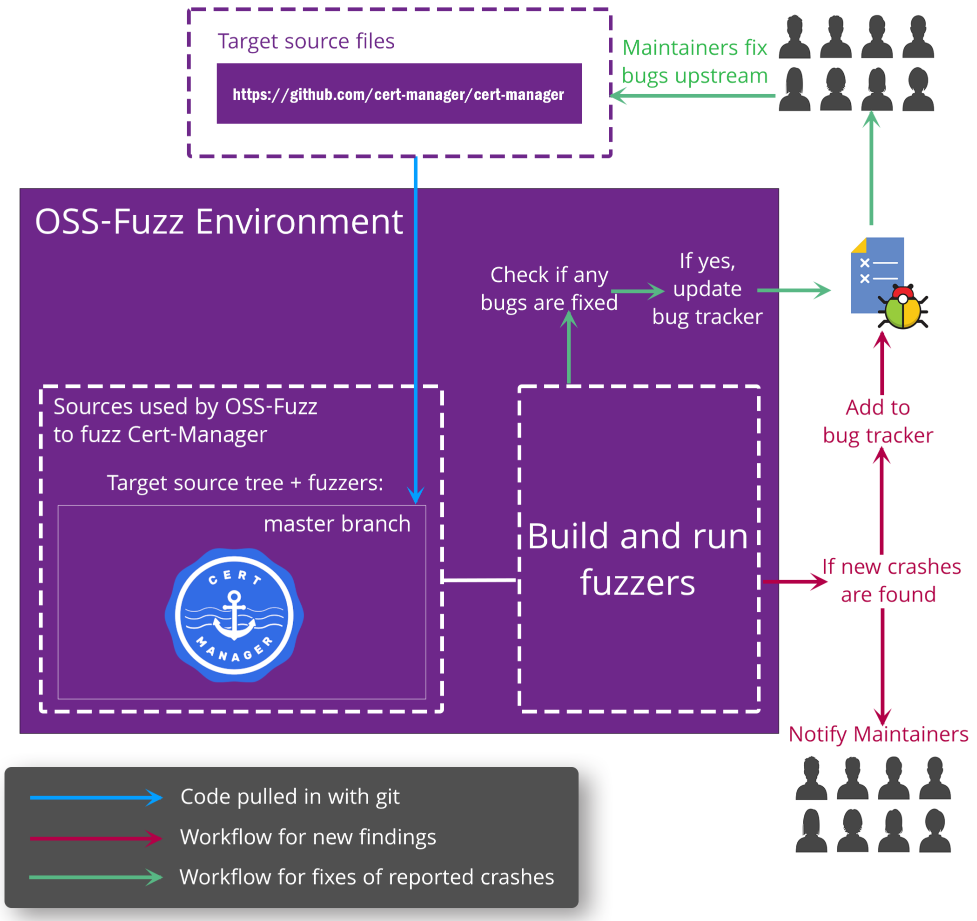

Target source files
https://github.com/cert-manager/cert-manager
Maintainers fix bugs upstream
OSS-Fuzz Environment
Check if any bugs are fixed
If yes, update bug tracker
Sources used by OSS-Fuzz to fuzz Cert-Manager
Target source tree + fuzzers:
master branch
CERT MANAGER
Build and run fuzzers
Add to bug tracker
If new crashes are found
Notify Maintainers
Code pulled in with git
Workflow for new findings
Workflow for fixes of reported crashes

Figure 1: Cert-Manager fuzzing architecture

The current OSS-Fuzz setup builds the fuzzers by cloning the upstream Cert-Manager GitHub repository to get the latest Cert-Manager source code and its fuzz tests. OSS-Fuzz then builds the fuzzers against the cloned Cert-Manager source code, which ensures that the fuzzers always run against the latest commit.

This build cycle happens daily, and OSS-Fuzz will verify whether any existing bugs have been fixed. If OSS-fuzz finds that any bugs have been fixed, OSS-Fuzz marks the crashes as fixed in the Monorail bug tracker and notifies maintainers.

At runtime, OSS-Fuzz uses its corpus accumulated from previous fuzz runs. If OSS-Fuzz detects any crashes when running the fuzzers, OSS-Fuzz performs the following actions:
1. A detailed crash report is created.
2. An issue in the Monorail bug tracker is created.
3. An email is sent to maintainers with links to the report and relevant entries in the bug tracker.

