PRESENTS

Crossplane Fuzzing Audit

In collaboration with the Crossplane project maintainers and The Linux Foundation

Authors

Adam Korczynski <adam@adalogics.com>
David Korczynski <david@adalogics.com>
Date: 23rd 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

In July 2022, Ada Logics started a fuzzing audit of Crossplane. The goal of the audit was to improve Crossplanes fuzzing efforts. At the time the audit started, Crossplane had not done any prior fuzz testing, and the audit began with integrating Crossplane into OSS-Fuzz - Googles open source project that runs the fuzzers of critical open source projects and reports issues to maintainers if the fuzzers detect any. Once Crossplane was integrated into OSS-Fuzz, Ada Logics began assessing Crossplanes code base to determine which entrypoints the fuzzers should target. In total three code assets were tested in this audit:

https://github.com/crossplane/crossplane:	The core Crossplane repository.
https://github.com/crossplane/crossplane-runtime:	A set of go libraries used to build controllers in the Crossplane stack.
github.com/google/go-containerregistry:	A 3rd-party dependency that handles complex parsing for Crossplane

All fuzzers were merged into the cncf-fuzzing repository, https://github.com/cncf/cncf-fuzzing, from which OSS-Fuzz would pull them and test against the latest master branches of the three repositories being tested.

Ada Logics created a PR for CI integration, so that the fuzzers can test new code contributions to Crossplane: https://github.com/crossplane/crossplane/pull/3553.

The audit found 4 issues: 2 in Crossplane and 2 in 3rd-party dependencies.

Results summarised

13 fuzzers written for Crossplane
4 issues found
All fuzzers integrated into OSS-Fuzz
2 security vulnerabilities found by the fuzzers
Fuzzers supported by Crossplanes CI

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

Crossplane maintainers involved in the audit

Name	Title	Email
Philippe Scorsolini	Senior Software Engineer	philippe.scorsolini@upbound.io
Jared Watts	Steering Committee and Crossplane Maintainer	jared@upbound.io
Alper Ulucinar	Staff Software Engineer	alper@upbound.io
Nic Cope	Crossplane Steering Committee	negz@upbound.io

Assets

Url	Branch
https://github.com/crossplane/crossplane	`master`
https://github.com/crossplane/crossplane-runtime	`master`
https://github.com/google/go-containerregistry	`main`

