

PRESENTS

# Dapr Fuzzing Audit

In collaboration with the Dapr project maintainers and The Linux Foundation



## Authors

Adam Korczynski <[adam@adalogics.com](mailto:adam@adalogics.com)>

David Korczynski <[david@adalogics.com](mailto:david@adalogics.com)>

Date: 30th June 2023

This report is licensed under Creative Commons 4.0 (CC BY 4.0)

# 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<sup>1</sup>.

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) <https://github.com/cncf/cncf-fuzzing> where questions and queries are welcome.

---

<sup>1</sup> <https://github.com/google/oss-fuzz>

# Executive summary

In this engagement, Ada Logics worked on creating a fuzzing suite for Dapr. At the time of this engagement, Dapr was doing no fuzzing for any of its sub projects, and the goal of this fuzzing audit was to build the fundamental infrastructure and improve the fuzzing efforts in a continuous manner. Ada Logics did that by first integrating Dapr into OSS-Fuzz and add fuzzers for important API's of the Dapr eco system. At the end of the audit, all fuzzers are running continuously by way of OSS-Fuzz which will report if they find any crashes.

Ada Logics wrote a total of 39 fuzzers that found 3 issues - 2 of which had their root cause in 3rd-party libraries. At the time of the audits completion, all issues have been fixed. The fuzzers cover three of Daprs sub projects: 1) the Dapr Runtime, 2) Dapr kit and 3) Components-Contrib.

## Results summarised

39 fuzzers developed

All fuzzers added to Daprs OSS-Fuzz integration

Fuzzing covers the Dapr Runtime, Kit and Components-Contrib sub projects.

3 issues were found.

- 1 index out of range
- 2 panics in Go standard library

# Table of Contents

<b>CNCF security and fuzzing audits</b>	<b>2</b>
<b>Executive summary</b>	<b>3</b>
<b>Table of Contents</b>	<b>4</b>
<b>Dapr fuzzing</b>	<b>5</b>
<b>Issues found by fuzzers</b>	<b>13</b>
<b>Runtime stats</b>	<b>18</b>

