PRESENTS

Knative Fuzzing Audit

In collaboration with the Knative project maintainers and The Linux Foundation

Authors

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

In this engagement, Ada Logics worked on improving Knatives fuzzing suite. At the time of this engagement, Knative was not integrated into OSS-Fuzz, and the goal of this fuzzing audit was to first integrate Knative into OSS-Fuzz and then build upon this integration and improve the fuzzing efforts in a continuous manner.

The fuzzing audit added fuzzers for complex data processing APIs as well as roundtrip tests for the Knative custom resource types.

Most development of the fuzzers was carried out in the CNCF-Fuzzing repository, https://github.com/cncf/cncf-fuzzing/tree/main/projects/knative. This allowed the auditors to make small iterations of the fuzzers throughout the audit and avoid imposing the overhead of having the Knative maintainers review trivial changes to the fuzzers. OSS-Fuzz was instructed to pull the fuzzers from CNCF-Fuzzing in addition to the fuzzers from Knatives repositories.

The fuzzers found a single crash during the audit and continue to test the Knative code base after the audit has concluded.

Results summarised
29 fuzzers developed
All fuzzers added to Knatives OSS-Fuzz integration
1 crash found in a 3rd-party dependency

Table of Contents

CNCF security and fuzzing audits **2**
Executive summary **3**
Table of Contents **4**
Project Summary **5**
Knative fuzzing **6**
Issues found by fuzzers **19**

Project Summary

Ada Logics auditors

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

Knative maintainers involved in the audit

Name	Title	Email
Evan Anderson	Knative Maintainer	evan.k.anderson@gmail.com

Assets

Url	Branch
https://github.com/knative/eventing	`main`
https://github.com/knative/serving	`main`
https://github.com/knative/pkg	`main`

Knative fuzzing

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

Architecture

A central component in Knatives approach to fuzzing is continuous fuzzing by way of OSS-Fuzz. The Knative source code and the source code for the Knative fuzzers are the two key software packages that OSS-Fuzz uses to fuzz Knative. The following figure gives an overview of how OSS-Fuzz uses these two packages and what happens when an issue is found/fixed.