Crossplane fuzzing

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

Architecture

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

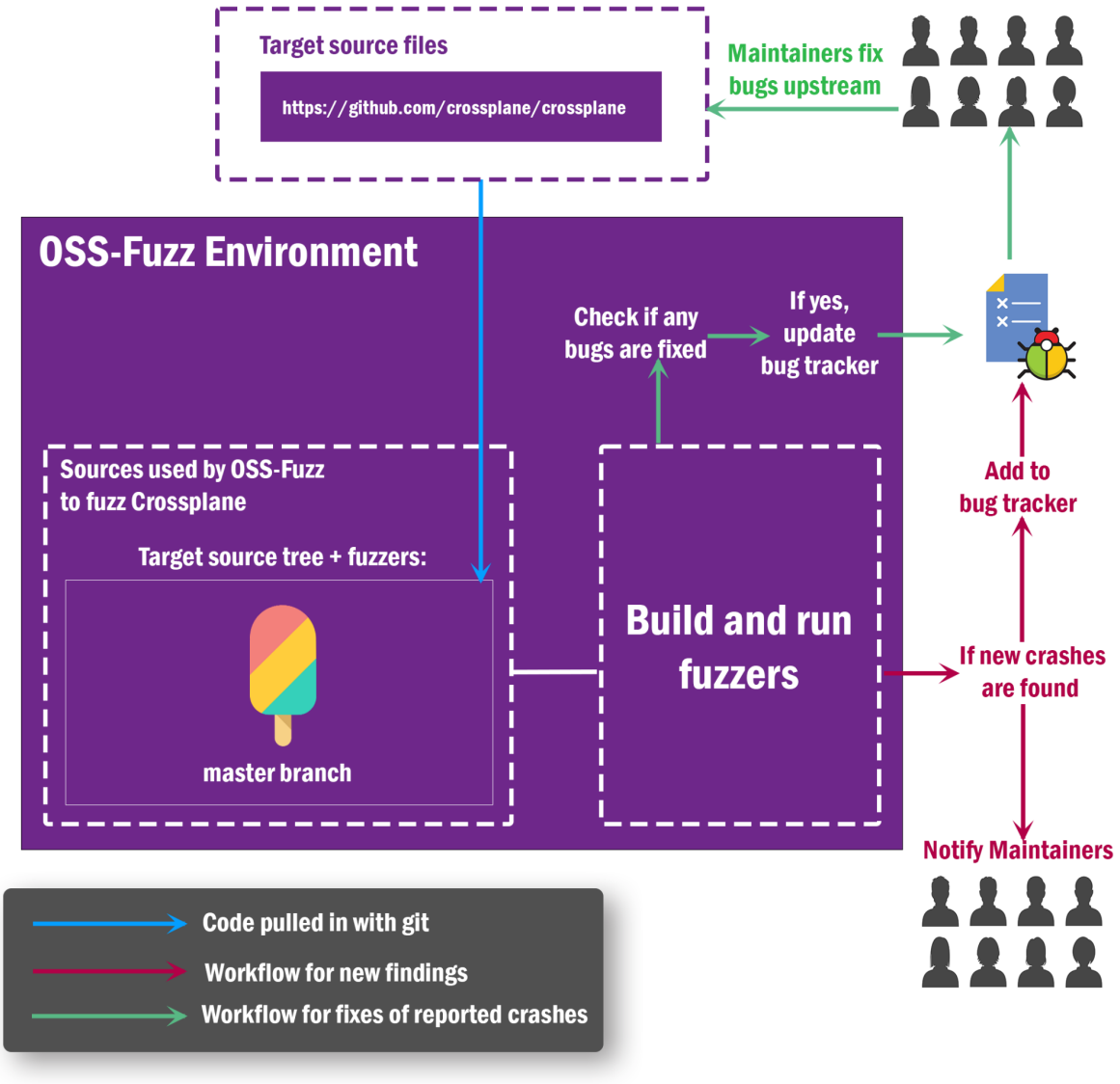

Figure 0.1: Crossplane's fuzzing architecture

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

Fuzzers

Ada Logics wrote 13 fuzzers during the fuzzing audit. In this section we present a highlight of the fuzzers and which parts of Crossplane they test.

Overview

#	Name	Package
1	`FuzzPatchApply`	`github.com/crossplane/crossplane/internal/controller/apiextensions/composite`
2	`FuzzTransform`	`github.com/crossplane/crossplane/internal/controller/apiextensions/composite`
3	`FuzzParse`	`github.com/crossplane/crossplane/internal/xpkg`
4	`FuzzPropagateConnection`	`github.com/crossplane/crossplane/internal/controller/apiextensions/claim`
5	`FuzzNewCompositionRevision`	`github.com/crossplane/crossplane/internal/controller/apiextensions/composition`
6	`FuzzAsComposition`	`github.com/crossplane/crossplane/internal/controller/apiextensions/composite`
7	`FuzzPackageRevision`	`github.com/crossplane/crossplane/internal/controller/pkg/manager`
8	`FuzzGCRExtract*`	`github.com/crossplane/crossplane/internal/controller/pkg/revision`

9	FuzzParseReference*	github.com/crossplane/crossplane/internal/controller/pkg/revision
10	FuzzForCompositeResourceXcrd	github.com/crossplane/crossplane/internal/xcrd
11	FuzzForCompositeResourceClaim	github.com/crossplane/crossplane/internal/xcrd
12	FuzzFindXpkgInDir	github.com/crossplane/crossplane/internal/xpkg
13	FuzzDag	github.com/crossplane/crossplane/internal/dag

* Removed after the audit.

Target APIs

1: FuzzPatchApply

Invokes `github.com/crossplane/crossplane/internal/controller/apiextensions/composite.Apply()` with a pseudo-randomized `Patch` and two mocked types as `Composite` and `Composed` respectively.

2: FuzzTransform

Creates a pseudo-randomized `github.com/crossplane/crossplane/apis/apiextensions/v1.Transform` and passes it to `github.com/crossplane/crossplane/internal/controller/apiextensions/composite.Resolve()` together with a pseudo-random string as the second parameter.

