PRESENTS

Cilium Fuzzing Audit

In collaboration with the Cilium project maintainers and The Linux Foundation

Authors

Adam Korczynski <adam@adalogics.com>
David Korczynski <david@adalogics.com>
Date: 13th February, 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 Ciliums fuzzing suite. At the time of this engagement, Cilium was already integrated into OSS-Fuzz, and the goal of this fuzzing audit was to build upon this integration and improve the fuzzing efforts in a continuous manner.

The fuzzing audit added fuzzers for a lot of data processing APIs that found a number of different issues in Cilium and its dependencies. None of the issues were considered critical, however, they revealed some issues in Cilium that prompted rewrites and deprecation of some packages throughout the source tree.

Prior to this engagement, Ciliums OSS-Fuzz integration was set up in manner where the fuzzers and the OSS-Fuzz build script were located in Ciliums own repository[2]. In this fuzzing audit, most development of the fuzzers was carried out in the CNCF-Fuzzing repository, https://github.com/cncf/cncf-fuzzing/tree/main/projects/cilium. This allowed the Ada Logics team to make smaller iterations of the fuzzers throughout the audit and avoid imposing the overhead of having the Cilium maintainers review trivial changes to the fuzzers. OSS-Fuzz was instructed to pull the fuzzers from CNCF-Fuzzing in addition to the fuzzers from Ciliums repository.

Results summarised
14 fuzzers developed
All fuzzers added to Ciliums OSS-Fuzz integration
All fuzzers supported by Ciliums CIFuzz integration
8 crashes were found. • 5 cases of excessive memory allocation • 1 index out of range • 1 time out • 1 nil-dereference