# Dapr fuzzing

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

## Architecture

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

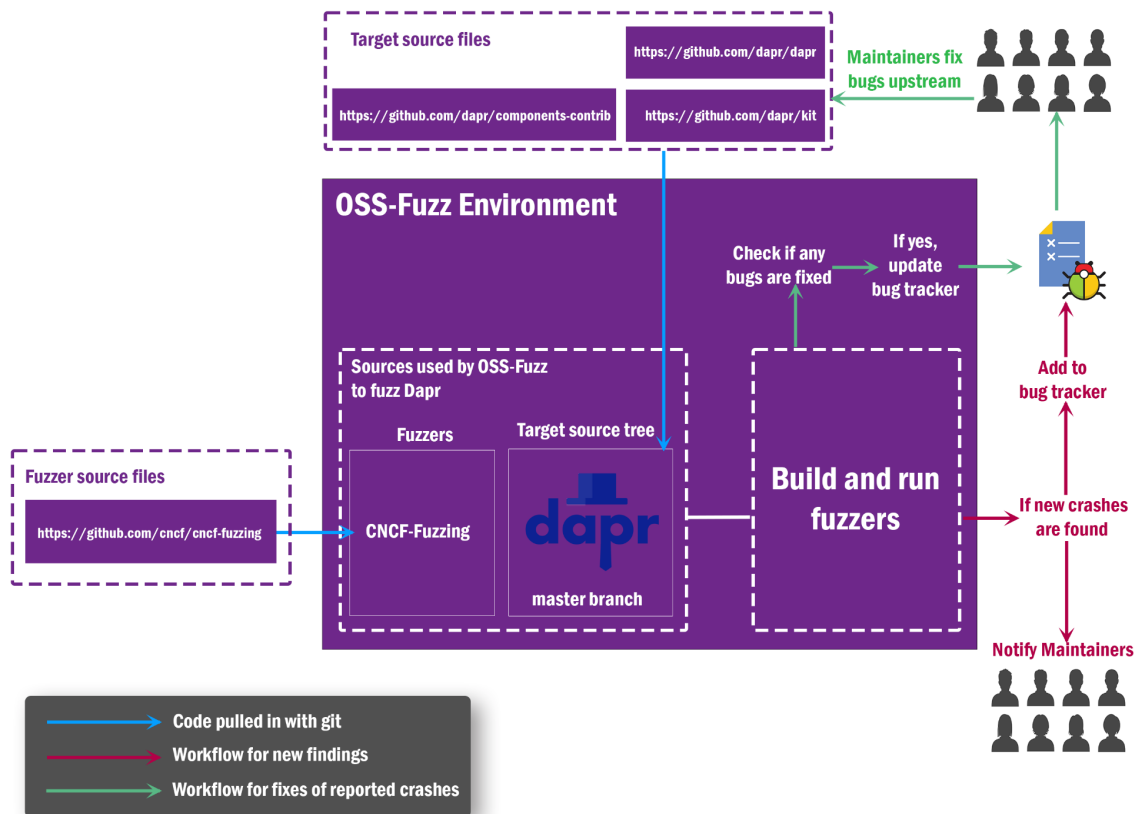


Figure 0.1: Dapr's fuzzing architecture

The current OSS-Fuzz set up builds the fuzzers by cloning the upstream Dapr Github repository to get the latest Dapr source code and the CNCF-Fuzzing Github repository to get the latest set of fuzzers, and then builds the fuzzers against the cloned Dapr code. As such, the fuzzers are always run against the latest Dapr 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 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 Dapr maintainers will fix issues upstream, and OSS-Fuzz will pull the latest Dapr master branch the next time it performs a fuzz run and verify that a given issue has been fixed.

## Dapr Fuzzers

In this section we present a highlight of the Dapr fuzzers and which parts of Dapr they test.

### Overview

#	Name	Package
1	FuzzExprDecodeString	github.com/dapr/dapr/pkg/expr
2	FuzzHandleRequest	github.com/dapr/dapr/pkg/injector
3	FuzzFSMPlacementState	github.com/dapr/dapr/pkg/placement/raft
4	FuzzDaprRuntime	github.com/dapr/dapr/pkg/runtime
5	FuzzInvokeRemote	github.com/dapr/dapr/pkg/messaging
6	FuzzParseAccessControlSpec	github.com/dapr/dapr/pkg/acl
7	FuzzActorsRuntime	github.com/dapr/dapr/pkg/actors
8	FuzzCryptoKeysAny	github.com/dapr/kit/crypto
9	FuzzCryptoKeysJson	github.com/dapr/kit/crypto
10	FuzzCryptoKeysRaw	github.com/dapr/kit/crypto
11	FuzzSymmetric	github.com/dapr/kit/crypto
12	FuzzAescbcaead	github.com/dapr/kit/crypto/aescbcaead
13	FuzzParseEnvString	github.com/dapr/dapr/pkg/injector/sidecar

14	<code>FuzzIsOperationAllowedByAccessControlPolicy</code>	<code>github.com/dapr/dapr/pkg/acl</code>
15	<code>FuzzIsEndpointAllowed</code>	<code>github.com/dapr/dapr/pkg/http</code>
16	<code>FuzzHTTPRegex</code>	<code>github.com/dapr/dapr/pkg/http</code>
17	<code>FuzzOnPostStateTransaction</code>	<code>github.com/dapr/dapr/pkg/http</code>
18	<code>FuzzOnBulkPublish</code>	<code>github.com/dapr/dapr/pkg/http</code>
19	<code>FuzzOnPublish</code>	<code>github.com/dapr/dapr/pkg/http</code>
20	<code>FuzzOnDirectActorMessage</code>	<code>github.com/dapr/dapr/pkg/http</code>
21	<code>FuzzOnDeleteActorTimer</code>	<code>github.com/dapr/dapr/pkg/http</code>
22	<code>FuzzOnGetActorReminder</code>	<code>github.com/dapr/dapr/pkg/http</code>
23	<code>FuzzOnActorStateTransaction</code>	<code>github.com/dapr/dapr/pkg/http</code>
24	<code>FuzzOnDeleteActorReminder</code>	<code>github.com/dapr/dapr/pkg/http</code>
25	<code>FuzzOnCreateActorTimer</code>	<code>github.com/dapr/dapr/pkg/http</code>
26	<code>FuzzOnRenameActorReminder</code>	<code>github.com/dapr/dapr/pkg/http</code>
27	<code>FuzzOnCreateActorReminder</code>	<code>github.com/dapr/dapr/pkg/http</code>
28	<code>FuzzOnDirectMessage</code>	<code>github.com/dapr/dapr/pkg/http</code>
29	<code>FuzzPublishEvent</code>	<code>github.com/dapr/dapr/pkg/grpc</code>
30	<code>FuzzInvokeService</code>	<code>github.com/dapr/dapr/pkg/grpc</code>
31	<code>FuzzBulkPublishEventAlpha1</code>	<code>github.com/dapr/dapr/pkg/grpc</code>
32	<code>FuzzStateEndpoints</code>	<code>github.com/dapr/dapr/pkg/grpc</code>
33	<code>FuzzActorEndpoints</code>	<code>github.com/dapr/dapr/pkg/grpc</code>
34	<code>FuzzGetConfiguration</code>	<code>github.com/dapr/dapr/pkg/grpc</code>
35	<code>FuzzDubboSerialization</code>	<code>github.com/dapr/components-contrib/bindings/dubbo</code>
36	<code>FuzzAddTopic</code>	<code>github.com/dapr/components-contrib/pubsub/mqtt3</code>
37	<code>FuzzQuery</code>	<code>github.com/dapr/components-contrib/state/query</code>
38	<code>FuzzCheckRequestOptions</code>	<code>github.com/dapr/components-contrib/state</code>
39	<code>FuzzDecodeMetadata</code>	<code>github.com/dapr/components-contrib/metadata</code>