OSS-Fuzz has a 90-day disclosure policy, meaning that a bug becomes public in the bug tracker if it has not been fixed. The detailed report is never made public. The Cert-Manager maintainers will fix issues upstream, and OSS-Fuzz will pull the latest Cert-Manager master branch the next time it performs a fuzz run and verify that a given issue has been fixed.

Cert-Manager Fuzzers

In this section, we enumerate the Cert-Manager fuzz tests and the parts of Cert-Manager they test. We wrote a total of 9 fuzzers during the fuzzing audit. In this audit, we focused our efforts on writing fuzz tests for APIs with large call trees with an emphasis on targetting APIs that take in Kubernetes objects into Cert-Manager and then process them. Cert-Managers existing fuzzing setup, which Ada Logics made during its security audit, included fuzz tests for specific complex processing routines. One of these fuzzers recently found a security vulnerability while we were working on this fuzzing audit, and the setup had visibly been valuable to Cert-Manager. As such, with the completion of this fuzzing audit, Cert-Manager maintains a set of fuzz tests for specific complex APIs and a set of fuzz tests for the high-level APIs that reach those targeted APIs somewhere in their calltree. We believe this is a great set-up as the two types of fuzzers test Cert-Manager in different ways.

All the fuzzers we wrote during the audit are checked into Cert-Manager's upstream source tree. The list contains several fuzzers with duplicate names. The reason for this is that we have named the fuzzers according to the API that they test, and the fuzzers with duplicate names test APIs that are named similarly. For example, all the tests below are named `FuzzProcessItem`, and test APIs are named `ProcessItem()`.

Several of the fuzzers overapproximate their randomization of Kubernetes objects. That means that the objects that some of the fuzzers create and pass on to their target APIs would not be valid in a

Kubernetes cluster. For example, strings can include non-ASCII values. This overapproximates their test coverage, which we are happy with for three reasons. First, we need to make the fuzzers more complex to simulate real-world objects, which would increase the workload for maintaining these. The second reason is that we prefer to overapproximate than underapproximate, and by introducing more complexity, we would increase the risk of missing out on true positive issues if we either under-approximated during this audit or the Kubernetes ruleset for allowed characters in objects changed over time, and the Cert-Manager community did not update the fuzzers to reflect those changes. The third reason is that the level of overapproximation the fuzzers have will result in finding bugs that cannot be triggered in a real-world scenario. In contrast, they would not report issues that are false positives. In our opinion, this is a good approach, and we argue that these types of crashes should be fixed as they can cause further issues at a later time if they remain unfixed. As a side note, if the fuzzers, for some reason, should report excessive noise from specific fields in specific objects, the fuzzers can easily be adjusted over time to accommodate such a scenario. During the audit, we added the fuzzers to Cert-Managers OSS-Fuzz integration as soon as they were ready to be merged upstream, and after weeks of continuous running, we have not seen such noise.

#	Name	Cert-Manager Package
1	FuzzValidate_approval	`github.com/cert-manager/cert-manager/internal/webhook/admission/certificaterequest/approval`
2	FuzzProcessItem_issuing	`github.com/cert-manager/cert-manager/pkg/controller/certificates/issuing`
3	FuzzProcessItem_keymanager	`github.com/cert-manager/cert-manager/pkg/controller/certificates/keymanager`
4	FuzzProcessItem_readiness	`github.com/cert-manager/cert-manager/pkg/controller/certificates/readiness`
5	FuzzProcessItem_requestmanager	`github.com/cert-manager/cert-manager/pkg/controller/certificates/requestmanager`

#	Name	Cert-Manager Package
6	FuzzProcessItem_revisionmanager	`github.com/cert-manager/cert-manager/pkg/controller/certificates/revisionmanager`
7	FuzzProcessItem_trigger	`github.com/cert-manager/cert-manager/pkg/controller/certificates/trigger`
8	FuzzVaultCRController	`github.com/cert-manager/cert-manager/pkg/controller/certificaterequests/vault`
9	FuzzVenafiCRController	`github.com/cert-manager/cert-manager/pkg/controller/certificaterequests/venafi`

1: FuzzValidate_approval

This fuzzer tests the validation for approval plugins. The fuzzer creates a random `AdmissionRequest` and two random `CertificateRequest`, creates an approval plugin and invokes the approval plugins `Validate()` method with the three randomized objects.