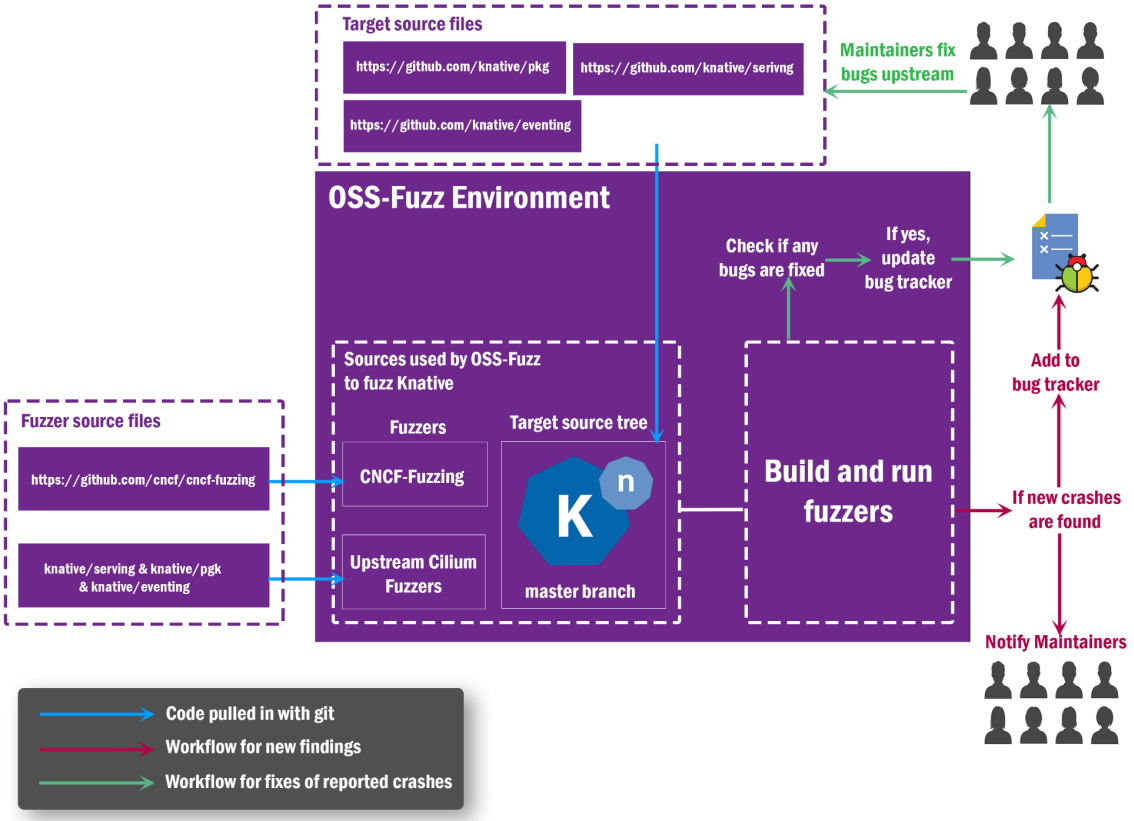

Figure 1.1: Knatives fuzzing architecture

The current OSS-Fuzz set up builds the fuzzers by cloning the upstream Knative Github repositories to get the latest Knative source code and the CNCF-Fuzzing Github repository to get the latest set of fuzzers, and then builds the fuzzers against the cloned Knative code. As such, the fuzzers are always run against the latest Knative commit. OSS-Fuzz pulls in three different repositories from Knatives github profile: `/pkg`, `/serving` and `/eventing`.

The reason for pulling in three repositories is that Knative organises its project in such a manner, where each main component is located in its own Github repository.

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 Monorail 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 Knative maintainers will fix issues upstream, and OSS-Fuzz will pull the latest Knative master branch the next time it performs a fuzz run and verify that a given issue has been fixed.

Knative Fuzzers

In this section we present a highlight of the Knative fuzzers and which parts of Knative they test. In total, 29 fuzzers were written during the fuzzing audit.
From a high level, the fuzzers written in this audit can be categorised into two groups:

1. New fuzzers
2. Optimization of the existing serialisation roundtrip fuzzers.

The new fuzzers were written for selected, particularly complex processing routines across the Knative projects. They were written as std lib golang harnesses. The optimization of the existing serialisation roundtrip fuzzers was carried out for the tests of Knatives custom resources. The roundtrip test is implemented here: https://github.com/knative/pkg/blob/44d1d7d97889b3fafe8d88e85f2d298c0d009aa1/apis/testing/roundtrip/roundtrip.go#L91, and is by and large Knatives own implementation of Kubernetes' roundtrip test: https://github.com/kubernetes/kubernetes/blob/3ffdfbe286ebcea5d75617da6accaf67f815e0cf/staging/src/k8s.io/apimachinery/pkg/api/apitesting/roundtrip/roundtrip.go#L51. In the early stages during the audit, Ada Logics did an assessment of the roundtrip fuzzers and found that these were slow, so we rewrote the `ExternalTypesViaJSON()` test which is used to test resource types from `knative.dev/pkg`, `knative.dev/pkg` and

`knative.dev/eventing`. We did this for a single fuzzer and let it run to see the benefits over time and were able to improve their performance by more than 300%. At the end of the audit, on the 26th February 2023, the optimized roundtrip test was 3,69 times faster than the gofuzz-based roundtrip test:

Type	Name	Tests executed	Avg. execs/second
Original	FuzzMessagingRoundTripTypesToJSON	53,203,898	141.8
Improved	FuzzMessagingRoundTripTypesToJSON Experimental	194,726,485	521.5

Having tested this out, Ada Logics proceeded to rewrite all Knatives roundtrip fuzzers to use the improved roundtrip test.

At the end of Knatives fuzzing audit, all existing roundtrip tests have been rewritten to the improved and faster version.

Fuzzers Overview

Here we enumerate the fuzzers that we wrote during this audit. We added the fuzzers to Knatives OSS-Fuzz integration ad-hoc which has allowed them to run with excessive CPU power during the audit itself. For details about the runtime stats, see the section “Runtime stats” later in this report. The fuzzers continue to run after the audit has completed, and will keep testing for hard-to-find bugs.
Below we first list all the fuzzers we wrote during the audit, and below we go into the details of each fuzzer.

