PRESENTS

Helm Fuzzing Security Audit

In collaboration with the Helm project maintainers and The Linux Foundation

Authors

Adam Korczynski <adam@adalogics.com>
David Korczynski <david@adalogics.com>
Date: 29th March, 2023

Table of contents

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 details the engagement whereby Ada Logics integrated continuous fuzzing into the Helm project. At the beginning of this engagement, Helm was not being fuzzed continuously, and the first step was to build the initial infrastructure to enable continuous fuzzing. Ada Logics did this by integrating Helm into OSS-Fuzz - Google's open source project that runs the fuzzers of critical open source projects continuously. After that had been accomplished, Ada Logics wrote fuzzers covering the Helm code base with an emphasis on exported entrypoints. We added the fuzzers to the CNCF-Fuzzing repository throughout the audit so they could run during the engagement itself. Because of the continuous setup, the fuzzers keep running after the audit has completed.

The fuzzers found 9 issues throughout the audit - of which 4 were assigned CVE's.

Results summarised
38 fuzzers were developed
OSS-Fuzz integration for continuous fuzzing set up
9 bugs were found • 4 nil-dereference • 4 out of memory issues • 1 stack overflow
4 CVEs assigned
All fuzzers are merged into the CNCF-fuzzing repository

Project Summary

Ada Logics auditors

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

Helm maintainers involved in the audit

Name	Title	Email
Martin Hickey	Senior Technical Staff Member	martin.hickey@ie.ibm.com
Matt Farina	Distinguished Engineer	matt@mattfarina.com
Scott Rigby	Software Engineer	scott@r6by.com
Matt Butcher	CEO	matt.butcher@fermyon.com