[2] https://github.com/cilium/cilium/blob/master/test/fuzzing/oss-fuzz-build.sh

Table of Contents

Cilium fuzzing

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

Architecture

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

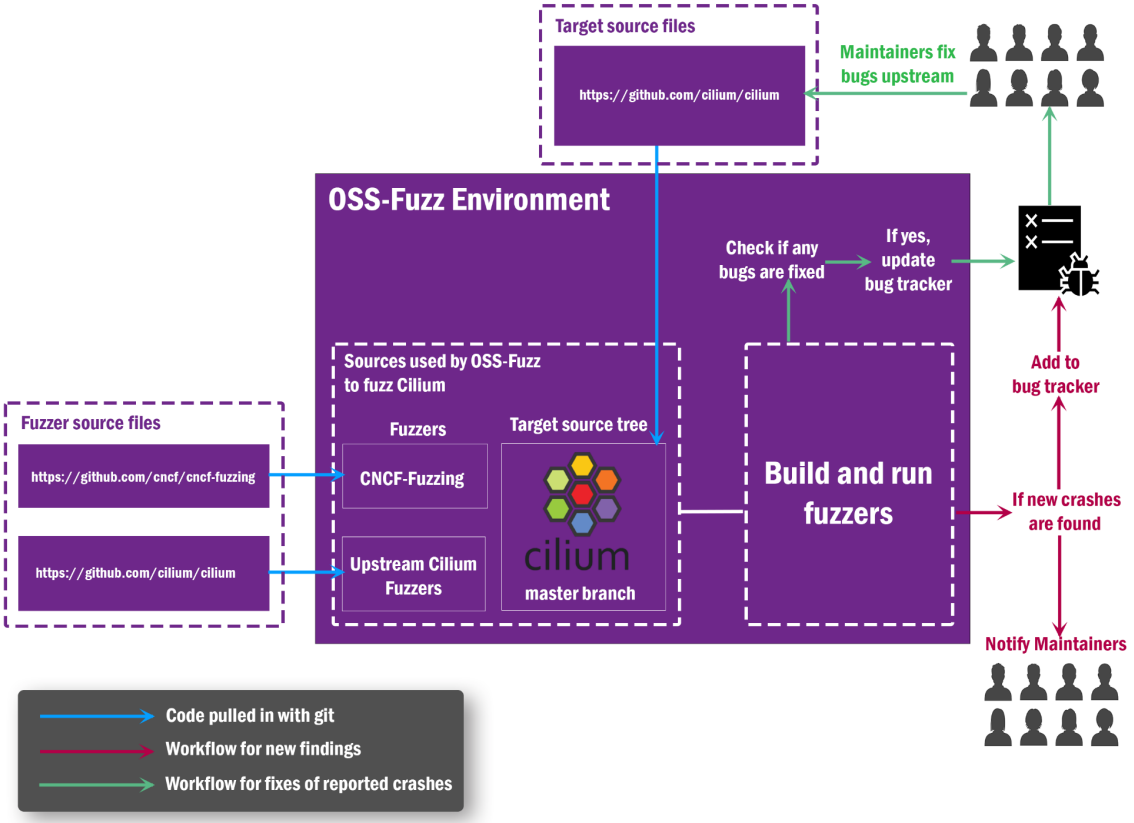

Figure 1.1: Ciliums fuzzing architecture

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

Cilium Fuzzers

In this section we present a highlight of the Cilium fuzzers and which parts of Cilium they test. In total, 14 fuzzers were written during the fuzzing audit.

Overview

#	Name	Package
1	`FuzzLabelsfilterPkg`	`cilium/pkg/labelsfilter`
2	`FuzzDecodeTraceNotify`	`cilium/pkg/monitor`
3	`FuzzFormatEvent`	`cilium/pkg/monitor`
4	`FuzzPayloadEncodeDecode`	`cilium/pkg/monitor/payload`
5	`FuzzElfOpen`	`cilium/pkg/elf`
6	`FuzzElfWrite`	`cilium/pkg/elf`
7	`FuzzMatchpatternValidate`	`cilium/pkg/fqdn/matchpattern`
8	`FuzzMatchpatternValidateWithoutCache`	`cilium/pkg/fqdn/matchpattern`
9	`FuzzParserDecode`	`cilium/pkg/hubble/parser`
10	`FuzzLabelsParse`	`cilium/pkg/k8s/slim/k8s/apis/labels`
11	`FuzzMultipleParsers`	`cilium/proxylib/cassandra`
12	`FuzzConfigParse`	`cilium/pkg/bgp/config`

13	FuzzNewVisibilityPolicy	cilium/pkg/policy
14	FuzzBpf	cilium/pkg/bpf

Target APIs

1: FuzzLabelsfilterPkg
Tests that `ParseLabelPrefixCfg()` and `Filter()` APIs of the `labelsfilter` package.

2: FuzzDecodeTraceNotify
Passes an empty `&TraceNotify{}` and a pseudo-random byte slice to `DecodeTraceNotify()`.

3: FuzzFormatEvent
Creates a pseudo-random `&Payload{}` and passes it to `(m *MonitorFormatter).FormatEvent()`.

4: FuzzPayloadEncodeDecode
Decodes a `Payload{}` with a pseudo-random byte slice.

5: FuzzElfOpen
Creates a file with data provided by the fuzzer and opens it by way of `github.com/cilium/cilium/pkg/elf.Open()`.

6: FuzzElfWrite
Creates a new Elf and writes pseudo-random data to it.

7: FuzzMatchpatternValidate
Passes a pseudo-random string to `github.com/cilium/cilium/pkg/fqdn/matchpattern.Validate()`.

8: FuzzMatchpatternValidateWithoutCache
Passes a pseudo-random string to `github.com/cilium/cilium/pkg/fqdn/matchpattern.Validate()`.

9: FuzzParserDecode
Instantiates a Hubble parser. It then creates a `MonitorEvent` and assigns either a `PerfEvent`, `AgentEvent` or a `LostEvent` to the `MonitorEvent's Payload`. Finally the fuzzer calls `github.com/cilium/cilium/pkg/hubble/parser.(p *Parser).Decode()`, passing the `MonitorEvent`.