#	Name	Package
1	`FuzzJsonDecode`	`knative.dev/pkg/webhook/json`
2	`FuzzAdmit`	`knative.dev/pkg/webhook/configmaps`
3	`FuzzNewObservabilityConfigFromConfigMap`	`knative.dev/pkg/metrics`
4	`FuzzChildName`	`knative.dev/pkg/kmeta`
5	`FuzzSendRawMessage`	`knative.dev/pkg/websocket`
6	`FuzzNewRevisionThrottler`	`knative.dev/serving/pkg/activator/net`
7	`FuzzRouteReconciler`	`knative.dev/serving/pkg/reconciler/route`
8	`FuzzDomainNameFromTemplate`	`knative.dev/serving/pkg/reconciler/route/domains`

9	FuzzValidation	knative.dev/serving/pkg/apis/serving/v1
10	FuzzMessagingRoundTripTypesToJSON	knative.dev/eventing/pkg/apis/messaging/v1
11	FuzzMessagingRoundTripTypesToJSONExperimental	knative.dev/eventing/pkg/apis/messaging/v1
12	FuzzSourcesRoundTripTypesToJSONExperimental	knative.dev/eventing/pkg/apis/sources/v1
13	FuzzRetryablehttpFromRequest	knative.dev/eventing/pkg/kncloudevents
14	FuzzFilters	knative.dev/eventing/pkg/broker/filter
15	FuzzDurableConnection	knative.dev/pkg/websocket
16	FuzzReceiveMessage	knative.dev/pkg/websocket
17	FuzzDuckV1beta1RoundTripTypesToJSONExperimental	knative.dev/pkg/apis/duck/v1beta1
18	FuzzDuckV1RoundTripTypesToJSONExperimental	knative.dev/pkg/apis/duck/v1
19	FuzzServingV1RoundTripTypesToJSONExperimental	knative.dev/serving/pkg/apis/serving/v1
20	FuzzFlowsRoundTripTypesToJSONExperimental	knative.dev/eventing/pkg/apis/flows/v1
21	FuzzEventingRoundTripTypesToJSONExperimental	knative.dev/eventing/pkg/apis/eventing/v1
22	FuzzAvailabilityNodePriorityScore	knative.dev/eventing/pkg/scheduler/plugins/core/availabilitynodepriority
23	FuzzAvailabilityZonePriorityScore	knative.dev/eventing/pkg/scheduler/plugins/core/availabilityzonepriority
24	FuzzEvenPodSpreadFilter	knative.dev/eventing/pkg/scheduler/plugins/core/evenpodspread
25	FuzzEvenPodSpreadScore	knative.dev/eventing/pkg/scheduler/plugins/core/evenpodspread
26	FuzzRemoveWithAvailabilityNodePriorityScore	knative.dev/eventing/pkg/scheduler/plugins/core/removewithavailabilitynodepriority
27	FuzzRemoveWithAvailabilityZonePriorityScore	knative.dev/eventing/pkg/scheduler/plugins/core/removewithavailabilityzonepriority
28	FuzzRemoveWithEvenPodSpreadPriorityScore	knative.dev/eventing/pkg/scheduler/plugins/core/removewithevenpodspreadpriority
29	FuzzValidateCESQLExpression	knative.dev/eventing/pkg/pkg/apis/eventing/v1

Fuzzer Descriptions

1: FuzzJsonDecode

Tests `knative.dev/pkg/webhook/json.Decode()`. The fuzzer passes a pseudo-randomized byte slice and a test interface.

2: FuzzAdmit

`FuzzAdmit` targets the webhook AdmissionControllers `Admit()` method: https://github.com/knative/pkg/blob/35bcd16656b5475bc628a52183a41ccd35902c99/webhook/configmaps/configmaps.go. It creates a pseudo-random request and passes it to `Admit()`. The fuzzer provides an insight into whether any requests can be made to the `Admit()` API and crash the validation routine.