Assets

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

Helm fuzzing

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

Architecture

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

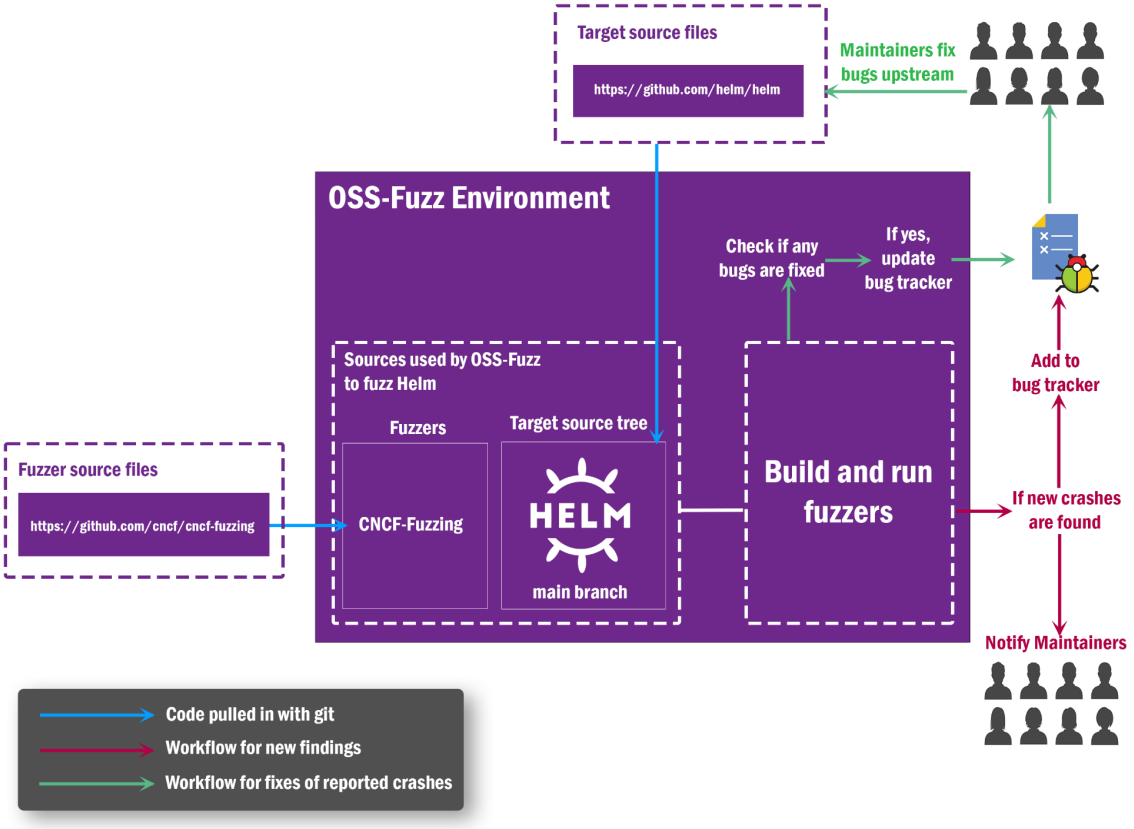

Figure 0.1: Helm's fuzzing architecture

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

This build cycle happens daily and OSS-Fuzz will verify if any existing bugs found by the fuzzers 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 Helm 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 reports are never made public. The Helm maintainers will fix issues upstream, and OSS-Fuzz will pull the latest Helm master branch the next time it performs a fuzz run and verify that a given issue has been fixed.

Fuzzers

In this section we will briefly iterate through the fuzzers that were developed and set up to run continuously. All fuzzers are uploaded to the cncf-fuzzing repository. The fuzzers are being built by OSS-fuzz in the build script.

#	Fuzzer Name	Package	Uploaded to
1	FuzzLintAll	helm.sh/helm/v3/pkg/lint	CNCF-fuzzing
2	FuzzSplitManifests	helm.sh/helm/v3/pkg/releaseutil	CNCF-fuzzing
3	FuzzSortManifests	helm.sh/helm/v3/pkg/releaseutil	CNCF-fuzzing
4	FuzzStorage	helm.sh/helm/v3/pkg/storage	CNCF-fuzzing
5	FuzzNewFromFiles	helm.sh/helm/v3/pkg/provenance	CNCF-fuzzing
6	FuzzParseMessageBlock	helm.sh/helm/v3/pkg/provenance	CNCF-fuzzing
7	FuzzMessageBlock	helm.sh/helm/v3/pkg/provenance	CNCF-fuzzing
8	FuzzKubeClient	helm.sh/helm/v3/pkg/kube	CNCF-fuzzing
9	FuzzGetTagMatchingVersionOrConstraint	helm.sh/helm/v3/pkg/registry	CNCF-fuzzing
10	FuzzparseReference	helm.sh/helm/v3/pkg/registry	CNCF-fuzzing
11	FuzzFindPlugins	helm.sh/helm/v3/pkg/plugin	CNCF-fuzzing
12	FuzzLoadAll	helm.sh/helm/v3/pkg/plugin	CNCF-fuzzing
13	FuzzfixLongPath	helm.sh/helm/v3/internal/third_party/dep/fs	CNCF-fuzzing
14	Fuzz_fixLongPath	helm.sh/helm/v3/internal/third_party/dep/fs	CNCF-fuzzing
15	FuzzSqlDriver	helm.sh/helm/v3/pkg/storage/driver	CNCF-fuzzing
16	FuzzRecords	helm.sh/helm/v3/pkg/storage/driver	CNCF-fuzzing
17	FuzzSecrets	helm.sh/helm/v3/pkg/storage/driver	CNCF-fuzzing
18	FuzzMemory	helm.sh/helm/v3/pkg/storage/driver	CNCF-fuzzing
19	FuzzCfgmaps	helm.sh/helm/v3/pkg/storage/driver	CNCF-fuzzing
20	FuzzMetadataValidate	helm.sh/helm/v3/pkg/chart	CNCF-fuzzing
21	FuzzDependencyValidate	helm.sh/helm/v3/pkg/chart	CNCF-fuzzing
22	FuzzEngineRender	helm.sh/helm/v3/pkg/engine	CNCF-fuzzing
23	FuzzActionRun	helm.sh/helm/v3/pkg/action	CNCF-fuzzing
24	FuzzShowRun	helm.sh/helm/v3/pkg/action	CNCF-fuzzing