10: FuzzLabelsParse
Passes a pseudo-random string to `github.com/cilium/cilium/pkg/k8s/slim/k8s/apis/labels.Parse()`.

11: FuzzMultipleParsers
Starts a log server, inserts a pseudo-random policy text, creates a new proxylib connection and calls `OnData()` against the connection. When creating the connection, the fuzzer chooses between the Cassandra, Kafka, R2d2 or Memcache parsers.

12: FuzzConfigParse
Passes the fuzzing testcase to the `bgp` configuration parser.

13: FuzzNewVisibilityPolicy
Creates a `VisibilityPolicy` using the fuzz testcase as the annotation parameter.

14: FuzzBpf
Creates two files, a “bpffFile” and an “elfFile”. Pseudo-random data is written to each file. The fuzzer then calls either `StartBPFFSMigration()` or `FinalizeBPFFSMigration()`. If it calls `FinalizeBPFFSMigration()` it may set the revert argument to true.

Issues found by fuzzers

A total of 8 unique crashes were reported during the audit. All crashes were triaged by the Cilium team who tracked the issues internally. The issues are as follows:

#	Title	Mitigation
1	Cilium Monitor: Out of memory when decoding specific payload data	Close: WontFix
2	Cilium Monitor: index out of range	Close: WontFix
3	Cilium Monitor: Out of memory panic	Close: WontFix
4	Excessive processing time required for rules with long DNS namesFuzzPayloadEncodeDecode	Improve Cilium limits on matchpattern
5	Excessive memory allocation when parsing MetalLB configuration	Deprecate feature
6	Excessive memory usage when loading and writing ELF file	Avoid reading/writing ELFs as part of datapath load
7	Excessive memory consumption when reading bytes of bpf_elf_map	Avoid reading/writing ELFs as part of datapath load
8	Hubble: nil-dereference in three-four parser	Check if nil before reading

Issue 1-3 were closed as WontFix'es. These were true, reproducible crashes, but triaging showed that they did not have impact on real-world use cases.
Public Github issues were created for issue 4, 5 and 6 to fix at a future date.
Issue 7 and 8 were fixed in Cilium.

1: Cilium Monitor: Out of memory when decoding specific payload data

OSS-Fuzz bug tracker:	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=49119
Mitigation:	Close: WontFix
ID:	ADA-CIL-FUZZ-1

Description

A fuzzer found that a well-crafted payload data byte slice could cause excessive memory consumption of the host machine.

OSS-Fuzz took the following steps to trigger the crash. Note that this happened inside the OSS-Fuzz environment which has a maximum of 2560Mb memory available:

```
1   package payload
2
3   import (
4           "testing"
5   )
6
7   func TestPoC(t *testing.T) {
8           data := []byte{251, 0, 99, 255, 255, 6}
9           pl := &Payload{}
10          pl.Decode(data)
11  }
```

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

… which resulted in allocating 3737Mb of memory that OSS-Fuzz reported as an out-of-memory issue:

```
0   ==13== ERROR: libFuzzer: out-of-memory (used: 3336Mb; limit: 2560Mb)
1     To change the out-of-memory limit use -rss_limit_mb=<N>
2
3   Live Heap Allocations: 24124699 bytes in 32 chunks; quarantined: 44415 bytes in 46 chunks; 7488 other chunks; total
4   chunks: 7566; showing top 95% (at most 8 unique contexts)
5   24120888 byte(s) (99%) in 11 allocation(s)
6      #0 0x52ef96 in __interceptor_malloc /src/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.cpp:69:3
7      #1 0x4ad997 in operator new(unsigned long) cxa_noexception.cpp
8      #2 0x458342 in main /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp:20:10
9      #3 0x7f6e83556082 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x24082) (BuildId:
10  1878e6b475720c7c51969e69ab2d276fae6d1dee)
```

Figure 1.2: Stacktrace from running the test case against the Cilium code base.