3: FuzzNewObservabilityConfigFromConfigMap

Tests the API that creates an `ObservabilityConfig` from a `ConfigMap`. The fuzzer first creates a pseudo-randomized `ConfigMap` and passes it to `knative.dev/pkg/metrics.NewObservabilityConfigFromConfigMap()` to see if any improper handling of the provided `ConfigMap` can crash Knative. This calls into Knatives parsing routine for `ConfigMaps` which is implemented here https://github.com/knative/pkg/blob/3c4dec9b9f0eb95cef428ac875c2b9425ff9ac60/configmap/parse.go#L239:

https://github.com/knative/pkg/blob/6ce976ce9255359fab375aacf31e6fe628722c57/metrics/config_observability.go#L111

```
func NewObservabilityConfigFromConfigMap(configMap *corev1.ConfigMap)
(*ObservabilityConfig, error) {
	oc := defaultConfig()
	if configMap == nil {
		return oc, nil
	}

	if err := cm.Parse(configMap.Data,
		cm.AsBool("logging.enable-var-log-collection",
&oc.EnableVarLogCollection),
		cm.AsString("logging.revision-url-template", &oc.LoggingURLTemplate),
		cm.AsString(ReqLogTemplateKey, &oc.RequestLogTemplate),
```

4: FuzzChildName

Tests a small API that generates a name based on two strings: 1) A resource name and 2) a suffix. The API has a level of complexity in that it has a regex call on the supplied suffix.

5: FuzzSendRawMessage

`FuzzSendRawMessage` is the first of three fuzzers that were written for the `knative.dev/pkg/websocket` package. The fuzzer sets up a connection and sends a binary message using the fuzzers testcase as the contents for the message. The raw message is sent via the `SendRaw()` method which is a thin wrapper around the `github.com/gorilla/websocket.Conn.WriteMessage()` API. As such, this fuzzer mainly tests the `github.com/gorilla/websocket` code base, but does so from the perspective of Knatives use case.

6: FuzzNewRevisionThrottler

Tests the `revisionThrottler` of `knative.dev/serving`, specifically its `handleUpdate` method.

7: FuzzRouteReconciler

Tests the Route reconciler by creating two pseudo-randomized objects: one Revision and one Route. The fuzzer validates these two objects and creates a test controller that it uses to reconcile.

8: FuzzDomainNameFromTemplate

Tests an API in the reconciler for Route resources that extracts a domain name from a template specified in the network configuration. The fuzzer generates a randomized configuration and adds it to the context. It then invokes DomainNameFromTemplate with a meta v1 object and a name created by the fuzzer.

9: FuzzValidation

Tests the validation routines of the following resource types:

Name	Validation URL
`knative.dev/serving/pkg/apis/serving/v1.Revision`	https://github.com/knative/serving/blob/0bc4171698a23552cecc6618bbf47ff3fc506d4f/pkg/apis/serving/v1/revision_validation.go#L35
`knative.dev/serving/pkg/apis/autoscaling/v1alpha1.PodAutoscaler`	https://github.com/knative/serving/blob/6826a1ba8d2108ed6e835fee41d2ccc5639fece3/pkg/apis/autoscaling/v1alpha1/pa_validation.go#L28
`knative.dev/serving/pkg/apis/serving/v1.Metric`	https://github.com/knative/serving/blob/6826a1ba8d2108ed6e835fee41d2ccc5639fece3/pkg/apis/autoscaling/v1alpha1/metric_validation.go#L28
`knative.dev/serving/pkg/apis/serving/v1.Configuration`	https://github.com/knative/serving/blob/6826a1ba8d2108ed6e835fee41d2ccc5639fece3/pkg/apis/serving/v1/configuration_validation.go#L28
`knative.dev/serving/pkg/apis/serving/v1.Route`	https://github.com/knative/serving/blob/0bc4171698a23552cecc6618bbf47ff3fc506d4f/pkg/apis/serving/v1/route_validation.go#L29

`knative.dev/serving/pkg/apis/serving/v1.Service`	https://github.com/knative/serving/blob/0bc4171698a23552cecc6618bbf47ff3fc506d4f/pkg/apis/serving/v1/service_validation.go#L27
`knative.dev/serving/pkg/apis/serving/v1alpha1.DomainMapping`	https://github.com/knative/serving/blob/c24dc144afc51a535e7118bc113515be939a626b/pkg/apis/serving/v1alpha1/domainmapping_validation.go#L32
`knative.dev/serving/pkg/apis/serving/v1beta1.DomainMapping`	https://github.com/knative/serving/blob/5a51323d83e2b9dfccd8228e4f8604136831a902/pkg/apis/serving/v1beta1/domainmapping_validation.go#L32

The fuzzer selects a resource type to test, instantiates the resource and adds values to it. It then calls `Validate()` on the created resource.