## Target APIs

### 1: `FuzzExprDecodeString`

Tests the decoding of strings into an `Expr` type. The fuzzer uses the test case as the input for the `DecodeString()` api.

## 2: FuzzHandleRequest

Tests the request handling of the injector package. The fuzzer creates a valid admission review body by creating a `*k8s.io/api/admission/v1.AdmissionReview` with pseudo-randomized values and then marshalling that into bytes. The fuzzer then creates a request with these bytes as its body and passes the request to the injectors http request handling api, `github.com/dapr/dapr/pkg/injector.(*injector).handleRequest()`.

## 3: FuzzFSMPlacementState

This fuzzer tests the fsm's handling of raft log entries. The fuzzer creates a new FSM (finite state machine). It then creates a raft log and applies it to the FSM. Finally, the fuzzer invokes the FSM's `PlacementState` method to test if a raft log entry could cause disruption.

## 4: FuzzDaprRuntime

This is an extensive fuzzer that tests the dapr runtime package. The fuzzer implements its own mocked pubsub type. The fuzzer has 4 targets:

1. `processComponentAndDependents()`:  
<https://github.com/dapr/dapr/blob/827678fc9823f7ebbd6edc2dd00e140cf5309f/pkg/runtime/runtime.go#L2634>
2. `Publish()`:  
<https://github.com/dapr/dapr/blob/827678fc9823f7ebbd6edc2dd00e140cf5309f/pkg/runtime/runtime.go#L2069>
3. `BulkPublish()`:  
<https://github.com/dapr/dapr/blob/827678fc9823f7ebbd6edc2dd00e140cf5309f/pkg/runtime/runtime.go#L2095>
4. `Subscribe()`:  
<https://github.com/dapr/dapr/blob/827678fc9823f7ebbd6edc2dd00e140cf5309f/pkg/runtime/runtime.go#L2126>

## 5: FuzzInvokeRemote

Tests the `invokeRemoteStream()` method of `github.com/dapr/dapr/pkg/messaging.(*directMessaging)`. The fuzzer creates a new request using `github.com/dapr/dapr/pkg/messaging/v1.NewRequest`, using the fuzz test case to randomize the raw data, the actors and the metadata.

## 6: FuzzParseAccessControlSpec

Tests the parsing routine for `github.com/dapr/dapr/pkg/config.AccessControlSpec`. The fuzzer randomizes the fields of an `AccessControlSpec` and passes it onto `ParseAccessControlSpec`.



## 7: FuzzActorsRuntime

Tests Daprs implementation of the [github.com/dapr/dapr/pkg/actors](https://github.com/dapr/dapr/pkg/actors).Actors interface. The fuzzer initiates a new actorsRuntime and calls the following methods in pseudo-random order using pseudo-random values for each call:

1. `Call()`
2. `GetState()`
3. `TransactionalStateOperation()`
4. `IsActorHosted()`
5. `CreateReminder()`
6. `CreateTimer()`
7. `DeleteTimer()`
8. `RenameReminder()`

## 8: FuzzCryptoKeysAny

This fuzzer tests key parsing and serialization routines in [github.com/dapr/kit/crypto](https://github.com/dapr/kit/crypto). The fuzzer carries out three steps: It first creates a new key using the test case as the raw bytes. It then serializes the key and finally compares the returned bytes with the initial raw bytes. If these are not equal, then the fuzzer panics.