25	FuzzDependencyList	helm.sh/helm/v3/pkg/action	CNCF-fuzzing
26	FuzzActionList	helm.sh/helm/v3/pkg/action	CNCF-fuzzing
27	FuzzLoadDir	helm.sh/helm/v3/pkg/chart/loader	CNCF-fuzzing
28	FuzzProcessDependencies	helm.sh/helm/v3/pkg/chartutil	CNCF-fuzzing
29	FuzzIsChartDir	helm.sh/helm/v3/pkg/chartutil	CNCF-fuzzing
30	FuzzExpandFile	helm.sh/helm/v3/pkg/chartutil	CNCF-fuzzing
31	FuzzCreateFrom	helm.sh/helm/v3/pkg/chartutil	CNCF-fuzzing
32	FuzzIndex	helm.sh/helm/v3/pkg/repo	CNCF-fuzzing
33	FuzzIndexDirectory	helm.sh/helm/v3/pkg/repo	CNCF-fuzzing
34	FuzzChartRepositoryLoad	helm.sh/helm/v3/pkg/repo	CNCF-fuzzing
35	FuzzRepoFileUtils	helm.sh/helm/v3/pkg/repo	CNCF-fuzzing
36	FuzzWriteFile	helm.sh/helm/v3/pkg/repo	CNCF-fuzzing
37	FuzzIgnoreParse	helm.sh/helm/v3/internal/ignore	CNCF-fuzzing
38	FuzzStrvalsParse	helm.sh/helm/v3/pkg/strvals	CNCF-fuzzing

Rundown of fuzzers

FuzzLintAll
Creates temporary directory with pseudo-random directories and files and lints the directory contents with `helm.sh/helm/v3/pkg/lint.All()`.

FuzzSplitManifests
Tests `helm.sh/helm/v3/pkg/releaseutil.SplitManifests()` with a pseudo-random string.

FuzzSortManifests
Tests `helm.sh/helm/v3/pkg/releaseutil.SortManifests()` with a pseudo-random files map.

FuzzStorage
Sets up a storage backend and tests the storage APIs like `Create`, `Update` and `Delete`.

FuzzNewFromFiles
Creates a new `Signatory` from pseudo-randomized files.

FuzzParseMessageBlock
Tests `helm.sh/helm/v3/pkg/provenance.parseMessageBlock()` with a pseudo-random byte-slice.

FuzzMessageBlock
Tests `helm.sh/helm/v3/pkg/provenance.parseMessageBlock()` with a directory containing pseudo-random files and directories.

FuzzKubeClient
Sets up a kube client and tests its `Build` and `Update` methods.

FuzzGetTagMatchingVersionOrConstraint
Tests `helm.sh/helm/v3/pkg/registry.GetTagMatchingVersionOrConstraint()` with a pseudo-randomized slice of tags and a pseudo-randomized version string.

FuzzparseReference
Tests `helm.sh/helm/v3/pkg/registry.parseReference()` with a pseudo-random string.

FuzzFindPlugins
Tests `helm.sh/helm/v3/pkg/plugin.FindPlugins()` with a directory containing pseudo-random files and directories.

FuzzLoadAll
Tests `helm.sh/helm/v3/pkg/plugin.LoadAll()` with a directory containing pseudo-random files and directories.

FuzzfixLongPath
Tests `helm.sh/helm/v3/internal/third_party/dep/fs.copyFile()` with pseudo-randomized file contents.

Fuzz_fixLongPath
Tests `helm.sh/helm/v3/internal/third_party/dep/fs.fixLongPath()` with a pseudo-random string.

FuzzSqlDriver
Tests the SQL Driver and its methods. A mock SQL database is set up and `Create`, `Update`, `Query`, `Delete` are called in pseudo-random order with pseudo-random data.

