PRESENTS

Fuzzing audit of containerd

In collaboration with the containerd project maintainers and The Linux Foundation

Authors

Adam Korczynski <adam@adalogics.com>
David Korczynski <david@adalogics.com>
Date: 2nd March, 2023

CNCF security and fuzzing audits

This report details a fuzzing audit commissioned by the CNCF and the engagement is part of the broader efforts carried out by CNCF in securing 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 carrying out manual audits. Over the last handful of years, CNCF has been investing in security audits, fuzzing and software supply chain security that has helped proactively discover and fix hundreds of issues.

Fuzzing is a proven technique for finding security and reliability issues in software and the efforts so far have enabled fuzzing integration into more than twenty CNCF projects through a series of dedicated fuzzing audits. In total, more than 350 bugs have been found through fuzzing of CNCF projects. The fuzzing efforts of CNCF have focused on enabling continuous fuzzing of projects to ensure continued security analysis, which is done by way of the open source fuzzing project OSS-Fuzz[1].

CNCF continues work in this space and will further increase investment to improve security across its projects and community. 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 in memory safe languages. 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.

[1] https://github.com/google/oss-fuzz

Executive Summary

This report outlines the work that has been done to integrate fuzzing into the containerd project. The work was funded by the Cloud Native Computing Foundation and was carried out by Ada Logics throughout 2021 and 2022.

Results summarised
Extensive fuzzing integration efforts spanning over more than a year resulted in 28 fuzzers targeting containerd code.
All fuzzers are running continuously on OSS-Fuzz and in the CI.
4 crashes found, 1 CVE: • 1 panic from invalid digests • 1 type confusion • 1 slice bounds out of range • 1 memory exhaustion vulnerability (CVE-2023-25153)
All fuzzers are merged upstream into the containerd repository.

The high-level goal of the engagement was to integrate fuzzing into containerd in a continuous manner. To meet this goal, Ada Logics executed the following sub-goals:

1. New fuzzers: A significant number of fuzzers, totalling 28 fuzzers, were written to cover different parts of the code base. The fuzzers range in nature from focus on complex processing routines to whole-package fuzz harnesses.
2. Continuous fuzzing by way of OSS-Fuzz: All fuzzers were added to the open source fuzzing service OSS-Fuzz. This enables the fuzzers to run continuously and provides additional features e.g. issue reporting, issue tracking and enables the fuzzers to operate on the latest version of containerd.
3. CI integration: All OSS-Fuzz projects can add CIFuzz[2] to their CI pipeline. This tests all pull requests with fuzzers that cover the code being changed. Ada Logics integrated CIFuzz for containerd.

All fuzzing harnesses in this engagement were integrated into OSS-Fuzz. To accommodate rapid development and prevent excessive upstream review cycles, new fuzzers were added to the cncf-fuzzing repository[3], and OSS-Fuzz was instructed to pull the fuzzers from there at build time. containerd maintainer Kazuyoshi Kato migrated the fuzzers upstream to the containerd repository and instructed OSS-Fuzz to use them from there.

[2] https://google.github.io/oss-fuzz/getting-started/continuous-integration/
[3] https://github.com/cncf/cncf-fuzzing

The audit comprised a significant fuzzing effort resulting in three issues found. The three issues reported is a low number, and this demonstrates containerd's high level of code quality. This is even more impressive considering the size of the fuzzing effort which spanned more than a year.

Table of contents

Project Summary

Ada Logics auditors

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

containerd maintainers involved in the audit

Name	Title	Email
Kazuyoshi Kato	Senior Software Development Engineer	katokazu@amazon.com
Samuel Karp	Staff Software Engineer	samuelkarp@google.com

Assets

Url	Branch
https://github.com/containerd/containerd	`main`

containerd fuzzing

In this section we present details on containerd's fuzzing set up, and in particular the overall fuzzing architecture as well as the specific fuzzers developed.

Architecture

A central component in containerd's fuzzing suite is continuous fuzzing by way of OSS-Fuzz. containerd upstream source tree is the key software package that OSS-Fuzz uses to fuzz containerd. The following figure gives an overview of how OSS-Fuzz uses containerd's upstream repository and what happens when an issue is found/fixed.