## 9: FuzzCryptoKeysJson

Similar to `FuzzCryptoKeys` but only invokes parsing of json-formatted raw bytes.

## 10: FuzzCryptoKeysRaw

Similar to `FuzzCryptoKeysAny` but does not specify a content type.

## 11: FuzzSymmetric

Tests the encryption and decryption apis of [github.com/dapr/kit/crypto](https://github.com/dapr/kit/crypto). The fuzzer first encrypts and then decrypts using a set of randomized parameters. It then compares whether the decrypted plain text is identical as the initial, panicking if it is not.

## 12: FuzzAescbcaead

Tests Daprs AEAD implementation in [github.com/dapr/kit/crypto/aescbcaead](https://github.com/dapr/kit/crypto/aescbcaead). The fuzzer carries out three steps: It first creates a cipher using one of Daprs own apis for the same:

1. `github.com/dapr/kit/crypto/aescbcaead.NewAESCBC128SHA256()`
2. `github.com/dapr/kit/crypto/aescbcaead.NewAESCBC192SHA384()`
3. `github.com/dapr/kit/crypto/aescbcaead.NewAESCBC256SHA384()`
4. `github.com/dapr/kit/crypto/aescbcaead.NewAESCBC256SHA512()`

For this step the fuzzer uses raw bytes by the fuzzer. It then proceeds to invoke the ciphers two methods, `Seal()` and `Open()` using pseudo-random data by the fuzzer for both method calls.

### 13: FuzzParseEnvString

Tests the `ParseEnvString()` with a string created by the fuzzer.

### 14: FuzzIsOperationAllowedByAccessControlPolicy

Tests whether input to `IsOperationAllowedByAccessControlPolicy()` can cause crashes. `IsOperationAllowedByAccessControlPolicy()` is particularly exposed to input of lower trust.

### 15: FuzzIsEndpointAllowed

Dapr relies on a simple check to verify that a request has access to an endpoint:

<https://github.com/dapr/dapr/blob/ca08aa97deebf93306a66af7c31c9b1309f7a9b3/pkg/http/endpoint.go#L71>. This check relies on the standard Go API `strings.HasPrefix`.

`FuzzIsEndpointAllowed` tests whether `strings.HasPrefix` correctly fulfills the security assumption Dapr has made about it.

### 16: FuzzHTTPRegex

Tests an exposed Regex that extracts parameters from incoming requests.

### 17: FuzzOnPostStateTransaction

Tests the `onPostStateTransaction()` HTTP endpoint with a request body specified by the fuzzer.

### 18: FuzzOnBulkPublish

Tests the `onBulkPublish()` HTTP endpoint with a request body specified by the fuzzer.

### 19: FuzzOnPublish

Tests the `onPublish()` HTTP endpoint with a request body specified by the fuzzer.

### 20: FuzzOnDirectActorMessage

Tests the `onDirectActorMessage()` HTTP endpoint with a request body specified by the fuzzer.

### 21: FuzzOnDeleteActorTimer

Tests the `onDeleteActorTimer()` HTTP endpoint with a request body specified by the fuzzer.

## 22: FuzzOnGetActorReminder

Tests the `onGetActorReminder()` HTTP endpoint with a request body specified by the fuzzer.

## 23: FuzzOnActorStateTransaction

Tests the `onActorStateTransaction()` HTTP endpoint with a request body specified by the fuzzer.

## 24: FuzzOnDeleteActorReminder

Tests the `onDeleteActorReminder()` HTTP endpoint with a request body specified by the fuzzer.

## 25: FuzzOnCreateActorTimer

Tests the `onCreateActorTimer()` HTTP endpoint with a request body specified by the fuzzer.

## 26: FuzzOnRenameActorReminder

Tests the `onRenameActorReminder()` HTTP endpoint with a request body specified by the fuzzer.

## 27: FuzzOnCreateActorReminder

Tests the `onCreateActorReminder()` HTTP endpoint with a request body specified by the fuzzer.

## 28: FuzzOnDirectMessage

Tests the `onDirectMessage()` HTTP endpoint with a request body specified by the fuzzer.

## 29: FuzzPublishEvent

Tests the `PublishEvent()` GRPC endpoint with a request body specified by the fuzzer.