3: FuzzParse

Tests the Crossplane Runtime parser with a `NopCloser` containing the testcase from the fuzzer.

4: FuzzPropagateConnection

Tests `github.com/crossplane/crossplane/internal/controller/apiextensions/claim.(a *APIConnectionPropagator).PropagateConnection()`. The fuzzer creates an `APIConnectionPropagator` with a mock client and invokes its `PropagateConnection()` method with a pseudo-randomized `Composite` and `Composed`.

5: FuzzNewCompositionRevision

Invokes `github.com/crossplane/crossplane/internal/controller/apiextensions/composition.NewCompositionRevision()` with pseudo-randomized `revision` and `compSpecHash` parameters.

6: FuzzAsComposition

Randomizes a `github.com/crossplane/crossplane/apis/apiextensions/v1beta1.CompositionRevision` and passes it to

`github.com/crossplane/crossplane/internal/controller/apiextensions/composite.AsComposition()`.

7: FuzzPackageRevision
Has two targets: `github.com/crossplane/crossplane/internal/controller/pkg/manager.(*PackageRevisioner).Revision()` and `github.com/crossplane/crossplane/internal/xpkg.FriendlyID()`. To invoke `Revision()`, the fuzzer creates a `*PackageRevisioner` with a mock fetcher. It then invokes `Revision()` with a pseudo-randomized `github.com/crossplane/crossplane/apis/pkg/v1.Provider`. After that, the fuzzer proceeds to creating two strings and passing them onto `FriendlyID()`.

8: FuzzGCRExtract*
Tests the 3rd-party API used by Crossplane: `github.com/google/go-containerregistry/pkg/v1/mutate.Extract()`. The fuzzer creates a tar file that it then passes to the target API to be extracted.

9: FuzzParseReference*
Tests the 3rd-party API used by Crossplane: `github.com/google/go-containerregistry/pkg/name.ParseReference()`. The testcase from the fuzzer is passed to the target API.

10: FuzzForCompositeResourceXcrd
Tests `github.com/crossplane/crossplane/internal/xcrd.ForCompositeResourceXcrd()` by creating a pseudo-random `github.com/crossplane/crossplane/apis/apiextensions/v1.CompositeResourceDefinition` and passing it to the target API.

11: FuzzForCompositeResourceClaim
Tests `github.com/crossplane/crossplane/internal/xcrd.FuzzForCompositeResourceClaim()` by creating a pseudo-random `github.com/crossplane/crossplane/apis/apiextensions/v1.CompositeResourceDefinition` and passing it to the target API.

12: FuzzFindXpkgInDir
The goal of this fuzzer is to test `github.com/crossplane/crossplane/internal/xpkg.FindXpkgInDir()` and `github.com/crossplane/crossplane/internal/xpkg.ParseNameFromMeta()`. The fuzzer prepares a `github.com/spf13/afero.MemMapFs` and creates pseudo-random files in it. The `MemMapFs` is then passed to the two target APIs as the first argument and "`/`" as the second.

13: FuzzDag

Tests `MapDag` initialization by creating a slice of pseudo-random nodes and passing it to `github.com/crossplane/crossplane/internal/dag.(d *MapDag).Init()`.

* Removed after the audit.

Issues found by fuzzers

The audit resulted in 4 unique issues. Some of these were reported with multiple similar crashes by OSS-Fuzz. This happens for several reasons: For example, changes in Crossplane meant that a crash was moved from one place to another in the codebase causing a testcase to be invalid and a new one to be valid. Ada Logics assessed all crashes reported by OSS-Fuzz and have only included the unique ones in this report.