10: FuzzMessagingRoundTripTypesToJSON
Implements a fuzzer similar to `TestMessagingRoundtripTypesToJSON()`: https://github.com/knative/eventing/blob/cae627dd99007f177f32d18d2bbff45fae7b1e7d/pkg/apis/messaging/v1/roundtrip_test.go#L84. Fuzzer 11, `FuzzMessagingRoundTripTypesToJSONExperimental`, implements the same test but with a more efficient roundtrip test.

11: FuzzMessagingRoundTripTypesToJSONExperimental
Implements a roundtrip fuzzer for the eventing v1 sources custom resource types. These are:

Resource	URL
`InMemoryChannel`	https://github.com/knative/eventing/blob/1aa90e544188960d273e59853d8ee767d06061d9/pkg/apis/messaging/v1/in_memory_channel_types.go#L33
`InMemoryChannelList`	https://github.com/knative/eventing/blob/1aa90e544188960d273e59853d8ee767d06061d9/pkg/apis/messaging/v1/in_memory_channel_types.go#L81
`Subscription`	https://github.com/knative/eventing/blob/d99685b2d967357578d04b56494ade28046d28be/pkg/apis/messaging/v1/subscription_types.go#L37
`SubscriptionList`	https://github.com/knative/eventing/blob/d99685b2d967357578d04b56494ade28046d28be/pkg/apis/messaging/v1/subscription_types.go#L138
`Channel`	https://github.com/knative/eventing/blob/a6afc4792bc5f0feb02a0c0062452428267a8449/pkg/apis/messaging/v1/channel_types.go#L34
`ChannelList`	https://github.com/knative/eventing/blob/a6afc4792bc5f0feb02a0c0062452428267a8449/pkg/apis/messaging/v1/channel_types.go#L92

The fuzzer was originally implemented here: https://github.com/knative/eventing/blob/cae627dd99007f177f32d18d2bbff45fae7b1e7d/pkg/apis/messaging/v1/roundtrip_test.go#L84 and called into `knative.dev/pkg/apis/testing/roundtrip.ExternalTypesViaJSON()` and we rewrote

`ExternalTypesViaJSON()` to be faster, as well as the custom functions of the messaging v1 types to be useable by the rewritten roundtrip test. The roundtrip test can be found here: https://github.com/AdamKorcz/kubefuzzing/tree/main/pkg/roundtrip.

12: FuzzSourcesRoundTripTypesToJSONExperimental

Implements a roundtrip fuzzer for the eventing v1 sources custom resource types. These are:

Resource	URL
ApiServerSource	https://github.com/knative/eventing/blob/b3184ba9dcd0d0891a46ec88a7ddfd91f3ae12e9/pkg/apis/sources/v1/apiserver_types.go#L33
ApiServerSourceList	https://github.com/knative/eventing/blob/b3184ba9dcd0d0891a46ec88a7ddfd91f3ae12e9/pkg/apis/sources/v1/apiserver_types.go#L126
SinkBinding	https://github.com/knative/eventing/blob/b20c96b4df513ebaa474ff3c4bf1b2a8c6f30d79/pkg/apis/sources/v1/sinkbinding_types.go#L38
SinkBindingList	https://github.com/knative/eventing/blob/b20c96b4df513ebaa474ff3c4bf1b2a8c6f30d79/pkg/apis/sources/v1/sinkbinding_types.go#L97
ContainerSource	https://github.com/knative/eventing/blob/d37dbc88e6fd4784a659cd5228c86bbb0986d7f6/pkg/apis/sources/v1/container_types.go#L34
ContainerSourceList	https://github.com/knative/eventing/blob/d37dbc88e6fd4784a659cd5228c86bbb0986d7f6/pkg/apis/sources/v1/container_types.go#L84
PingSource	https://github.com/knative/eventing/blob/6363a8f38d4656dbc741668c3ba5ac5e6b38e708/pkg/apis/sources/v1/ping_types.go#L34
PingSourceList	https://github.com/knative/eventing/blob/6363a8f38d4656dbc741668c3ba5ac5e6b38e708/pkg/apis/sources/v1/ping_types.go#L101

The fuzzer was originally implemented here: https://github.com/knative/eventing/blob/6363a8f38d4656dbc741668c3ba5ac5e6b38e708/pkg/apis/sources/v1/roundtrip_test.go#L79 and called into `knative.dev/pkg/apis/testing/roundtrip.ExternalTypesViaJSON()` and Ada Logics rewrote `ExternalTypesViaJSON()` to be faster, as well as the custom functions of the sources v1 types to be useable by the rewritten roundtrip test. The roundtrip test can be found here: https://github.com/AdamKorcz/kubefuzzing/tree/main/pkg/roundtrip.