FuzzRecords
Tests `helm.sh/helm/v3/pkg/storage/driver.record` and its methods in pseudo-random order with pseudo-random data.

FuzzSecrets

Tests `helm.sh/helm/v3/pkg/storage/driver.Secrets` and its methods in pseudo-random order with pseudo-random data.

FuzzMemory
Tests the memory driver and its methods in pseudo-random order with pseudo-random data.

FuzzCfgmaps
Tests `helm.sh/helm/v3/pkg/storage/driver.ConfigMaps` and its methods in pseudo-random order with pseudo-random data.

FuzzMetadataValidate
Tests the validation of `helm.sh/helm/v3/pkg/chart.Metadata`.

FuzzDependencyValidate
Tests the validation of `helm.sh/helm/v3/pkg/chart.Dependency`.

FuzzEngineRender
Passes a pseudo-randomized chart and values to `helm.sh/helm/v3/pkg/engine.Render`.

FuzzActionRun
Runs an upgrade action with a pseudo-randomized `helm.sh/helm/v3/pkg/chart.Chart`.

FuzzShowRun
Creates a `helm.sh/helm/v3/pkg/action.Show` and runs it with a pseudo-randomized `helm.sh/helm/v3/pkg/chart.Chart`.

FuzzDependencyList
Tests `helm.sh/helm/v3/pkg/action.*Depency.List()` with a `chartpath` containing pseudo-random files and directories.

FuzzActionList
Creates a helm.sh/helm/v3/pkg/action.List, creates two releases and runs it.

FuzzLoadDir
Tests `helm.sh/helm/v3/pkg/loader.LoadDir()` with a directory containing pseudo-random files and directories.

FuzzProcessDependencies
Tests `helm.sh/helm/v3/pkg/chartutil.ProcessDependencies()` with a pseudo-randomized `chart` and `Values`.

FuzzIsChartDir

Tests `helm.sh/helm/v3/pkg/chartutil.IsChartDir()` with a pseudo-randomized chart dir.

FuzzExpandFile
Tests `helm.sh/helm/v3/pkg/chartutil.ExpandFile()` by expanding a file containing pseudo-random data.

FuzzCreateFrom
Attempts to create a chart from a source directory containing pseudo-randomized files and directories.

FuzzIndex
Creates two `IndexFile` objects and merges them.

FuzzIndexDirectory
Creates an `IndexFile` object from a directory containing pseudo-random files and directories.

FuzzChartRepositoryLoad
Loads a `ChartRepository` containing pseudo-random files and directories.

FuzzRepoFileUtils
Tests the `Add`, `Update`, `Has`, `Get`, `Remove` methods of `helm.sh/helm/v3/pkg/chart.Chart`.

FuzzWriteFile
Creates an index file with pseudo-random contents and it to a new file.

FuzzIgnoreParse
Parses pseudo-random rules file contents.

FuzzStrvalsParse
Tests the strvals parser with a pseudo-random string.

Findings

The fuzzers found 9 crashes during the audit, which we list here. In this section we go into depth with each of the issues.

#	Type	ID	Fixed
1	Denial of service through string value parsing	ADA-helm-22-01	Yes
2	Nil-dereference in 3rd party dependency	ADA-helm-22-02	No
3	Excessive resource consumption by helm.sh/helm/v3/pkg/engine.Render	ADA-helm-22-03	Yes
4	Stack-overflow vulnerability causing Denial-of-Service	ADA-helm-22-04	Yes
5	Well-crafted index file can cause Denial-of-Service	ADA-helm-22-05	Yes
6	Well-crafted schema file can cause Denial-of-Service	ADA-helm-22-06	Yes
7	Nil-dereference in `helm show`	ADA-helm-22-07	Yes
8	Nil-dereference in dependency processing	ADA-helm-22-08	Yes
9	Nil-dereference in dependency processing	ADA-helm-22-09	Yes