The function where this issue may be triggered is run from the "cilium monitor" command, which may be invoked inside the Cilium container by a privileged user. Cilium exercises control over the generation and processing of these messages, so the likelihood of malformed input is low. The impacted component is not a long-running process so there is no expected availability or observability impact. As a consequence of this, the issue will be closed without a fix.

2: Cilium Monitor: index out of range

OSS-Fuzz bug tracker:	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=49723
Mitigation:	Close: WontFix
ID:	ADA-CIL-FUZZ-2

Description

An index out of range panic in Golangs `gob` package was triggerable from Ciliums payload package when decoding a `Payload` with a particularly well-crafted testcase.

Proof of concept

The following test reproduces the issue. The key lines are 8 and 10, where line 8 contains the payload and line 10 the entry point that causes the panic. The data produced on line 8 corresponds to the data generated by the fuzzer.

```
1   package payload
2
3   import (
4           "testing"
5   )
6
7   func TestPoC(t *testing.T) {
8           data := []byte{18, 127, 255, 2, 0, 248, 127, 255, 255, 255, 255,
9                         255, 255, 255, 255, 255, 25, 67, 36}
10          pl := &Payload{}
11          pl.Decode(data)
12  }
```

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

To reproduce the issue the above test can be run against Cilium main branch commit `b4794d690b5690d70c074bffd1db7593e3938e65` by placing the test inside the `pkg/monitor/payload` directory:

```
1   git clone https://github.com/cilium/cilium
2   cd cilium
3   git checkout b4794d690b5690d70c074bffd1db7593e3938e65
4   cd pkg/monitor/payload
```

Figure 2.2: Extracting the relevant commit of Cilium to reproduce ADA-CIL-FUZZ-2

Create the test as `poc_test.go` with the contents of the test above and then run `go test -run=TestPoC`. You should get the following stacktrace:

```
panic: runtime error: index out of range [-9223372036854775808] [recovered]
        panic: runtime error: index out of range [-9223372036854775808] [recovered]
        panic: runtime error: index out of range [-9223372036854775808]

goroutine 19 [running]:
testing.tRunner.func1.2({0x653040, 0xc00015e048})
        /usr/local/go/src/testing/testing.go:1396 +0x24e
testing.tRunner.func1()
        /usr/local/go/src/testing/testing.go:1399 +0x39f
panic({0x653040, 0xc00015e048})
        /usr/local/go/src/runtime/panic.go:884 +0x212
encoding/gob.catchError(0xc00016c1f0)
        /usr/local/go/src/encoding/gob/error.go:38 +0x6d
panic({0x653040, 0xc00015e048})
        /usr/local/go/src/runtime/panic.go:884 +0x212
encoding/gob.(*Decoder).decodeStruct(0xc00016c180?, 0xc00015c180, {0x655440?, 0xc0001ae0c0?, 0x4ec2ef?})
        /usr/local/go/src/encoding/gob/decode.go:462 +0x2cc
encoding/gob.(*Decoder).decodeValue(0xc00016c180, 0x485f0?, {0x62bc00?, 0xc0001ae0c0?, 0xc00016c198?})
        /usr/local/go/src/encoding/gob/decode.go:1210 +0x24e
encoding/gob.(*Decoder).recvType(0xc00016c180, 0x40)
        /usr/local/go/src/encoding/gob/decoder.go:67 +0x13c
encoding/gob.(*Decoder).decodeTypeSequence(0xc00016c180, 0x0)
        /usr/local/go/src/encoding/gob/decoder.go:164 +0x7b
encoding/gob.(*Decoder).DecodeValue(0xc00016c180, {0x644540?, 0xc000115230?, 0x8?})
        /usr/local/go/src/encoding/gob/decoder.go:225 +0x18f
encoding/gob.(*Decoder).Decode(0xc00016c180, {0x644540?, 0xc000115230?})
        /usr/local/go/src/encoding/gob/decoder.go:202 +0x165
github.com/cilium/cilium/pkg/monitor/payload.(*Payload).DecodeBinary(...)
        /tmp/cilium/pkg/monitor/payload/monitor_payload.go:98
github.com/cilium/cilium/pkg/monitor/payload.(*Payload).ReadBinary(0x5007b4?, {0x6bc800?, 0xc000115260?})
        /tmp/cilium/pkg/monitor/payload/monitor_payload.go:82 +0x3f
github.com/cilium/cilium/pkg/monitor/payload.(*Payload).Decode(...)
        /tmp/cilium/pkg/monitor/payload/monitor_payload.go:65
github.com/cilium/cilium/pkg/monitor/payload.TestPOC(0x0?)
        /tmp/cilium/pkg/monitor/payload/pl_test.go:10 +0xad
```