13: FuzzRetryablehttpFromRequest

FuzzRetryablehttpFromRequest tests an API in a 3rd-party library used by Knative to convert an HTTP request to a request that performs automatic retries. The 3rd-party API is `github.com/hashicorp/go-retryablehttp.FromRequest()` which takes a `*net/http.Request` as argument and returns a

`*github.com/hashicorp/go-retryablehttp.Request`. The fuzzers testcase is used as the body for the `*net/http.Request`.

14: FuzzFilters

FuzzFilters tests the filter handling procedures in https://github.com/knative/eventing/blob/main/pkg/broker/filter/filter_handler.go. The fuzzer makes a call to either `applySubscriptionsAPIFilters()` or `applyAttributesFilter()` with filters and an event. As such, the fuzzer is a reduced version of `filterEvent()` without the logging and with a simplified selection of which API to call. `filterEvent()` is invoked in the broker handler.

More details about Knative brokers can be found here:

- About Brokers: https://knative.dev/docs/eventing/brokers/

15: FuzzDurableConnection

FuzzDurableConnection implements the second of three fuzzers for the `knative.dev/pkg/websocket` package. The fuzzer is similar to "5: FuzzSendRawMessage" in that it also sends a raw message, but it starts the connection with `knative.dev/pkg/websocket.NewDurableSendingConnection()`, whereas "5: FuzzSendRawMessage" starts the connection with `knative.dev/pkg/websocket.newConnection()`.

16: FuzzReceiveMessage

FuzzReceiveMessage is the third of three fuzzers written for the `knative.dev/pkg/websocket` package. This fuzzer tests the `keepAlive()` method which reads the next reader of a connection and sends it to the channel. The fuzzer randomises the data in the next reader of the connection.

17: FuzzDuckV1beta1RoundTripTypesToJSONExperimental

Implements a roundtrip fuzzer for the `knative.dev/pkg/apis/duck/v1beta1` custom resource types. These are:

Resource	URL
AddressableType	https://github.com/knative/pkg/blob/27fe4e19108060ab00018c8d98d87c76c934e47c/apis/duck/v1beta1/addressable_types.go#L57
AddressableTypeList	https://github.com/knative/pkg/blob/27fe4e19108060ab00018c8d98d87c76c934e47c/apis/duck/v1beta1/addressable_types.go#L124
KResource	https://github.com/knative/pkg/blob/a99300deff34c04163d69ef7e55ae7c2a87fe5da/apis/duck/v1beta1/status_types.go#L46
KResourceList	https://github.com/knative/pkg/blob/a99300deff34c04163d69ef7e55ae7c2a87fe5da/apis/duck/v1beta1/status_types.go#L146

Binding	https://github.com/knative/pkg/blob/27fe4e19108060ab00018c8d98d87c76c934e47c/apis/duck/v1beta1/binding_types.go#L33
BindingList	https://github.com/knative/pkg/blob/27fe4e19108060ab00018c8d98d87c76c934e47c/apis/duck/v1beta1/binding_types.go#L87

The fuzzer was originally implemented here: https://github.com/knative/pkg/blob/273ba59a1132a6d3036e609db955b9930a50091d/apis/duck/v1beta1/test/roundtrip_test.go and called into `knative.dev/pkg/apis/testing/roundtrip.ExternalTypesViaJSON()` which we rewrote to be faster.