#	ID	Title	Severity	Fixed
1	ADA-CROSS-FUZZ-1	User-controlled variable determines slice size	High	Yes
2	ADA-CROSS-FUZZ-2	runtime.DefaultUnstructuredConverter.ToUnstructured panics if if a pointer-typed field without omitempty specifier is attempted to be converted	Moderate	Yes
3	ADA-CROSS-FUZZ-3	Missing slice length checks causes out of bounds read	Low	Yes
4	ADA-CROSS-FUZZ-4	Time out in 3rd-party dependency	Moderate	Yes

The most interesting finding was ADA-CROSS-FUZZ-1 which was a bug that allowed partially-trusted users to control the size of allocated memory of the node without an upper limit. The issue was triaged by the Crossplane maintainers and was assigned assigned CVE's in both Crossplane and Crossplane-runtime:

Project	Github Advisory	CVE
Crossplane	GHSA-v829-x6hh-cqfq	CVE-2023-27484
Crossplane-runtime	GHSA-vfvj-3m3g-m532	CVE-2023-27483

1: User-controlled variable determines slice size

ID	ADA-CROSS-FUZZ-1
Severity	High
Fixed	Yes
Fuzzer	`FuzzPatchApply`
OSS-Fuzz bug tracker: • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=49338&q=crossplane&can=1 (expired) • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=53819&q=crossplane&can=1	

Description

Crossplane runtime attempts to create a slice that is larger than the memory available. The size of the slice is determined by input controlled by the fuzzer. The Crossplane team triaged the issue and found that it was security-critical and impacted both Crossplane and Crossplane-runtime. They requested CVE's for both projects:

Project	Github Advisory	CVE
Crossplane	GHSA-v829-x6hh-cqfq	CVE-2023-27484
Crossplane-runtime	GHSA-vfvj-3m3g-m532	CVE-2023-27483

This issue existed in two places in https://github.com/crossplane/crossplane-runtime/blob/master/pkg/fieldpath/paved.go highlighted below:

```
func prepareElement(array []any, current, next Segment) {
	// If this segment is not the final one and doesn't exist we need to
	// create it for our next segment.
	if array[current.Index] == nil {
		switch next.Type {
		case SegmentIndex:
			array[current.Index] = make([]any, next.Index+1)
		case SegmentField:
			array[current.Index] = make(map[string]any)
		}
		return
	}

	// If our next segment indexes an array that exists in our current segment's
	// element we must ensure the array is long enough to set the next segment.
	if next.Type != SegmentIndex {
		return
```

```
        }

        na, ok := array[current.Index].([]any)
        if !ok {
                return
        }

        if int(next.Index) < len(na) {
                return
        }

        array[current.Index] = append(na, make([]any, int(next.Index)-len(na)+1)...)
}

func prepareField(object map[string]any, current, next Segment) {
        // If this segment is not the final one and doesn't exist we need to
        // create it for our next segment.
        if _, ok := object[current.Field]; !ok {
                switch next.Type {
                case SegmentIndex:
                        object[current.Field] = make([]any, next.Index+1)
                case SegmentField:
                        object[current.Field] = make(map[string]any)
                }
                return
        }

        // If our next segment indexes an array that exists in our current segment's
        // field we must ensure the array is long enough to set the next segment.
        if next.Type != SegmentIndex {
                return
        }

        na, ok := object[current.Field].([]any)
        if !ok {
                return
        }

        if int(next.Index) < len(na) {
                return
        }

        object[current.Field] = append(na, make([]any, int(next.Index)-len(na)+1)...)
}
```