## 30: FuzzInvokeService

Tests the `InvokeService()` GRPC endpoint with a request body specified by the fuzzer.

## 31: FuzzBulkPublishEventAlpha1

Tests the `BulkPublishEventAlpha1()` HTTP endpoint with a request body specified by the fuzzer.

## 32: FuzzStateEndpoints

Tests the GRPC endpoints related to state with requests containing a body specified by the fuzzer.

### 33: FuzzActorEndpoints

Tests the GRPC endpoints related to actors with requests containing a body specified by the fuzzer.

### 34: FuzzGetConfiguration

Tests the `GetConfiguration()` GRPC endpoint with a request body specified by the fuzzer.

### 35: FuzzDubboSerialization

Tests the Hessian serializer implemented in

`dubbo.apache.org/dubbo-go/v3/protocol/dubbo/impl` which Dapr Components-Contrib uses.

### 36: FuzzAddTopic

Tests the `addTopic()` method of the mqtt pubsub.

### 37: FuzzQuery

Tests the parsing routing of Dapr Components-Contrib uses in its `query` package.

### 38: FuzzCheckRequestOptions

Tests the CheckRequestOptions API in Dapr Components-Contrib with state options for Get, Delete and Set.

### 39: FuzzDecodeMetadata

Tests the decoding routine which handles metadata across the Dapr Components-Contrib source tree.

# Issues found by fuzzers

The fuzzers found three issues during the time of the audit itself. One of these were in Daprs code base and the remaining two were in 3rd-party dependencies. At the end of the audit, all issues have been fixed.

The fuzzers continue to test Daprs code and might find more bugs in the same code or new bugs introduced after the audit itself. If that happens, OSS-Fuzz will notify the Dapr team with a stacktrace and a reproducer testcase.

#	Title	Mitigation
1	Index out of range in raft log reading	Fixed
2	Malicious raw key triggers out of range panic in Go standard library	Fixed
3	Key with empty seed will trigger panic in Go standard library	Fixed

## Index out of range in raft log reading

OSS-Fuzz bug tracker:	<a href="https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=58799">https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=58799</a>
Mitigation:	Fixed in <a href="https://github.com/dapr/dapr/pull/6343">https://github.com/dapr/dapr/pull/6343</a>
ID:	ADA-DAP-FUZZ-1

### Description

A fuzzer found an index out of bounds in Dapr fsm's reading of raft log data. The issue was a missing bounds check for the data of a raf log entry, however, the raft log entry could be shorter than the requested slice index resulting in an out of range panic. The issue existed on the highlighted lines below:

<https://github.com/dapr/dapr/blob/1c95ad119a4257d1f0f1403eda0aced56c3fe848/pkg/placement/raft/fsm.go#L145>

```
145 func (c *FSM) Apply(log *raft.Log) interface{} {
146     var (
147         err      error
148         updated bool
149     )
150
151     if log.Index < c.state.Index() {
152         logging.Warnf("old: %d, new index: %d. skip apply", c.state.Index,
153 log.Index)
154         return false
155     }
156
157     switch CommandType(log.Data[0]) {
158     case MemberUpsert:
159         updated, err = c.upsertMember(log.Data[1:])
160     case MemberRemove:
161         updated, err = c.removeMember(log.Data[1:])
162     default:
163         err = errors.New("unimplemented command")
164     }
165
166     if err != nil {
167         logging.Errorf("fsm apply entry log failed. data: %s, error: %s",
168             string(log.Data), err.Error())
169         return false
170     }
171
172     return updated
173 }
```

Figure 1.1: Proof of concept payload to trigger issue ADA-DAP-FUZZ-1

# Malicious raw key triggers out of range panic in Go standard library

OSS-Fuzz bug tracker:	<a href="https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=58954">https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=58954</a>
Mitigation:	Fixed in: <a href="https://github.com/golang/go/issues/60411#event-9334104392">https://github.com/golang/go/issues/60411#event-9334104392</a>
ID:	ADA-DAP-FUZZ-2