18: FuzzDuckV1RoundTripTypesToJSONExperimental

Implements a roundtrip fuzzer for the `knative.dev/pkg/apis/duck/v1` custom resource types. These are:

Resource	URL
AddressableType	https://github.com/knative/pkg/blob/7101e9d4f6c65dad6dd3b5935deda37176976337/apis/duck/v1/addressable_types.go#L55
AddressableTypeList	https://github.com/knative/pkg/blob/7101e9d4f6c65dad6dd3b5935deda37176976337/apis/duck/v1/addressable_types.go#L116
KResource	https://github.com/knative/pkg/blob/27fe4e19108060ab00018c8d98d87c76c934e47c/apis/duck/v1/kresource_type.go#L50
KResourceList	https://github.com/knative/pkg/blob/27fe4e19108060ab00018c8d98d87c76c934e47c/apis/duck/v1/kresource_type.go#L84
Binding	https://github.com/knative/pkg/blob/b3f27fd9308b678731fe5eb557d598d73a7d6b3d/apis/duck/v1/binding_types.go#L33
BindingList	https://github.com/knative/pkg/blob/b3f27fd9308b678731fe5eb557d598d73a7d6b3d/apis/duck/v1/binding_types.go#L87

The fuzzer was originally implemented here: https://github.com/knative/pkg/blob/273ba59a1132a6d3036e609db955b9930a50091d/apis/duck/v1/test/roundtrip_test.go and called into `knative.dev/pkg/apis/testing/roundtrip.ExternalTypesViaJSON()` which we rewrote to be faster.

19: FuzzServingV1RoundTripTypesToJSONExperimental

Implements a roundtrip fuzzer for the `knative.dev/serving/pkg/apis/serving/v1` custom resource types. These are:

Resource	URL
Revision	https://github.com/knative/serving/blob/2e77abf553c97ca0e0cf751ce4144a52e

	6207751/pkg/apis/serving/v1/revision_types.go#L36
RevisionList	https://github.com/knative/serving/blob/2e77abf553c97ca0e0cf751ce4144a52e6207751/pkg/apis/serving/v1/revision_types.go#L178
Configuration	https://github.com/knative/serving/blob/491f288b3f33709a80ee7cea09db5dbd46f859fa/pkg/apis/serving/v1/configuration_types.go#L35
ConfigurationList	https://github.com/knative/serving/blob/491f288b3f33709a80ee7cea09db5dbd46f859fa/pkg/apis/serving/v1/configuration_types.go#L106
Route	https://github.com/knative/serving/blob/491f288b3f33709a80ee7cea09db5dbd46f859fa/pkg/apis/serving/v1/route_types.go#L37
RouteList	https://github.com/knative/serving/blob/491f288b3f33709a80ee7cea09db5dbd46f859fa/pkg/apis/serving/v1/route_types.go#L184
Service	https://github.com/knative/serving/blob/491f288b3f33709a80ee7cea09db5dbd46f859fa/pkg/apis/serving/v1/service_types.go#L43
ServiceList	https://github.com/knative/serving/blob/491f288b3f33709a80ee7cea09db5dbd46f859fa/pkg/apis/serving/v1/service_types.go#L132

The fuzzer was originally implemented here: https://github.com/knative/serving/blob/fad9d5535b86295cc20c13f9102fd525b8ced8d8/pkg/apis/serving/v1/roundtrip_test.go and called into `knative.dev/pkg/apis/testing/roundtrip.ExternalTypesViaJSON()` which we rewrote to be faster.

20: FuzzFlowsRoundTripTypesToJSONExperimental

Implements a roundtrip fuzzer for the `knative.dev/eventing/pkg/apis/flows/v1` custom resource types. These are:

Resource	URL
Sequence	https://github.com/knative/eventing/blob/4cc5ecf9635e16af876183798fe6a80c774f188f/pkg/apis/flows/v1/sequence_types.go#L37
SequenceList	https://github.com/knative/eventing/blob/4cc5ecf9635e16af876183798fe6a80c774f188f/pkg/apis/flows/v1/sequence_types.go#L139
Parallel	https://github.com/knative/eventing/blob/a34aaa09f7d25516ad289416dfb2876c1db70169/pkg/apis/flows/v1/parallel_types.go#L36
ParallelList	https://github.com/knative/eventing/blob/a34aaa09f7d25516ad289416dfb2876c1db70169/pkg/apis/flows/v1/parallel_types.go#L155

The fuzzer was originally implemented here: https://github.com/knative/eventing/blob/cae627dd99007f177f32d18d2bbff45fae7b1e7d/pkg/apis/flows/v1/roundtrip_test.go and called into `knative.dev/pkg/apis/testing/roundtrip.ExternalTypesViaJSON()` which we rewrote to be faster.