Stack trace

```
fatal error: runtime: out of memory

runtime stack:
runtime.throw({0x26766c4?, 0x2430?})
        runtime/panic.go:1047 +0x5f fp=0x7ffe30995c50 sp=0x7ffe30995c20 pc=0x5a4edf
runtime.sysMapOS(0x10c001400000, 0x472800000?)
        runtime/mem_linux.go:187 +0x11b fp=0x7ffe30995c98 sp=0x7ffe30995c50 pc=0x586adb
runtime.sysMap(0x3fdfca0?, 0x7ff0e0300000?, 0x599fa0?)
        runtime/mem.go:142 +0x35 fp=0x7ffe30995cc8 sp=0x7ffe30995c98 pc=0x5864b5
```

```
runtime.(*mheap).grow(0x3fdfca0, 0x239374?)
        runtime/mheap.go:1459 +0x23d fp=0x7ffe30995d38 sp=0x7ffe30995cc8 pc=0x59705d
runtime.(*mheap).allocSpan(0x3fdfca0, 0x239374, 0x0, 0x0)
        runtime/mheap.go:1191 +0x1be fp=0x7ffe30995dd0 sp=0x7ffe30995d38 pc=0x59679e
runtime.(*mheap).alloc.func1()
        runtime/mheap.go:910 +0x65 fp=0x7ffe30995e18 sp=0x7ffe30995dd0 pc=0x596225
runtime.systemstack()
        runtime/asm_amd64.s:492 +0x46 fp=0x7ffe30995e20 sp=0x7ffe30995e18 pc=0x5d3266

goroutine 17 [running, locked to thread]:
runtime.systemstack_switch()
        runtime/asm_amd64.s:459 fp=0x10c000da3628 sp=0x10c000da3620 pc=0x5d3200
runtime.(*mheap).alloc(0x4726e8000?, 0x239374?, 0x0?)
        runtime/mheap.go:904 +0x65 fp=0x10c000da3670 sp=0x10c000da3628 pc=0x596165
runtime.(*mcache).allocLarge(0x455960?, 0x4726e7120, 0x0)
        runtime/mcache.go:233 +0x85 fp=0x10c000da36c0 sp=0x10c000da3670 pc=0x585445
runtime.mallocgc(0x4726e7120, 0x29c9420, 0x1)
        runtime/malloc.go:1029 +0x57e fp=0x10c000da3738 sp=0x10c000da36c0 pc=0x57b69e
runtime.makeslice(0x29df3a0?, 0x10c000b4f590?, 0x10c000b38900?)
        runtime/slice.go:103 +0x52 fp=0x10c000da3760 sp=0x10c000da3738 pc=0x5bb912
github.com/crossplane/crossplane-runtime/pkg/fieldpath.prepareField(0x10c000b5e1f8?, {0x4?,
{0x10c000b38900?, 0x2951920?}, 0x10c000b50fa0?}, {0x10c000da3930?, {0x0?, 0x5742d8?},
0x10c000c0c1e0?})

github.com/crossplane/crossplane-runtime@v0.19.0-rc.0.0.20221114195150-65044f043902/pkg/fieldpath/pav
ed.go:426 +0x59e fp=0x10c000da3808 sp=0x10c000da3760 pc=0x25f7f3e
github.com/crossplane/crossplane-runtime/pkg/fieldpath.(*Paved).setValue(0x10c0004acd60,
{0x10c000497180, 0x3, 0x4}, {0x0, 0x0})

github.com/crossplane/crossplane-runtime@v0.19.0-rc.0.0.20221114195150-65044f043902/pkg/fieldpath/pav
ed.go:381 +0x6a5 fp=0x10c000da3970 sp=0x10c000da3808 pc=0x25f6de5
github.com/crossplane/crossplane-runtime/pkg/fieldpath.(*Paved).SetValue(0x0?, {0x10c000b38900, 0x60},
{0x0, 0x0})

github.com/crossplane/crossplane-runtime@v0.19.0-rc.0.0.20221114195150-65044f043902/pkg/fieldpath/pav
ed.go:457 +0x1a5 fp=0x10c000da39f8 sp=0x10c000da3970 pc=0x25f8205
github.com/crossplane/crossplane-runtime/pkg/fieldpath.(*Paved).MergeValue(0x2b73cb0?,
{0x10c000b38900, 0x60}, {0x0, 0x0}, 0x0)

github.com/crossplane/crossplane-runtime@v0.19.0-rc.0.0.20221114195150-65044f043902/pkg/fieldpath/mer
ge.go:47 +0x2eb fp=0x10c000da3a60 sp=0x10c000da39f8 pc=0x25f2a8b
github.com/crossplane/crossplane/internal/controller/apiextensions/composite.patchFieldValueToObject({0x10
c000b38900, 0x60}, {0x0, 0x0}, {0x2b73cb0?, 0x10c000b58ea0}, 0x0?)
        github.com/crossplane/crossplane/internal/controller/apiextensions/composite/merge.go:96 +0x152
fp=0x10c000da3ad8 sp=0x10c000da3a60 pc=0x261e3b2
github.com/crossplane/crossplane/internal/controller/apiextensions/composite.ApplyFromFieldPathPatch({{0x2
675bf0, 0x16}, 0x10c000b50d40, 0x10c000b4f290, 0x10c000b50d60, 0x10c000b50d70, {0x0, 0x0, 0x0}, 0
x0}, ...)
        github.com/crossplane/crossplane/internal/controller/apiextensions/composite/patches.go:194 +0x62c
fp=0x10c000da3bc0 sp=0x10c000da3ad8 pc=0x261ff8c
github.com/crossplane/crossplane/internal/controller/apiextensions/composite.ApplyToObjects({{0x2675bf0,
0x16}, 0x10c000b50d40, 0x10c000b4f290, 0x10c000b50d60, 0x10c000b50d70, {0x0, 0x0, 0x0}, 0x0}, ...)
        github.com/crossplane/crossplane/internal/controller/apiextensions/composite/patches.go:85 +0xa38
fp=0x10c000da3cd8 sp=0x10c000da3bc0 pc=0x261ef98
github.com/crossplane/crossplane/internal/controller/apiextensions/composite.Apply(...)
        github.com/crossplane/crossplane/internal/controller/apiextensions/composite/patches.go:71
```

```
github.com/crossplane/crossplane/internal/controller/apiextensions/composite.FuzzPatchApply({0x60f0000231d0, 0xa9, 0xa9})
```