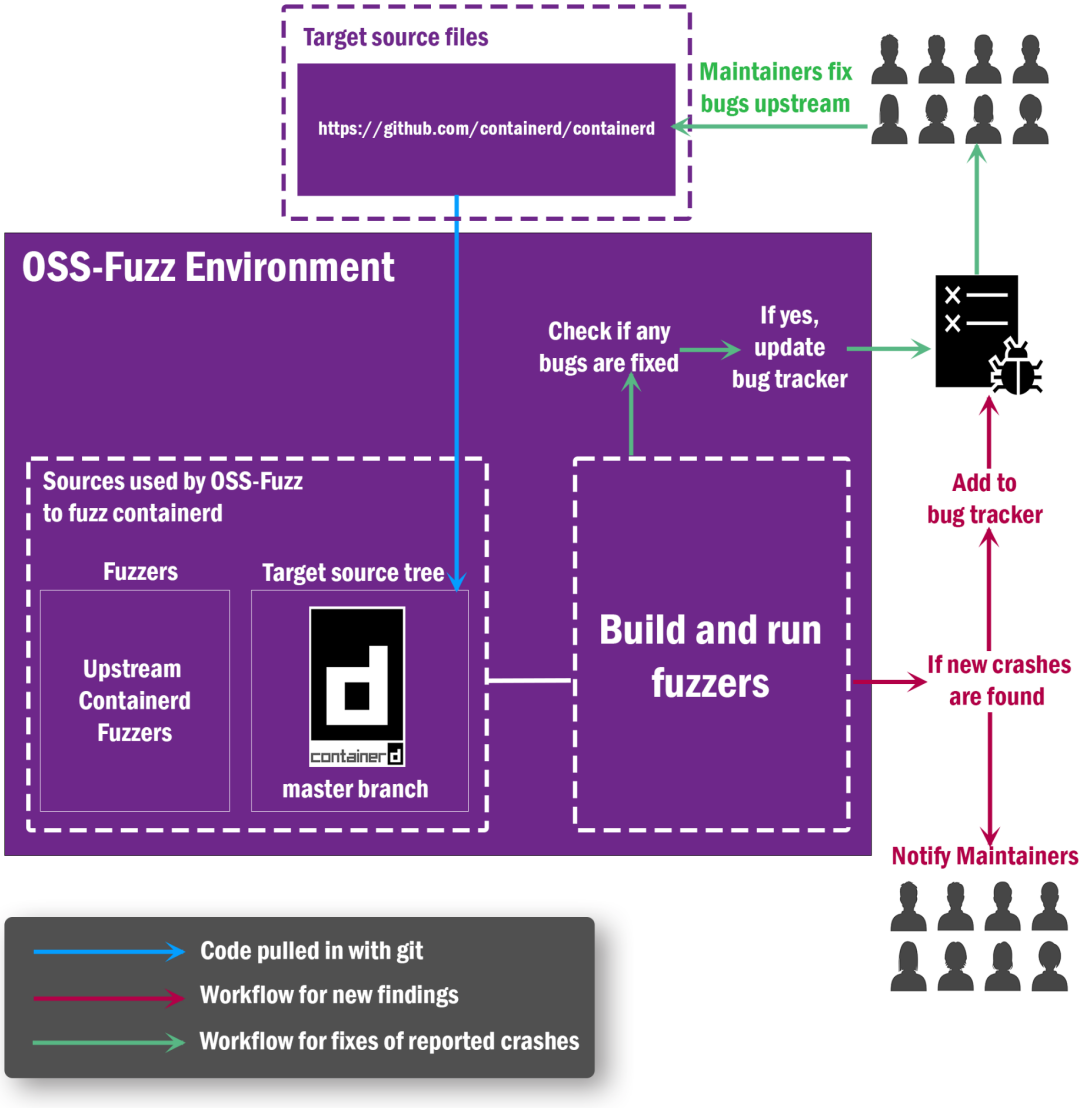

Figure 0.1: containerd's fuzzing architecture

The current OSS-Fuzz setup builds the fuzzers by cloning the upstream containerd Github repository to get the latest containerd source code and then builds the fuzzers in the cloned containerd source tree. As such, the fuzzers are always run against the latest containerd commit.

This build cycle happens daily and OSS-Fuzz will verify if 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.