Of the found crashes, 4 were assigned CVE's. These were:

#	Issue	CVE ID	GHSA
1	Issue 1: Denial of service through string value parsing	CVE-2022-36055	GHSA-7hfp-qfw3-5jxh
2	Issue 4: Stack-overflow vulnerability causing Denial-of-Service	CVE-2022-23524	GHSA-6rx9-889q-vv2r
3	Issue 5: Well-crafted index file can cause Denial-of-Service	CVE-2022-23525	GHSA-53c4-hhmh-vw5q
4	Issue 6: Well-crafted schema file can cause Denial-of-Service	CVE-2022-23526	GHSA-67fx-wx78-jx33

Issue 1: Denial of service through string value parsing

Source	helm.sh/helm/v3/pkg/strvals
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=44734
ID	ADA-helm-22-01
Fixed	Yes

The `strvals` fuzzer found an issue in Helms `strvals` package that could allow a malicious actor to pass a well-crafted string that could cause denial-of-service. The issue was caused by an untrusted user being able to craft a string that could control the amount of memory that Helm would allocate. The root cause was found to be in `helm.sh/helm/v3/pkg/strvals.setIndex()`, where an untrusted user could control the index variable thus allowing untrusted input to specify a high index and cause an out-of-memory:

```
func setIndex(list []interface{}, index int, val interface{}) (l2 []interface{},
err error) {
	defer func() {
		if r := recover(); r != nil {
			err = fmt.Errorf("error processing index %d: %s", index, r)
		}
	}()

	if index < 0 {
		return list, fmt.Errorf("negative %d index not allowed", index)
	}
	if len(list) <= index {
		newlist := make([]interface{}, index+1)
		copy(newlist, list)
		list = newlist
	}
	list[index] = val
	return list, nil
}
```

OSS-Fuzz found that this vulnerability could both crash Helm with a Go out-of-memory panic, and also cause resource exhaustion on the machine which could allow an attack that could deny the machine running Helm.

The issue has been fixed in Helm 3.9.4.
The issue was assigned CVE-2022-36055
Github Advisory: https://github.com/helm/helm/security/advisories/GHSA-7hfp-qfw3-5jxh

Issue 2: Nil-dereference in 3rd party dependency

Source	`github.com/xeipuuv/gojsonschema.isKind()`
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=44967
ID	ADA-helm-22-03
Fixed	No

A well-crafted schema passed to `helm.sh/helm/v3/pkg/chartutil.ValidateAgainstSingleSchema()` will cause a nil-pointer dereference crash in the `github.com/xeipuuv/gojsonschema dependency`.

The stacktrace of the crash:

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x0 pc=0x1f6ce0d]