2: FuzzProcessItem_issuing

This fuzzer tests the `certificates-issuing` controller in https://github.com/cert-manager/cert-manager/tree/master/pkg/controller/certificates/issuing. The fuzzer creates a random certificate and secret and adds them to a mocked cluster. Both of these objects can be invalid. The fuzzer has the option to add an issuing certificate which it can either randomize entirely or use a valid one. In both cases - whether to include an issuing certificate and whether to randomize it entirely - the fuzzer chooses which action to take based on its testcase.

Finally, the fuzzer invokes the controllers `ProcessItem()` method.

3: FuzzProcessItem_keymanager

This fuzzer tests the `certificates-key-manager` controller in https://github.com/cert-manager/cert-manager/tree/master/pkg/controller/certificates/keymanager. The fuzzer creates a random certificate, secret and a set of certificate requests. All of these objects are randomized entirely and can be invalid in a production-level cluster. The fuzzer then adds these projects to the mocked cluster and invokes the controllers `ProcessItem()` method.

4: FuzzProcessItem_readiness

This fuzzer tests the `certificates-readiness` controller in https://github.com/cert-manager/cert-manager/tree/master/pkg/controller/certificates/readiness. The fuzzer creates a random certificate and secret which it randomizes entirely. The fuzzer has the option to randomize the x509 bytes in the secret or use valid ones. The fuzzer then adds the two objects to the mocked cluster and invokes the controller's `ProcessItem()` method.

5: FuzzProcessItem_requestmanager

This fuzzer tests the `certificates-request-manager` controller in https://github.com/cert-manager/cert-manager/tree/master/pkg/controller/certificates/requestmanager. The fuzzer first creates a set of random certificate requests. Next, it creates a certificate which can be either entirely random or the fuzzer can choose to generate a valid certificate. This allows the fuzzer to easier test certain code paths in the target's calltree that would be blocked by an invalid certificate. At the same time, the fuzzer also tests for edge cases related to an invalid certificate. The fuzzer also creates a secret that it can randomize entirely, or it can create a valid one. The fuzzer then adds the objects to the mocked cluster and invokes the controller's `ProcessItem()` method.

6: FuzzProcessItem_revisionmanager

This fuzzer tests the `certificates-revision-manager` controller in https://github.com/cert-manager/cert-manager/tree/master/pkg/controller/certificates/revisionmanager. The fuzzer creates a set of randomized certificate requests and a set of random certificates. Both resources can be entirely randomized and will overapproximate the range of issues the test for. The fuzzer then adds the objects to the mocked cluster and invokes the controller's `ProcessItem()` method.

7: FuzzProcessItem_trigger

This fuzzer tests the `certificates-revision-manager` controller in https://github.com/cert-manager/cert-manager/tree/master/pkg/controller/certificates/trigger. The fuzzers adds a set of randomized certificates and a set of secrets to the mocked cluster and invokes the controllers `ProcessItem()` method. All objects can be fully randomized and can overapproximate the range of issues they test for.