In each fuzzing iteration, 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 bug tracker is created.
3. An email is sent to maintainers with links to the report and relevant entry 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 containerd maintainers will fix issues upstream, and OSS-Fuzz will pull the latest containerd master branch the next time it performs a fuzz run and verify that a given issue has been fixed.

containerd's fuzzers

In this section we will give an outline of the fuzzers written for containerd. At the time of conclusion of the fuzzing audit, all fuzzers reside in the containerd repository. They have been implemented by either the standard library Go fuzzing engine or the go-fuzz fuzzing engine, both of which are supported for continuous testing by OSS-Fuzz.

The following fuzzers have been written for Containerd during the fuzzing audit.

#	Name	Target package
1	`FuzzLoadDefaultProfile`	`containerd/contrib/apparmor`
2	`FuzzApply`	`containerd/archive`
3	`FuzzImportIndex`	`containerd/images/archive`
4	`FuzzCSWalk`	`containerd/content/local`
5	`FuzzArchiveExport`	`containerd/images/archive`
6	`FuzzUUIDParse`	`github.com/google/uuid`
7	`FuzzExchange`	`containerd/events/exchange`
8	`FuzzImageStore`	`containerd/images`
9	`FuzzLeaseManager`	`containerd/leases`
10	`FuzzContainerStore`	`containerd/containers`

11	`FuzzContentStore`	`containerd/content`
12	`FuzzCRISandboxServer`	`containerd/pkg/cri/sbserver`
13	`FuzzImagesCheck`	`containerd/images`
14	`FuzzDiffApply`	`containerd/diff/apply`
15	`FuzzDiffCompare`	`containerd/diff/walking`
16	`FuzzCRIServer`	`containerd/pkg/cri/server`
17	`FuzzParseAuth`	`containerd/pkg/cri/server`
18	`FuzzContainerdImport`	`containerd`
19	`FuzzParseProcPIDStatus`	`containerd/pkg/cap`
20	`FuzzDecompressStream`	`containerd/archive/compression`
21	`FuzzPlatformsParse`	`containerd/platforms`
22	`FuzzContentStoreWriter`	`containerd/content/local`
23	`FuzzFetcher`	`containerd/remotes/docker`
24	`FuzzParseDockerRef`	`containerd/remotes/docker`
25	`FuzzParseHostsFile`	`containerd/remotes/docker/config`
26	`FuzzParseAuthHeader`	`containerd/remotes/docker/auth`
27	`FuzzConvertManifest`	`containerd/remotes`
28	`FuzzFiltersParse`	`containerd/filters`

Target APIs

FuzzLoadDefaultProfile

Creates a file, writes pseudo-random bytes to the file and passes the filename to `LoadDefaultProfile()`.

FuzzApply

Applies a pseudo-random tar archive on a directory.

FuzzImportIndex

Instantiates a content store, creates pseudo-random tar bytes and imports the tar bytes by way of `ImportIndex()`.

FuzzCSWalk

Creates a pseudo-random blob store, walks it and verifies each entry.

FuzzArchiveExport

Creates a pseudo-random blob store and exports it.

FuzzUUIDParse
Tests the 3rd-party dependency `github.com/google/uuid`.

FuzzExchange
Instantiates an exchange and calls `Publish()` and `Forward()` with pseudo-randomized events.

FuzzImageStore
Instantiates an image store and calls its `Create()`, `List()`, `Update()`, `Delete()` methods in pseudo-random order with pseudo-randomized parameters for each method.

FuzzLeaseManager
Instantiates a lease manager and calls its `Create()`, `List()`, `AddResource()`, `Delete()`, `DeleteResource()`, `ListResources()` methods in pseudo-random order with pseudo-randomized parameters for each method.

FuzzContainerStore
Instantiates a container store and calls its `Create()`, `List()`, `Delete()`, `Update()`, `Get()` methods in pseudo-random order with pseudo-randomized parameters for each method.

FuzzContentStore
Instantiates a content store and calls its `Info()`, `Update()`, `Walk()`, `Delete()`, `ListStatuses()`, `Status()`, `Abort()`, `Commit()` methods in pseudo-random order with pseudo-randomized parameters for each method.