Parameters

The three parameters passed to Apply are:

Patch


```
{
  "type": "FromCompositeFieldPath",
  "fromFieldPath": "",
  "combine": {
    "variables": [],
    "strategy": "",
    "string": {
      "fmt": ""
    }
  },
  "toFieldPath":
"\u0000\u0004\u0000\ufffdB\ufffd\ufffd\ufffdterface\u0000\u0003\u0000\u0000\u0000\u0000\
u0000\u0000\u0000\u0000\u0000\u0000`\u0000\u0000\u0000\u0000**X[000000000000000001193731
857]i\u0000\u0001\u001a\tn\u0000\u0000\u0000\u0000\u0000\u0000\u0001\u0000\u0000\u0000\u
0000\u0000\u0000\t\u0000\u0000\u0000\u0000\u0000\u0000\u0000tefac",
  "patchSetName": ""
}
```


Composite


```
{
  "creationTimestamp": null,
  "Policy": null,
  "To": null
}
```


Composed


```
{
  "creationTimestamp": null,
  "Ref": null,
  "To": null
}
```

2: runtime.DefaultUnstructuredConverter.ToUnstructured panics if if a pointer-typed field without omitempty specifier is attempted to be converted

ID	ADA-CROSS-FUZZ-2
Severity	Moderate
Fixed	Yes
Fuzzer	`FuzzPatchApply`
OSS-Fuzz bug tracker: • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=50987	

Description

The `FuzzPatchApply` fuzzer found that if `resource.Composite` with a nil connection details last published time to `Patch.Apply`, subsequent calls to `runtime.DefaultUnstructuredConverter.ToUnstructured` to convert it to an `unstructured.Unstructured` will panic. The issue has been fixed in Crossplane here: https://github.com/crossplane/crossplane/pull/3394 and here: https://github.com/crossplane/crossplane-runtime/pull/361, and an upstream PR has been created here: https://github.com/kubernetes-sigs/structured-merge-diff/issues/229.

Credit to Alper Rifat Ulucinar for his work on triaging this issue.