8: FuzzVaultCRController

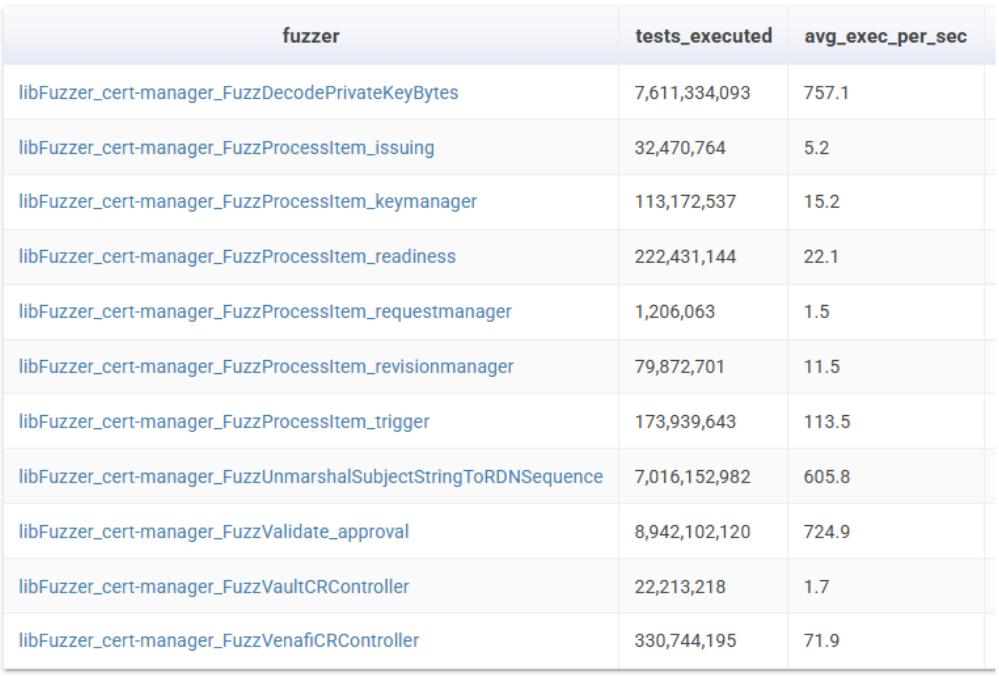
This fuzzer tests Cert-Managers Vault issuer resource by creating a randomized vault issuer and processing it by way of the certificate request controller in https://github.com/cert-manager/cert-manager/tree/master/pkg/controller/certificaterequests. The fuzzer tests this by first creating a certificate request with the Vault base issuer. The fuzzer then either creates and adds a secret and an issuer to the mocked cluster or adds a Vault issuer object. Once the fuzzer has added the created and randomized objects to the mocked cluster, it invokes the controller's `Sync()` method.

9: FuzzVenafiCRController

This fuzzer tests Cert-Managers Venafi issuer resource by creating a randomized vault issuer and processing it by way of the certificate request controller in https://github.com/cert-manager/cert-manager/tree/master/pkg/controller/certificaterequests. The fuzzer tests this by first creating a certificate request with the Venafi base issuer. The fuzzer then either creates and adds a secret and an issuer to the mocked cluster or adds a Venafi issuer object. Once the fuzzer has added the created and randomized objects to the mocked cluster, it invokes the controller's `Sync()` method.

Runtime stats

During the audit - between 1st November 2024 and 12th February 2025 - OSS-Fuzz has run the fuzzers with an accumulated number of executions over 20 billion. Some of these fuzzers are faster than others due to the infrastructure setup required in the slower fuzzers. Nonetheless, OSS-Fuzz's ample compute has supported running Cert-Managers fuzzers tens and hundreds of millions of times over the three months of Cert-Managers fuzzing audit:



fuzzer	tests_executed	avg_exec_per_sec
libFuzzer_cert-manager_FuzzDecodePrivateKeyBytes	7,611,334,093	757.1
libFuzzer_cert-manager_FuzzProcessItem_issuing	32,470,764	5.2
libFuzzer_cert-manager_FuzzProcessItem_keymanager	113,172,537	15.2
libFuzzer_cert-manager_FuzzProcessItem_readiness	222,431,144	22.1
libFuzzer_cert-manager_FuzzProcessItem_requestmanager	1,206,063	1.5
libFuzzer_cert-manager_FuzzProcessItem_revisionmanager	79,872,701	11.5
libFuzzer_cert-manager_FuzzProcessItem_trigger	173,939,643	113.5
libFuzzer_cert-manager_FuzzUnmarshalSubjectStringToRDNSequence	7,016,152,982	605.8
libFuzzer_cert-manager_FuzzValidate_approval	8,942,102,120	724.9
libFuzzer_cert-manager_FuzzVaultCRController	22,213,218	1.7
libFuzzer_cert-manager_FuzzVenafiCRController	330,744,195	71.9

Figure 2: Cert-Manager fuzzers runtime stats 1st Noveber 2024 to 12th February 2025

A reason for the heavy setup that leads to slower execution time is that we primarily wrote fuzz tests for controllers during this audit. These fuzzers are e2e tests where we set up the controller in a manner that is close to a production use case. We then make the controller process Kubernetes resources. As a result, if the fuzzers find a crash, the triager can review the crash from a context of e2e testing and how the crash impacts a near-production use case.