FuzzCRISandboxServer
Creates a sandbox CRI server and in pseudo-random order calls its `CreateContainer()`, `RemoveContainer()`, `AddSandbox()`, `ListContainers()`, `StartContainer()`, `ContainerStats()`, `ListContainerStats()`, `ContainerStatus()`, `StopContainer()`, `UpdateContainerResources()`, `ListImages()`, `RemoveImages()`, `ImageStatus()`, `ImageFsInfo()`, `ListPodSandbox()`, `PortForward()`, `RemovePodSandbox()`, `RunPodSandbox()`, `PodSandboxStatus()`, `StopPodSandbox()`, `Status()`, `UpdateRuntimeConfig()` methods.

FuzzImagesCheck
Creates a content store and checks it with a pseudo-randomized image descriptor.

FuzzDiffApply
Creates a pseudo-random oci descriptor and a pseudo-random number of mounts and applies them to a file system applier.

FuzzDiffCompare

Creates two different slices of pseudo-random mounts and compares them.

FuzzCRIServer
Creates a CRI server and in pseudo-random order calls its `CreateContainer()`, `RemoveContainer()`, `AddSandbox()`, `ListContainers()`, `StartContainer()`, `ContainerStats()`, `ListContainerStats()`, `ContainerStatus()`, `StopContainer()`, `UpdateContainerResources()`, `ListImages()`, `RemoveImages()`, `ImageStatus()`, `ImageFsInfo()`, `ListPodSandbox()`, `PortForward()`, `RemovePodSandbox()`, `RunPodSandbox()`, `PodSandboxStatus()`, `StopPodSandbox()`, `Status()`, `UpdateRuntimeConfig()` methods.

FuzzParseAuth
Passes a pseudo-random auth config and host string to `ParseAuth()`.

FuzzContainerdImport
Creates a client and imports pseudo-random tar bytes.

FuzzParseProcPIDStatus
Passes pseudo-random bytes to ParseProcPIDStatus().

FuzzDecompressStream
Passes a pseudo-random byte slice to DecompressStream().

FuzzPlatformsParse
Passes a pseudo-random string to platforms.Parse().

FuzzContentStoreWriter
Writes pseudo-random bytes to a content store.

FuzzFetcher
Fetches pseudo-random bytes from a demo registry.

FuzzParseDockerRef
Parses a pseudo-random docker reference.

FuzzParseHostsFile
Tests parseHostsFile() with a pseudo-random host file.

FuzzParseAuthHeader
Tests `ParseAuthHeader()` with a http header containing a pseudo-random `WWW-Authenticate` string.

FuzzConvertManifest

Tests `ConvertManifest()` with a pseudo-random oci descriptor.

FuzzFiltersParse

Parses a pseudo-random filter string.

Runtime stats

The power of continuous fuzzing by way of OSS-Fuzz means that a lot of compute power is used to analyse the code. There are significant benefits to this as fuzzing relies on genetic mutational algorithms that over time increasingly builds up a corpus of test cases. This means, the time spent fuzzing is an important factor when determining the completeness of a fuzzing audit. As of 6th December 2022, the total runtime stats in the form of tests executed and hours run is:

Name	Total tests executed	Total runtime (hours)
FuzzLoadDefaultProfile	22,570,694,284	1,169.1
FuzzApply	829,056,289	1,987.6
FuzzImportIndex	164,873,077	1,944.8
FuzzCSWalk	60,715,597	1,869.9
FuzzArchiveExport	278,209,239	1,956.5
FuzzUUIDParse	31,089,700,668	1,761
FuzzExchange	307,395,090	963
FuzzImageStore	522,581,063	1,991.8
FuzzLeaseManager	238,822,003	2,005
FuzzContainerStore	347,279,095	1,990.6
FuzzContentStore	276,186,129	1,826.5
FuzzCRISandboxServer	179,829	1,054.1
FuzzImagesCheck	27,365,017	60.2
FuzzDiffApply	20,314,511	47.1
FuzzDiffCompare	105,514,110	876
FuzzCRIServer	174,508	1,105.4
FuzzParseAuth	19,691,079,012	2,022.1
FuzzContainerdImport	263,979	1,530.4
FuzzParseProcPIDStatus	59,210,853,905	2,028.8
FuzzDecompressStream	21,788,080	72.2
FuzzPlatformsParse	39,424,788,221	2,240.7
FuzzContentStoreWriter	27,813,273	626
FuzzFetcher	5,673,591,790	2,031.2
FuzzParseDockerRef	2,170,086,760	2,039.2
FuzzParseHostsFile	11,085,831	70.5
FuzzParseAuthHeader	34,822,726,125	1,492.6

FuzzConvertManifest	60,628,025	63.2
FuzzFiltersParse	23,637,619,834	1,567.2

Issues found

The fuzzers have found 4 unique crashes; 3 in containerd and 1 in a third-party dependency. All three issues have been fixed upstream.