Stack trace

```
panic: value method k8s.io/apimachinery/pkg/apis/meta/v1.Time.ToUnstructured called using nil *Time pointer
goroutine 17 [running, locked to thread]:
k8s.io/apimachinery/pkg/apis/meta/v1.(*Time).ToUnstructured(0x100000001?)
    <autogenerated>:1 +0x71
sigs.k8s.io/structured-merge-diff/v4/value.TypeReflectCacheEntry.ToUnstructured({0x1, 0x0, 0x0, 0x0, 0x1,
0x0, 0x0, {0x0, 0x0, 0x0}}, ...)
    sigs.k8s.io/structured-merge-diff/v4@v4.2.1/value/reflectcache.go:188 +0xa9a
k8s.io/apimachinery/pkg/runtime.toUnstructured({0x1f3c540?, 0x10c00013b4e0?, 0x0?}, {0x1da6580?,
0x10c0002b85b0?, 0x0?})
    k8s.io/apimachinery@v0.24.0/pkg/runtime/converter.go:655 +0x19a
k8s.io/apimachinery/pkg/runtime.structToUnstructured({0x1de9c00?, 0x10c00013b4e0?, 0x6305b0?},
{0x1da6580?, 0x10c0002b8570?, 0x10c000017668?})
    k8s.io/apimachinery@v0.24.0/pkg/runtime/converter.go:843 +0xb91
k8s.io/apimachinery/pkg/runtime.toUnstructured({0x1de9c00?, 0x10c00013b4e0?, 0xa?}, {0x1da6580?,
0x10c0002b8570?, 0x98?})
    k8s.io/apimachinery@v0.24.0/pkg/runtime/converter.go:692 +0x119f
k8s.io/apimachinery/pkg/runtime.structToUnstructured({0x1edae00?, 0x10c00013b380?, 0x1005c85e5?},
{0x1dbbe40?, 0x10c00012c688?, 0x0?})
    k8s.io/apimachinery@v0.24.0/pkg/runtime/converter.go:843 +0xb91
k8s.io/apimachinery/pkg/runtime.toUnstructured({0x1edae00?, 0x10c00013b380?, 0x10c00013b380?},
{0x1dbbe40?, 0x10c00012c688?, 0x7f89fe5604d0?})
```

```
    k8s.io/apimachinery@v0.24.0/pkg/runtime/converter.go:692 +0x119f
k8s.io/apimachinery/pkg/runtime.(*unstructuredConverter).ToUnstructured(0x3337830, {0x1f3acc0?,
0x10c00013b380})
    k8s.io/apimachinery@v0.24.0/pkg/runtime/converter.go:586 +0x6f6
github.com/crossplane/crossplane-runtime/pkg/fieldpath.PaveObject({0x1f48288?, 0x10c00013b380?})
    github.com/crossplane/crossplane-runtime@v0.18.0-rc.0.0.20220722162506-9ea84ae53615/pkg/fieldpath/p
aved.go:55 +0x52
github.com/crossplane/crossplane/apis/apiextensions/v1.patchFieldValueToObject({0x0, 0x0}, {0x0, 0x0},
{0x1f48288?, 0x10c00013b380}, 0x0?)
    github.com/crossplane/crossplane/apis/apiextensions/v1/composition_patches.go:199 +0x5d
github.com/crossplane/crossplane/apis/apiextensions/v1.(*Patch).applyFromFieldPathPatch(0x10c00003d450,
{0x1f48260?, 0x10c0004677a0?}, {0x1f48288, 0x10c00013b380})
    github.com/crossplane/crossplane/apis/apiextensions/v1/composition_patches.go:253 +0x2e5
github.com/crossplane/crossplane/apis/apiextensions/v1.(*Patch).Apply(0x10c00003d450, {0x1f48260,
0x10c0004677a0}, {0x1f48288, 0x10c00013b380}, {0x0, 0x0, 0x0?})
    github.com/crossplane/crossplane/apis/apiextensions/v1/composition_patches.go:129 +0x4a5
github.com/crossplane/crossplane/apis/apiextensions/v1.FuzzPatchApply({0x603000001900, 0x16, 0x16})
    github.com/crossplane/crossplane/apis/apiextensions/v1/patch_fuzzer.go:84 +0x425
```

3: Missing slice length checks causes out of bounds read

ID	ADA-CROSS-FUZZ-3
Severity	Low
Fixed	Yes
Fuzzer	`FuzzNewCompositionRevision`
OSS-Fuzz bug tracker: • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=53815	