Figure 2.3: Stacktrace from running the test case against the Cilium code base.

This issue has been closed without a fix for the same reasons as ADA-CIL-FUZZ-1.

3: Cilium Monitor: Out of memory panic

OSS-Fuzz bug tracker	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=53070
Mitigation	Close: WontFix
ID:	ADA-CIL-FUZZ-3

Description

FuzzFormatEvent found a crash when a well-crafted Payload would be passed to `(m *MonitorFormatter).FormatEvent()`.

The vulnerable payload json was: `{"Data":"gvsB/yAgIA==","CPU":32,"Lost":32,"Type":9}`

OSS-Fuzz took the following steps to trigger the crash. Note that this happened inside the OSS-Fuzz environment which has a maximum of 2560Mb memory available:

```
1  package main
2
3  import (
4          "github.com/cilium/cilium/pkg/monitor/format"
5          "github.com/cilium/cilium/pkg/monitor/payload"
6  )
7
8  func main() {
9          pl := &payload.Payload{
10                 Data: []byte("gvsB/yAgIA=="),
11                 CPU:  32,
12                 Lost: uint64(32),
13                 Type: 9,
14         }
15         mf := format.NewMonitorFormatter(0, nil)
16         mf.FormatEvent(pl)
17 }
```

Figure 3.0: Sample program that triggered the out-of-memory issue, inside the OSS-Fuzz environment

… which resulted in allocating 3797Mb of memory that OSS-Fuzz reported as an out-of-memory issue:

```
0  ==13== ERROR: libFuzzer: out-of-memory (used: 3797Mb; limit: 2560Mb)
1    To change the out-of-memory limit use -rss_limit_mb=<N>
2
3  Live Heap Allocations: 24131934 bytes in 41 chunks; quarantined: 142310 bytes in 56 chunks; 9911 other chunks; total
4  chunks: 10008; showing top 95% (at most 8 unique contexts)
5  24120824 byte(s) (99%) in 9 allocation(s)
6      #0 0x52efb6 in __interceptor_malloc /src/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.cpp:69:3
7      #1 0x4ad9b7 in operator new(unsigned long) cxa_noexception.cpp
8      #2 0x458362 in main /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp:20:10
9      #3 0x7f92b5028082 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x24082) (BuildId:
10 1878e6b475720c7c51969e69ab2d276fae6d1dee)
```

Figure 3.1: Stack trace reported by OSS-Fuzz

This issue has been closed without a fix for the same reasons as ADA-CIL-FUZZ-1 and ADA-CIL-FUZZ-2.

4: Excessive processing time required for rules with long DNS names

OSS-Fuzz bug tracker:	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=49019
Mitigation:	Improve Cilium limits on matchpattern
ID:	ADA-CIL-FUZZ-4

Description

A fuzzer found that a well-crafted payload passed to `github.com/cilium/cilium/pkg/fqdn/matchpattern.ValidateWithoutCache()` would cause Cilium to spend excessive time on a single process. The issue was reported by OSS-Fuzz as a time-out from spending 61 seconds on a single invocation of `ValidateWithoutCache()`.