#	ID	Title	Fixed
1	ADA-CONT-1	digest.Algorithm() and digest.Encoded() may panic for invalid digests	Yes
2	ADA-CONT-2	interface conversion: interface {} is *errors.errorString, not string	Yes
3	ADA-CONT-3	Slice bounds out of range in mount.(*Mount).Mount	Yes
4	ADA-CONT-4	OCI image importer memory exhaustion	Yes

All issues were reported by OSS-Fuzz by way of its bug tracker which the containerd maintainers have access to. Below we cover each issue found with a link to each bug report in the bug tracker. These reports contain an overview of the issue, however, reproducer test cases and stacktraces are not publicly available and this includes even after the bugs are marked as fixed.

The most notable issue is ADA-CONT-4: “OCI image importer memory exhaustion”. This issue could be exploited to exhaust memory on the host machine if a victim would import a particularly well-crafted OCI image. containerd have assigned this issue CVE-2023-25153 with severity moderate and released a fix for it in containerd 1.6.18: https://github.com/containerd/containerd/releases/tag/v1.6.18.

Issue 1: `digest.Algorithm()` and `digest.Encoded()` may panic for invalid digests

OSS-Fuzz bug tracker:	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=50776&q=containerd
Fixed:	Yes
ID:	ADA-CONT-1

Description

A fuzzer demonstrated that lack of validation was in place for digests before calling either `Digest.Algorithm()` and `Digest.Encoded()`. The result was a panic from a 3rd-party library. The issue affected several calls in `images/archive`, and it was fixed by validating the digest before invoking any of the methods that could cause the panic.

The underlying issue of this crash is that Digest.Encoded() and Digest.Algorithm() do not validate the digest but rather assumes that it has been validated. This is also mentioned in the documentation:

https://github.com/containerd/containerd/blob/3e7bbb8a491840ddc099749ea2af30c1796557f7/vendor/github.com/opencontainers/go-digest/digest.go#L127

```
134  // Encoded returns the encoded portion of the digest. This will panic if the
135  // underlying digest is not in a valid format.
136  func (d Digest) Encoded() string {
137          return string(d[d.sepIndex()+1:])
138  }
```

Containerd was lacking this validation in several places across the code base and was added here: https://github.com/containerd/containerd/pull/7488

Issue 2: interface conversion: interface {} is *errors.errorString, not string

OSS-Fuzz bug tracker:	• https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=48057 • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=50781 • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=51697 • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=53356 • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=53768
Fixed	Yes
ID:	ADA-CONT-2

Description

A type confusion was found in the 3rd-party library `github.com/pelletier/go-toml@v1.9.3`. The call in Containerd to the 3rd-party library is made in https://github.com/containerd/containerd/blob/main/remotes/docker/config/hosts.go:

```
327  func parseHostsFile(baseDir string, b []byte) ([]hostConfig, error) {
328          tree, err := toml.LoadBytes(b)
329          if err != nil {
330                  return nil, fmt.Errorf("failed to parse TOML: %w", err)
331          }
332          ...
```

Figure 2.2: Point of failure of ADA-CONT-2

The issue was fixed upstream in https://github.com/pelletier/go-toml/commit/fed1464066413075eac02cd4dc368b5221845541

Issue 3: Slice bounds out of range in mount.(*Mount).Mount

OSS-Fuzz bug tracker:	• https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=50829 • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=52682
Fixed:	Yes
ID:	ADA-CONT-3

Description

A crash was found in `github.com/containerd/containerd/mount.compactLowerdirOption()` where a fuzzer could trigger an out of range by passing a well-crafted string in the slice in the function argument. The entrypoint of the fuzzers to `compactLowerdirOption()` was `(m *Mount) Mount(target string)` which passes `m.Options` to `compactLowerdirOption()`.