Description

NewCompositionRevision reads the first bytes of the compSpecHash without checking the length first. If the compSpecHash is too short, an out of bounds read panic will occur:

https://github.com/crossplane/crossplane/blob/8c77706ff8b180b3bb4cd43920a8afce85dfe312/internal/controller/apiextensions/composition/revision.go#L31-L53

```
func NewCompositionRevision(c *v1.Composition, revision int64, compSpecHash string)
*v1beta1.CompositionRevision {
	cr := &v1beta1.CompositionRevision{
		ObjectMeta: metav1.ObjectMeta{
			Name: fmt.Sprintf("%s-%s", c.GetName(), compSpecHash[0:7]),
			Labels: map[string]string{
				v1beta1.LabelCompositionName: c.GetName(),
				// We cannot have a label value longer than 63 chars
				//
https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#syntax-and-cha
racter-set
				v1beta1.LabelCompositionHash: compSpecHash[0:63],
			},
		},
		Spec: NewCompositionRevisionSpec(c.Spec, revision),
	}

	ref := meta.TypedReferenceTo(c, v1.CompositionGroupVersionKind)
	meta.AddOwnerReference(cr, meta.AsController(ref))

	for k, v := range c.GetLabels() {
		cr.ObjectMeta.Labels[k] = v
	}

	return cr
}
```

4: Time out in 3rd-party dependency

ID	ADA-CROSS-FUZZ-4
Severity	Moderate
Fixed	Yes
Fuzzer	`FuzzParse`
OSS-Fuzz bug tracker: • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=49192	

Description

`FuzzParse`, which tests a 3rd-party parsing routine, found that a particularly well-crafted string could cause the parser to spend excessive time on a single request invocation.

At the time of the audit, Crossplane uses the `github.com/google/go-containerregistry/pkg/name.ParseReference()` the following places:

- https://github.com/crossplane/crossplane/blob/master/internal/initializer/installer.go
- https://github.com/crossplane/crossplane/blob/master/cmd/crank/update.go
- https://github.com/crossplane/crossplane/blob/master/internal/controller/pkg/manager/revisioner.go
- https://github.com/crossplane/crossplane/blob/master/cmd/crank/install.go
- https://github.com/crossplane/crossplane/blob/master/internal/controller/pkg/revision/imageback.go
- https://github.com/crossplane/crossplane/blob/master/internal/controller/pkg/revision/dependency.go
- https://github.com/crossplane/crossplane/blob/master/internal/controller/pkg/resolver/reconciler.go

Runtime stats

All fuzzers written during this audit were added ad-hoc to Crossplanes OSS-Fuzz integration. OSS-Fuzz ran the fuzzers continuously and reported the issues via email to the maintainers. In this table we present the runtime stats as of the 30th December 2022 for each fuzzer.

#	Name	Times executed	Total runtime hours
1	`FuzzPatchApply`	`11,337,642`	`146.2`
2	`FuzzTransform`	`103,067,031`	`373.2`
3	`FuzzParse`	`181,758,787`	`411.3`
4	`FuzzPropagateConnection`	`538,134,219`	`316.9`
5	`FuzzNewCompositionRevision`	`22,756`	`0.3`
6	`FuzzAsComposition`	`394,622,293`	`365.1`
7	`FuzzPackageRevision`	`458,470,899`	`348.5`
8	`FuzzGCRExtract`	`1,062,095,023`	`373.9`
9	`FuzzParseReference`	`343,394,432`	`332.9`
10	`FuzzForCompositeResource`	`906,398,749`	`309.6`
11	`FuzzForCompositeResourceClaim`	`1,220,542,269`	`375.8`
12	`FuzzFindXpkgInDir`	`294,206,967`	`380.3`
13	`FuzzDag`	`3,050,774,637`	`378.9`

Conclusions and future work

In this audit, Ada Logics initiated and improved crossplanes fuzzing efforts. The efforts resulted in 13 fuzzers, 4 issues found and an OSS-Fuzz integration with support for continuous testing and CI integration.
4 issues is a low amount for a fuzzing audit, and credit goes to the Crossplane team for developing and maintaining a high level of code quality throughout the Crossplane code base. Fuzzing has added value to the Crossplane ecosystem by identifying two vulnerabilities and continued testing by OSS-Fuzz.

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

Maintaining the fuzzers
The fuzzers should keep running without breaking to keep testing the code for harder-to-find bugs and build up the corpus.
Over time, Crossplanes fuzzing suite will improve passively from new features in fuzzing as the industry is working to implement new bug detectors into fuzzing. These will be added in a non-intrusive way, meaning that as long as Crossplanes 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 Crossplane, and it will ensure that the fuzzer is written by the developer itself.

We thank the Linux Foundation for sponsoring this project as well as the team at Crossplane 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.