The time-out happens when Cilium passes the testcase onto `regexp.Compile(TESTCASE)`. Cilium does this without checking the length of the input, and a long input string can make Cilium spend excessive time on a single invocation. To trigger the crash, the fuzzer had generated a string longer than 70,000 bytes.

The issue was triaged by the Cilium team, and an issue has been opened here: https://github.com/cilium/cilium/issues/21491

The input to this function is first submitted to the Kubernetes apiserver and stored in a Custom Resource field. This requires a high level of privileges to insert. Furthermore, Kubernetes typically imposes various limits on such fields and on the size of the entire resource objects, so it is possible that it is rejected before it reaches this point. Kubernetes will only then forward the object to Cilium for Cilium to then process this object with the code being fuzzed in this scenario.

Due to these mitigating factors, the Cilium maintainers do not consider this to be likely to occur in a real user environment. That said, improvements can be made in the Cilium tree which is why the above Github issue has been created.

5: Excessive memory allocation when parsing MetalLB configuration

OSS-Fuzz bug tracker	• https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=51786 • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=53059
Mitigation	Deprecate feature
ID:	ADA-CIL-FUZZ-5

Description

A fuzzer that tests the 3rd-party metallb config parser found that it is possible to cause excessive memory consumption of the host machine if a well-crafted config file was being parsed.

The parsing routine failed at 2 different places, one in Cilium itself and one in the 3rd-party library handling the parsing.

The found issues have prompted a discussion around deprecating MetalLB support instead of fixing the issue in the 3rd-party dependency itself:
https://github.com/cilium/cilium/issues/22246

6: Excessive memory usage when loading and writing ELF file

OSS-Fuzz bug tracker:	• https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=51731 • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=52981 • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=53015 • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=53066
Mitigation:	Avoid reading/writing ELFs as part of datapath load
ID:	ADA-CIL-FUZZ-6

Description

A fuzzer that Ciliums reading and writing routines of ELF files could trigger both an out-of-memory panics as well as a time-out from excessive processing time on a single elf file. The root cause is an issue in Golang which is well known to the Golang maintainers. In Golang, it is not considered a security vulnerability issue in Golang itself due to the intended usage of the `elf` package.

To trigger the issue, the fuzzer creates a file containing the test case. It then invokes `github.com/cilium/cilium/pkg/elf.Open()` with the path to the created file. `Open()` reads the file and passes the file contents onto `github.com/cilium/cilium/pkg/elf.NewELF()` which passes the file contents onto `debug/elf.NewFile()` in the standard library where the crash happens.

Cilium issue: https://github.com/cilium/cilium/issues/22245

Significant local privileges are required to invoke this bug, so this is not considered a security concern by the Cilium core team. The Cilium maintainers have longer term plans to remove this code, and this will be addressed as part of that effort.

7: Excessive memory consumption when reading bytes of bpf_elf_map

OSS-Fuzz bug tracker	• https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=48961
Mitigation	Avoid reading/writing ELFs as part of datapath load
ID:	ADA-CIL-FUZZ-7

Description

`github.com/cilium/cilium/pkg/bpf.parseExtra()` parses extra bytes from the end of a `bpf_elf_map` struct. A fuzzer was able to invoke this method by calling `StartBPFFSMigration` and `FinalizeBPFFSMigration`.
The issue was fixed by removing the `parseExtra()` api altogether: https://github.com/cilium/cilium/pull/19159.