A good fuzzing suite will have a combinatin of e2e fuzz tests and tests for specific, processing-heavy APIs and methods. Where possible, we recommend dedicating specific fuzz tests to complex parsing and processing routines without setting up infrastructure such as controllers. Cert-Managers fuzzing suite indeed implements this approach with its e2e fuzzers for controllers complimented by three specific fuzz tests for complex targets: 1) `FuzzDecodePrivateKeyBytes`, 2) `FuzzUnmarshalSubjectStringToRDNSequence`, 3) `FuzzValidate_approval`.

Bugs and crashes

We described Cert-Managers OSS-Fuzz's workflow. Continuous fuzzing comes with many advantages one of which is that OSS-Fuzz will fuzz test the current state of Cert-Managers code base. As such, OSS-Fuzz may find crashes at any time, and it is important that Cert-Manager monitors alerts from OSS-Fuzz.

During this fuzzing audit, the fuzzers did not find any crashes, and since Cert-Manager integrated into OSS-Fuzz, its fuzzers have found just a few crashes. As such, in Cert-Managers cases, continuous monitoring of OSS-Fuzz notifications is important to catch bugs and potential security problems - when they rarely happen.

We can attest to the fact that the Cert-Manager community does indeed monitor OSS-Fuzz alerts. One of the fuzz tests we wrote for Cert-Manager during its security audit in late 2023 found in 2024 a security vulnerability in Cert-Managers parsing routines for PEM data that could allow an attacer to cause denial of service of a cluster running Cert-Manager (GHSA-r4pg-vg54-wxx4). The Cert-Manager team independently assessed the vulnerability from OSS-Fuzz reports and released Cert-Manager v1.16.2, v1.15.4 and v1.12.14 with a fix.

Cert-Manager has added several security contacts to the list of notifyees as well as Cert-Managers security mailing list, `cert-manager-security@googlegroups.com`. OSS-Fuzz manages its contact lists by way of the `project.yaml` file of integrated projects. Cert-Managers can be found here: https://github.com/google/oss-fuzz/blob/ae3cf711b8180d0e34d9954df77327f0fe6f3edf/projects/cert-manager/project.yaml.

Strategic recommendations

With the completion of this audit, Cert-Manager has a healthy fuzzing set-up which follows industry best practices. Below, we share a list of strategic recommendations for Cert-Manager to maintain this setup.

React to build breakages

We recommend that Cert-Manager monitors the OSS-Fuzz build for breakages which are likely to occur when or if Cert-Manager changes its source tree either structurally or through refactoring. Cert-Manager fuzzers should remain functional so that they keep testing Cert-Manager over time since some code parts might be reachable after a long time of running.

Cert-Manager has not seen many build breakages from upstream changes since it was integrated into OSS-Fuzz. As such, we expect the effort to be low moving forward.

Monitor OSS-Fuzz reports

OSS-Fuzz reports crashes to the email addresses in its `project.yaml` file (https://github.com/google/oss-fuzz/blob/ae3cf711b8180d0e34d9954df77327f0fe6f3edf/projects/cert-manager/project.yaml). The fuzzers can find crashes after months of runtime, and as such, we recommend keeping an eye out for reports from OSS-Fuzz.

Consider CI integration

OSS-Fuzz has a CI module called CI-Fuzz. This can build and run Cert-Managers fuzzers in its CI and test code contributions before they are merged into the source tree. Fuzzing in the CI can add processing time to Cert-Managers CI pipeline, and because of that we have not added it as part of this audit. Nonetheless, we recommend that the Cert-Manager community considers whether it fits the projects' workflow.

Explore new areas and vectors

Over time, as Cert-Manager receives more feedback from users, or if the project changes with new code parts, it may need more fuzz tests to cover either such new code parts or new vectors that might be exposed to untrusted data. When and if this happens, we recommend the Cert-Manager community to be mindful of adding new fuzz tests in the same manner as we have done in this audit: By checking them into Cert-Managers own source tree and adding them to Cert-Managers OSS-Fuzz build script.