goroutine 17 [running, locked to thread]:
github.com/xeipuuv/gojsonschema.isKind({0x27d3f40, 0x10c0006d8f70}, {0x10c0009e9658, 0x1, 0x23ca128})
        github.com/xeipuuv/gojsonschema@v1.2.0/utils.go:38 +0xad
github.com/xeipuuv/gojsonschema.parseSchemaURL({0x27d3f40, 0x10c0006d8f70})
        github.com/xeipuuv/gojsonschema@v1.2.0/draft.go:92 +0x6e
github.com/xeipuuv/gojsonschema.(*schemaPool).parseReferences(0x10c000150860, {0x27d3f40,
0x10c0006d8f70}, {0x10c00070ea20, {{0x0, 0x0, 0x0}}, 0x0, 0x0, 0x0, ...}, ...)
        github.com/xeipuuv/gojsonschema@v1.2.0/schemaPool.go:61 +0x111
github.com/xeipuuv/gojsonschema.(*SchemaLoader).Compile(0x10c00000f1b8, {0x2a46d38,
0x10c00000f188})
        github.com/xeipuuv/gojsonschema@v1.2.0/schemaLoader.go:177 +0x331
github.com/xeipuuv/gojsonschema.NewSchema({0x2a46d38, 0x10c00000f188})
        github.com/xeipuuv/gojsonschema@v1.2.0/schema.go:50 +0xef
github.com/xeipuuv/gojsonschema.Validate({0x2a46d38, 0x10c00000f188}, {0x2a46d38, 0x10c00000f1a0})
        github.com/xeipuuv/gojsonschema@v1.2.0/validation.go:41 +0x45
helm.sh/helm/v3/pkg/chartutil.ValidateAgainstSingleSchema(0x0, {0x10c000710800, 0x31, 0x31})
        helm.sh/helm/v3/pkg/chartutil/jsonschema.go:73 +0x209
helm.sh/helm/v3/pkg/chartutil.ValidateAgainstSchema(0x10c000668c80, 0x10c0006ce5a0)
        helm.sh/helm/v3/pkg/chartutil/jsonschema.go:35 +0x98
helm.sh/helm/v3/pkg/chartutil.ToRenderValues(0x10c000668c80, 0x10c000337650, {{0x10c000710b40,
0x34}, {0x0, 0x0}, 0x1, 0x1, 0x0}, 0x44287e0)
        helm.sh/helm/v3/pkg/chartutil/values.go:159 +0x41d
helm.sh/helm/v3/pkg/action.(*Upgrade).prepareUpgrade(0x10c000703b00, {0x10c000710b40,
0x10c00087feb0}, 0x10c000668c80, 0x1)
        helm.sh/helm/v3/pkg/action/upgrade.go:229 +0xb3b
helm.sh/helm/v3/pkg/action.(*Upgrade).RunWithContext(0x10c000703b00, {0x2a46718, 0x10c000144310},
{0x10c000710b40, 0x34}, 0x10c000875e10, 0x10c00087fea0)
        helm.sh/helm/v3/pkg/action/upgrade.go:143 +0x18a
helm.sh/helm/v3/pkg/action.(*Upgrade).Run(0x22fa77b, {0x10c000710b40, 0x0}, 0x0, 0x0)
        helm.sh/helm/v3/pkg/action/upgrade.go:126 +0x4b
helm.sh/helm/v3/pkg/action.FuzzActionRun({0x625000188900, 0x2318, 0x2318})
        helm.sh/helm/v3/pkg/action/action_fuzzer.go:68 +0x305
main.LLVMFuzzerTestOneInput(...)
```

```
./main.73520297.go:21
```

The crash happens on the highlighted line below:

https://github.com/xeipuuv/gojsonschema/blob/master/utils.go