The stacktrace demonstrates how the fuzzer triggered the crash:

```
0  panic: runtime error: slice bounds out of range [2:1]
1  goroutine 17 [running, locked to thread]:
2  github.com/containerd/containerd/mount.compactLowerdirOption({0x10c000936000?, 0x24, 0x24})
3      github.com/containerd/containerd/mount/mount_linux.go:283 +0x96e
4  github.com/containerd/containerd/mount.(*Mount).Mount(0x10c000a5daa0, {0x10c000a079c0, 0x1e})
5      github.com/containerd/containerd/mount/mount_linux.go:61 +0x2f4
6  github.com/containerd/containerd/mount.All(...)
7      github.com/containerd/containerd/mount/mount.go:35
8  github.com/containerd/containerd/mount.WithTempMount({0x311b918, 0x10c0000481c0},
9  {0x10c000934a80, 0x7, 0x6ae745?}, 0x10c000a5db88)
10     github.com/containerd/containerd/mount/temp.go:61 +0x685
11 github.com/containerd/containerd/diff/walking.(*walkingDiff).Compare(0x10c000084de8, {0x311b918,
12 0x10c0000481c0}, {0x10c000934a80, 0x7, 0x8}, {0x10c000084d18, 0x0, 0x0}, {0x0, ...})
13     github.com/containerd/containerd/diff/walking/differ.go:89 +0x82f
14 github.com/containerd/containerd/contrib/fuzz.FuzzDiffCompare({0x620000001080, 0xf05, 0xf05})
15     github.com/containerd/containerd/contrib/fuzz/diff_fuzzer.go:106 +0xa9c
```

Figure 3.1: Stacktrace for ADA-CONT-3

Issue 4: OCI image importer memory exhaustion

OSS-Fuzz bug tracker:	• https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=55038 • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=55084
Fixed:	Yes
ID:	ADA-CONT-4

Description

This finding has been assigned CVE-2023-25153.
GH advisory: https://github.com/containerd/containerd/security/advisories/GHSA-259w-8hf6-59c2

A fuzzer found an issue in the image importing logic of containerd, whereby an attacker could craft a malicious OCI image and cause system-wide resource exhaustion when the victim would import the image. The particular issue had its root cause in the way some of the files were processed in the image; if some of the files in the image were large, denial-of-service would be achieved by way of memory exhaustion.

The issue was reported by OSS-fuzz on January 12th 2023. For further information about impact, we refer to the Github advisory linked above, and the release notes for containerd 1.6.18: https://github.com/containerd/containerd/releases/tag/v1.6.18.

Conclusions and future work

In this audit, Ada Logics improved containerd's fuzzing efforts. The efforts resulted in 28 fuzzers, 3 issues found and a CI integration. The containerd team moved the fuzzers upstream and now maintain a well-organised fuzzing suite that runs continuously on OSS-Fuzz and on pull-requests in the CI.

The current state of fuzzing containerd is impressive and the containerd maintainers deserve a lot of credit for this. Three issues reported is a low number, and this demonstrates containerd's high level of code quality. This is even more impressive considering the size of the fuzzing effort which spanned more than a year.

Fuzzing is a continuous discipline, and we recommend that containerd takes the following steps moving forward:

Maintaining the fuzzers
The fuzzers should keep running without breaking to keep testing the code for hard-to-find bugs and build up the corpus.
Over time, containerd's fuzzing suite will improve passively from new features added to the fuzzing infrastructure of OSS-Fuzz. An exciting development in this context novel bug detectors targeted managed languages. These will be added in a non-intrusive way, meaning that as long as containerd's fuzzers run continuously, they will over time test for more classes of bugs.

Improve coverage
We recommend making it a continuous effort to identify missing test coverage of the fuzzers. This can be done using the code coverage visualisations provided by OSS-Fuzz.

Require fuzzers for new code
We recommend that new code contributions are required to be accompanied by fuzz tests. This will both ensure that new code gets tested from the moment it gets merged into containerd, and it will ensure that the fuzzer is written by the developer itself.

Acknowledgements

We thank the Linux Foundation for sponsoring this project as well as the team at containerd for a fruitful and enjoyable collaboration.

We also thank the maintainers and team of OSS-Fuzz for their efforts in making open source fuzzing possible in this manner.