## Description

A fuzzer testing kit/crypto found that malicious raw bytes can be parsed into a key that will trigger a panic in the Go standard library, when the key gets serialized. This is illustrated with the below PoC:

```
1 package main
2
3 import (
4     "bytes"
5     "fmt"
6     kitCrypto "github.com/dapr/kit/crypto"
7 )
8
9 func main() {
10     b := []byte{0xD, 0xD, 0xD, 0xD, 0xD, 0xD, 0xD, 0xD, 0x7B, 0xD, 0xD, 0xD,
11     0x9, 0x22, 0x2D, 0x22, 0x3A, 0x22, 0x8D, 0x8D, 0x8D, 0x8D, 0x5D, 0x3B, 0xFF, 0x72,
12     0x8D, 0x8D, 0x8D, 0x8D, 0x8D, 0x8D, 0x5D, 0x22, 0x2C, 0xA, 0xA, 0x22, 0x65, 0x22,
13     0x3A, 0x22, 0x59, 0x55, 0x75, 0x22, 0x2C, 0x9, 0xD, 0x22, 0x6E, 0x22, 0x3A, 0x22,
14     0x59, 0x55, 0x75, 0x22, 0x2C, 0x9, 0xD, 0x9, 0x22, 0x78, 0x28, 0x74, 0x78, 0x35,
15     0x75, 0x22, 0x3A, 0x22, 0x8D, 0x8D, 0x8D, 0x8D, 0x8D, 0x8D, 0x8D, 0x8D, 0x94,
16     0x5D, 0x22, 0x2C, 0x22, 0x74, 0x22, 0x3A, 0x22, 0x5D, 0x8D, 0x8D, 0x8D, 0x8D, 0x8D,
17     0x8D, 0x8D, 0x5D, 0x22, 0x2C, 0xA, 0xA, 0x22, 0x64, 0x22, 0x3A, 0x22, 0x59,
18     0x55, 0x75, 0x22, 0x2C, 0x9, 0xD, 0x22, 0x65, 0x22, 0x3A, 0x22, 0x59, 0x55, 0x75,
19     0x22, 0x2C, 0x9, 0xD, 0x9, 0x22, 0x78, 0x28, 0x74, 0x78, 0x35, 0x75, 0x22, 0x3A,
20     0x22, 0x8D, 0x8D, 0x8D, 0x8D, 0x8D, 0x8D, 0x8D, 0x8D, 0x8D, 0x8D, 0x94, 0x5D, 0x22,
21     0x2C, 0x22, 0x74, 0x22, 0x3A, 0x22, 0x5D, 0x22, 0x2C, 0x9, 0xD, 0x9, 0x22, 0x78,
22     0x34, 0x74, 0x22, 0x3A, 0x22, 0x7B, 0x8D, 0x22, 0x2C, 0x9, 0xD, 0x9, 0xD, 0xD, 0xD,
23     0xD, 0xD, 0xD, 0x22, 0x6B, 0x74, 0x79, 0x22, 0x3A, 0x22, 0x52, 0x53, 0x41, 0x22,
24     0x7D, 0x0, 0x0, 0x0, 0xB, 0xFF, 0xFF, 0x8D, 0x8D, 0x86, 0x22, 0xD, 0x0, 0x86, 0x86,
25     0x86, 0x86, 0x86, 0x86}
26     k, err := kitCrypto.ParseKey(b, "application/json")
27     if err != nil {
28         panic(err)
29     }
30     b2, err := kitCrypto.SerializeKey(k)
31     if err != nil {
32         panic(err)
33     }
34 }
```

Figure 2.1: Proof of concept payload to trigger issue ADA-DAP-FUZZ-2

Running this PoC will result in the following panic:

```
panic: runtime error: index out of range [-1]

goroutine 1 [running]:
crypto/internal/bigmod.NewModulusFromBig(0xc00014e080?)
    /usr/local/go/src/crypto/internal/bigmod/nat.go:390 +0x173
crypto/rsa.(*PrivateKey).Precompute(0xc00014e080)
    /usr/local/go/src/crypto/rsa/rsa.go:560 +0x99
crypto/x509.MarshalPKCS1PrivateKey(0xc00014e080)
    /usr/local/go/src/crypto/x509/pkcs1.go:105 +0x33
crypto/x509.MarshalPKCS8PrivateKey({0x6d2940?, 0xc00014e080?})
    /usr/local/go/src/crypto/x509/pkcs8.go:110 +0x2df
github.com/dapr/kit/crypto.SerializeKey({0x772c60, 0xc0001166c0})
    /home/adam/go/pkg/mod/github.com/dapr/kit@v0.0.5/crypto/keys.go:42 +0x125
main.main()
    /tmp/gopoc/main.go:16 +0x1e9
exit status 2
```

Tested with go version go1.20.2 linux/amd64.