```
func isKind(what interface{}, kinds ...reflect.Kind) bool {
	target := what
	if isJSONNumber(what) {
		// JSON Numbers are strings!
		target = *mustBeNumber(what)
	}
	targetKind := reflect.ValueOf(target).Kind()
	for _, kind := range kinds {
		if targetKind == kind {
			return true
		}
	}
	return false
}
```

Mitigation

The issue is functional and not security-relevant and will be addressed in public.

Issue 3: Excessive resource consumption by helm.sh/helm/v3/pkg/engine.Render

Source	helm.sh/helm/v3/pkg/engine.Render
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=44996
ID	ADA-helm-22-03
Fixed	Yes

A well crafted chart passed to `helm.sh/helm/v3/pkg/engine.Render()` can cause Helm to consume excessive memory.

OSS-fuzz allocates 2560mb when running the fuzzers, and a single test case consumed 2650mb causing Helm to run out of memory.

The test case can be downloaded in the issue link. Steps to reproduce this crash can be found here: https://google.github.io/oss-fuzz/advanced-topics/reproducing

Issue 4: Stack-overflow vulnerability causing Denial-of-Service

Source	helm.sh/helm/v3/pkg/strvals
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=51774
ID	ADA-helm-22-04
Fixed	Yes

A fuzzer found a stack-overflow in Helms strvals parser that could allow an attacker to pass a malicious string causing denial of service. The issue occurred due to unbounded recursion allowing a string to cause Helm to invoke the same API an unlimited amount of times. The fix was to introduce a limit to the number of times the vulnerable, recursive function could call itself.

The issue was assigned CVE-2022-23524.

Github Advisory: https://github.com/helm/helm/security/advisories/GHSA-6rx9-889q-vv2r

Issue 5: Well-crafted index file can cause Denial-of-Service

Source	helm.sh/helm/v3/pkg/repo
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=54417
ID	ADA-helm-22-05
Fixed	Yes

The fuzzer for Helms repo package found that a well-crafted index file could cause Helm to crash leading to Denial-of-Service. An index file - named index.yaml - is a file in a chart repository that has an index of all charts in the given repository. Typically the index file is created via the `helm repo index` command.

The fuzzer found that a well-crafted index file could cause denial-of-service if a specifically well-crafted index file was passed to Helm. Users that use the repo package in their own applications were vulnerable to this issue. The issue has been fixed in Helm 3.10.3 and later.

The issue was assigned CVE-2022-23525.

Github Advisory:
https://github.com/helm/helm/security/advisories/GHSA-53c4-hhmh-vw5q

Issue 6: Well-crafted schema file can cause Denial-of-Service

Source	helm.sh/helm/v3/pkg/chartutil
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=53014
ID	ADA-helm-22-06
Fixed	Yes

A fuzzer found an issue in Helm's handling of schema files that could lead to Denial of Service. A schema file is an optional feature in Helm that allows chart maintainers to define the structure on the `values.yaml` file in a chart. Chart maintainers can do so by specifying a schema in the `values.schema.json` file in the root of the chart.

Helms chartutil package parses this schema file into Go structures to validate the chart against the schema when users run the following commands:

- `helm install`
- `helm upgrade`
- `helm lint`
- `helm template`

The fuzzer found a vulnerability during chartutils parsing routine of a schema file that could allow a user to craft a malicious schema file that would cause denial-of-service. The vulnerability affected adopters using Helm as a library in their own applications, and it has been fixed in Helm 3.10.3 and later.

The issue has been assigned CVE-2022-23526.

Github Advisory: https://github.com/helm/helm/security/advisories/GHSA-67fx-wx78-jx33

Issue 7: Nil-dereference in `helm show`

Source	helm.sh/helm/v3/pkg/action
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=55078
ID	ADA-helm-22-07
Fixed	Yes

A fuzzer found a nil-dereference in the helm show command. The root cause was in `helm.sh/helm/v3/pkg/action.findReadme()` that loops through a slice of chart files. One of these files could be nil, and when getting its name, Helm would crash with a nil-dereference panic. This would happen on the highlighted line below:

https://github.com/helm/helm/blob/11738dde51447c7bfd1ef0c97cd2bd8fb5e3bfa1/pkg/action/show.go#L153

```
func findReadme(files []*chart.File) (file *chart.File) {
	for _, file := range files {
		for _, n := range readmeFileNames {
			if strings.EqualFold(file.Name, n) {
				return file
			}
		}
	}
	return nil
}
```

The issue has been fixed in https://github.com/helm/helm/pull/11926 by adding a nil-check:

```
func findReadme(files []*chart.File) (file *chart.File) {
	for _, file := range files {
		for _, n := range readmeFileNames {
			if file == nil {
				continue
			}
			if strings.EqualFold(file.Name, n) {
				return file
			}
		}
	}
	return nil
}
```

Issue 8: Nil-dereference in dependency processing

Source	helm.sh/helm/v3/pkg/action
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=55073
ID	ADA-helm-22-08
Fixed	Yes

One of Helm's fuzzers found a nil-dereference in helm.sh/helm/v3/pkg/chartutil.processDependencyEnabled(). The crash happened in a loop through the dependencies of a chart. In this loop, a dependency could be nil, and a missing nil-check would result in a nil-dereference panic. The nil-dereference occurred on the highlighted line below:

https://github.com/helm/helm/blob/c4952c9c8c5fce29635b9795b6070f616a31615c/pkg/chartutil/dependencies.go#L139