8: Hubble: nil-dereference in three-four parser

OSS-Fuzz bug tracker	• https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=48960 • https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=48957
Mitigation	Check if nil before reading
ID:	ADA-CIL-FUZZ-8

Description

Two nil-dereference panics were discovered in the Hubble three-four parser when accessing a field of `(*Parser).linkGetter`. At the time of occurrence, the crash would not be triggerable through any existing code paths of Ciliums and was considered a cosmetic change. The fuzzer created the parser through Ciliums own constructor and passed `nil` for all getters.

cilium/pkg/monitor/datapath_debug.go

```
254  case DbgEncap:
255          return fmt.Sprintf("Encapsulating to node %d (%#x) from seclabel %d",
     n.Arg1, n.Arg1, n.Arg2)
256  case DbgLxcFound:
257          ifname := linkMonitor.Name(n.Arg1)
```

Figure 8.1: Point of failure of ADA-CIL-Fuzz-8

cilium/pkg/monitor/datapath_debug.go

```
664          return nil
665  }
666
667  // if the interface is not found, `name` will be an empty string and thus
668  // omitted in the protobuf message
669  name, _ := p.linkGetter.GetIfNameCached(int(ifIndex))
```

Figure 8.2: Point of failure of ADA-CIL-Fuzz-8

Fix: https://github.com/cilium/cilium/pull/20446

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[3] the amounts of tests executed as well as the total CPU hours devoted:

Name	Total times executed	Total runtime (hours)
FuzzLabelsfilterPkg	280,380,108	1,332.7
FuzzDecodeTraceNotify	69,256,225,259	4,816.5
FuzzFormatEvent	63,743,550	129
FuzzPayloadEncodeDecode	1,689,825	245.2
FuzzElfOpen	72,033,696	224.2
FuzzElfWrite	28,824,726	135
FuzzMatchpatternValidate	9,814,975,456	5,146
FuzzMatchpatternValidateWithoutCache	1,250,359,477	7,492
FuzzParserDecode	59,149,192,095	6,927.5
FuzzLabelsParse	2,607,448,955	13,693.5
FuzzMultipleParsers	74,524	6.9
FuzzConfigParse	1,194,473,623	6,805.3
FuzzNewVisibilityPolicy	8,817,227,390	3,379.4
FuzzBpf	205,251,069	12.8

[3] As per 6th December 2022.

Conclusions and future work

This fuzzing audit added 14 fuzzers to the Cilium projects. A total of 8 issues were found and at the time of this writing 5 of these issues have been fixed where 3 of the issues are declared WontFix. The fuzzers were added to Ciliums OSS-Fuzz integration, so that they continue to test Cilium for hard-to-find bugs as well as new code. OSS-Fuzz will periodically pull the latest master of Cilium and run the fuzzers against that version of the source tree.

As this fuzzing audit concludes, it is important to highlight that fuzzing is a continuous effort and that the fuzzers should continue to run through Ciliums OSS-Fuzz integration. Some bugs may take a long time to find, and to allow the fuzzers to get deep into the code base, and it is imperative that the fuzzers keep running for that purpose.

For future work we recommend the following activities to the Cilium team:

Improve Ciliums testability: It may happen that some fuzzers find false positives that theoretically are bugs but cannot be triggered in any execution path to the part of the code where the crash occurs. This was the case with issue #8 of this fuzzing audit that sparked a discussion about what to do in these cases for Cilium. An excellent point was made by Cilium maintainer Joe Stringer who argued that false positives may be a sign that Cilium should improve its testability: https://github.com/cilium/cilium/pull/20446#discussion_r919120926. This is a great observation that also highlights the importance of continuity in fuzzing; Some crashes required multiple development iterations of both fuzzers and the code base itself in order to fully utilize the capabilities of fuzzing, and continuously improving both sides is important.

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: The Cilium community does a good job in adding unit tests to new code contributions, and we recommend that Cilium makes a policy out of adding fuzzers in addition to unit tests. The overhead for writing fuzzers in addition to unit tests in Go is low, since Go has its own fuzzing engine that makes writing unit tests and fuzzers a similar experience: https://go.dev/security/fuzz/.