## Key with empty seed will trigger panic in Go standard library

OSS-Fuzz bug tracker:	<a href="https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=59669">https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=59669</a>
Mitigation:	Fixed in: <a href="https://github.com/lestrrat-go/jwx/pull/947">https://github.com/lestrrat-go/jwx/pull/947</a>
ID:	ADA-DAP-FUZZ-3

When Dapr serializes a key with the `github.com/dapr/kit/crypto.SerializeKey()` utility, a malicious key can trigger a panic in the standard library. The panic happens when a 3rd-party dependency of Dapr - `github.com/lestrrat-go/jwx/v2` - calls its internal `buildOKPPrivateKey` with an empty `dbuf`:

<https://github.com/lestrrat-go/jwx/blob/639b10fcc1da45557f5eb419ceb76fab5c9a597b/jwk/okp.go#L86>

```
83 func buildOKPPrivateKey(alg jwa.EllipticCurveAlgorithm, xbuf []byte, dbuf []byte)
   (interface{}, error) {
84     switch alg {
85     case jwa.Ed25519:
86         ret := ed25519.NewKeyFromSeed(dbuf)
```

*Figure 3.1: Point of failure for ADA-DAP-FUZZ-3*

... which triggers the panic `panic: ed25519: bad seed length: 0`.

# Runtime stats

Continuity is an important element in fuzzing because fuzzers incrementally build up a corpus over time, therefore, the size of the corpus is a reflection of how much code the fuzzer has explored. OSS-Fuzz prioritises running fuzzers that continue to explore more code, and the CPU time presented by OSS-Fuzz runtime stats is thus a reflection of how much work the fuzzers have performed. The following tables lists for each fuzzer<sup>2</sup> the amounts of tests executed as well as the total CPU hours devoted.

Some of the fuzzers have low runtime stats because they were added later in the audit, and OSS-Fuzz has not run these excessively yet. Over time, OSS-Fuzz will dedicate the same resources to these fuzzers as it did for the fuzzers with hundreds or thousands or runtime hours.

Name	Total times executed	Total runtime (hours)
FuzzExprDecodeString	88,408,790	6,854.2
FuzzHandleRequest	2,264,178,745	11,835.4
FuzzFSMPlacementState	2,894,171,147	1,853.7
FuzzDaprRuntime	3,865,365,388	6,121.6
FuzzInvokeRemote	9,546,401,556	10,948.5
FuzzParseAccessControlSpec	13,773,856,623	2,675.7
FuzzActorsRuntime	567,164,460	7,103.4
FuzzCryptoKeysAny	1,198,741,277	1,243.2
FuzzCryptoKeysJson	993,721,224	182
FuzzCryptoKeysRaw	14,978,721	30.7
FuzzSymmetric	54,314,731,340	2,494.8
FuzzAescbcaead	38,739,137,129	1,332.7
FuzzParseEnvString	10,037,103,269	1,083.4
FuzzIsOperationAllowedByAccessControlPolicy	14,773,069,800	1,040
FuzzIsEndpointAllowed	14,946,766,284	948
FuzzHTTPRegex	8,254,046,809	973.2
FuzzOnPostStateTransaction	293,048,460	158.9
FuzzOnBulkPublish	250,004,357	79.6
FuzzOnPublish	234,991,607	102.8

---

<sup>2</sup> As per 29th June 2023.

FuzzOnDirectActorMessage	148,690,907	113.7
FuzzOnDeleteActorTimer	1,828,673,833	160.7
FuzzOnGetActorReminder	n/a	n/a
FuzzOnActorStateTransaction	1,479,628,808	125.2
FuzzOnDeleteActorReminder	1,407,702,416	131.5
FuzzOnCreateActorTimer	1,615,826,686	124.5
FuzzOnRenameActorReminder	783,778,610	100.9
FuzzOnCreateActorReminder	1,212,786,262	150.7
FuzzOnDirectMessage	639,129,845	135
FuzzPublishEvent	2,157,023	16.2
FuzzInvokeService	0	0
FuzzBulkPublishEventAlpha1	167,313	0
FuzzStateEndpoints	70,671,836	28.7
FuzzActorEndpoints	50,468,466	23
FuzzGetConfiguration	195,269,643	10.9
FuzzDubboSerialization	9,560	0
FuzzAddTopic	n/a	n/a
FuzzQuery	394,433,161	3.7
FuzzCheckRequestOptions	n/a	n/a
FuzzDecodeMetadata	104,056,233	3.7