```
	for _, req := range c.Metadata.Dependencies {
		if chartDependency := getAliasDependency(c.Dependencies(), req);
chartDependency != nil {
			chartDependencies = append(chartDependencies,
chartDependency)
		}
		if req.Alias != "" {
			req.Name = req.Alias
		}
	}
```

The issue has been fixed in https://github.com/helm/helm/pull/11927 by adding a nil-check:

```
	for _, req := range c.Metadata.Dependencies {
		if req == nil {
			continue
		}
		if chartDependency := getAliasDependency(c.Dependencies(), req);
chartDependency != nil {
			chartDependencies = append(chartDependencies,
chartDependency)
		}
		if req.Alias != "" {
			req.Name = req.Alias
		}
	}
```

Issue 9: Nil-dereference in dependency processing

Source	helm.sh/helm/v3/pkg/engine
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=55062
ID	ADA-helm-22-08
Fixed	Yes

A fuzzer found a nil-dereference in helm.sh/helm/v3/pkg/engine.recAllTpls() when retrieving the templates of a chart. The crash happened when looping through the templates of a chart where one of the templates could be nil. When reading the name of the template, a nil-dereference would occur on the highlighted line:

https://github.com/helm/helm/blob/863bc74e5ad090b97f69dcb643be8d969b07e7cf/pkg/engine/engine.go#L392

```
	newParentID := c.ChartFullPath()
	for _, t := range c.Templates {
		if !isTemplateValid(c, t.Name) {
			continue
		}
		templates[path.Join(newParentID, t.Name)] = renderable{
			tpl:      string(t.Data),
			vals:     next,
			basePath: path.Join(newParentID, "templates"),
		}
	}
```

The issue has been fixed in https://github.com/helm/helm/pull/11928 by adding a nil-check:

```
	for _, t := range c.Templates {
		if t == nil {
			continue
		}
		if !isTemplateValid(c, t.Name) {
			continue
		}
		templates[path.Join(newParentID, t.Name)] = renderable{
			tpl:      string(t.Data),
			vals:     next,
			basePath: path.Join(newParentID, "templates"),
		}
	}
```

Conclusions and future work

Short-term advice

1. Create a strategy for where the fuzzers should be maintained. They are now hosted at the cncf-fuzzing repository, however it is recommended for the Helm maintainers to move the fuzzers upstream.
   This is something Ada Logics will be happy to help with. The only information we need from the maintainers is where the fuzzers should be placed in the upstream repository. A list of options:
   a. Fuzzers are placed in the packages that they test.
   b. Fuzzers are placed in a helm/test/fuzzing package
   c. Fuzzers are placed in a separate repository, for example github.com/helm/fuzzing
2. Fuzzing is natively supported from Go 1.18 and onwards, and it may be worthwhile to rewrite the fuzzers to native Go fuzzers and place them in their respective directories similar to how unit tests are managed. OSS-Fuzz is able to handle native Go fuzzers as of a recent PR, so continuous fuzzing will remain supported.
3. Run the fuzzers in Helms CI with CIFuzz.
4. Improve the procedures and expectations among the maintainers to respond to reports by OSS-Fuzz.

Long-term advice

1. Assess which parts of the Helm ecosystem are missing coverage and write fuzzers to cover the missing parts. These fuzzers should run continuously on OSS-Fuzz, and if any bugs are found, they should be triaged and fixed within OSS-fuzz's 90 days disclosure policy.
2. When new code is submitted to Helm that will not be covered by existing fuzzers, make it a routine to include fuzzers that cover this code.

In this engagement we, Ada Logics, developed an extensive fuzzing suite for the Helm project. We integrated the fuzzing suite into the OSS-Fuzz fuzzing service such that all fuzzers are running continuously by OSS-Fuzz indefinitely. A total of 38 fuzzers were developed and a total of 4 crashes were found.

This work was commissioned by the Cloud Native Computing Foundation (CNCF).