21: FuzzEventingRoundTripTypesToJSONExperimental
Implements a roundtrip fuzzer for the `knative.dev/eventing/pkg/apis/eventing/v1` custom resource types. These are:

Resource	URL
Broker	https://github.com/knative/eventing/blob/1048c6cfe1434a26372951d40bd14227257231d5/pkg/apis/eventing/v1/broker_types.go#L39
BrokerList	https://github.com/knative/eventing/blob/1048c6cfe1434a26372951d40bd14227257231d5/pkg/apis/eventing/v1/broker_types.go#L104
Trigger	https://github.com/knative/eventing/blob/480979187310810069c72e3e3b3c916775255c53/pkg/apis/eventing/v1/trigger_types.go#L45
TriggerList	https://github.com/knative/eventing/blob/480979187310810069c72e3e3b3c916775255c53/pkg/apis/eventing/v1/trigger_types.go#L202

The fuzzer was originally implemented here: https://github.com/knative/eventing/blob/cae627dd99007f177f32d18d2bbff45fae7b1e7d/pkg/apis/eventing/v1/roundtrip_test.go and called into `knative.dev/pkg/apis/testing/roundtrip.ExternalTypesViaJSON()` which we rewrote to be faster.

22: FuzzAvailabilityNodePriorityScore
Tests the AvailabilityNodePriority plugin of the eventing scheduler. The fuzzer passes random `args`, `states`, `feasiblePods`, `key` and `podID` to AvailabilityNodePriority's `Score` method.

23: FuzzAvailabilityZonePriorityScore
Tests the AvailabilityZonePriority plugin of the eventing scheduler. The fuzzer passes random `args`, `states`, `feasiblePods`, `key` and `podID` to AvailabilityNodePriority's `Score` method.

24: FuzzEvenPodSpreadFilter
Tests the EvenPodSpread plugin of the eventing scheduler. The fuzzer passes random `args`, `states`, `key` and `podID` to EvenPodSpread's `Filter` method.

25: FuzzEvenPodSpreadScore
Tests the EvenPodSpread plugin of the eventing scheduler. The fuzzer passes random `args`, `states`, `feasiblePods`, `key` and `podID` to EvenPodSpread's `Score` method.

26: FuzzRemoveWithAvailabilityNodePriorityScore

Tests the RemoveWithAvailabilityNodePriority plugin of the eventing scheduler. The fuzzer passes random `args`, `states`, `feasiblePods`, `key` and `podID` to RemoveWithAvailabilityNodePriority's `Score` method.

27: FuzzRemoveWithAvailabilityZonePriorityScore
Tests the RemoveWithAvailabilityZonePriority plugin of the eventing scheduler. The fuzzer passes random `args`, `states`, `feasiblePods`, `key` and `podID` to RemoveWithAvailabilityNodePriority's `Score` method.

28: FuzzRemoveWithEvenPodSpreadPriorityScore
Tests the RemoveWithEvenPodSpreadPriority plugin of the eventing scheduler. The fuzzer passes random `args`, `states`, `feasiblePods`, `key` and `podID` to RemoveWithEvenPodSpreadPriority's `Score` method.

29: FuzzValidateCESQLExpression
Tests the CESQL parsing that `knative.dev/eventing/pkg/pkg/apis/eventing/v1` does when validating the CESQL filter. The string from the fuzzer is passed directly to `ValidateCESQLExpression()`.

Issues found by fuzzers

Slice bounds out of range when parsing the schedule in PingSourceSpec validation

OSS-Fuzz bug tracker:	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=56798
Mitigation:	Fixed
ID:	ADA-KNAT-FUZZ-1

The fuzzers found a single issue during the audit. The issue had its root cause in a third-party dependency, github.com/robfig/cron/v3, which Knative eventing uses to validate the schedule from the `PingSourceSpec`:

https://github.com/knative/eventing/blob/24d102d28eed4981609c9b3b8f3f888d8b942f85/pkg/apis/sources/v1beta2/ping_validation.go#L39-L49

```
39 func (cs *PingSourceSpec) Validate(ctx context.Context) *apis.FieldError {
40       var errs *apis.FieldError
41       schedule := cs.Schedule
42
43       errs = validateDescriptor(schedule)
44
45       if cs.Timezone != "" {
46             schedule = "CRON_TZ=" + cs.Timezone + " " + schedule
47       }
48
49       if _, err := cron.ParseStandard(schedule); err != nil {
```