ADALOGICS

Fluent Bit Security Audit

Fluent Bit Security Audit Report

Adam Korczynski, David Korczynski

01-08-2025

About Ada Logics

Ada Logics is a software security company founded in Oxford, UK, 2018 and is now based in London. We are a team of pragmatic security engineers and security researchers that work hands-on with code auditing, security automation and security tool development.

We are committed open source contributors and we routinely contribute to state of the art security tooling in the fuzzing domain such as advanced fuzzing tools like Fuzz Introspector and continuous fuzzing with OSS-Fuzz. For example, we have contributed to fuzzing of hundreds of open source projects by way of OSS-Fuzz. We regularly perform security audits of open source software and make our reports publicly available with findings and fixes, and we have audited many of the most widely used cloud native applications.

Ada Logics contributes to solving the challenge of securing the software supply-chain. To this end, we develop the tooling and infrastructure needed for ensuring a secure software development lifecycle, and we deploy these tools to critical software packages. On the tooling and infrastructure side, we contribute to projects such as the OpenSSF Scorecard project as well as the Sigstore projects like SLSA and Cosign.

Ada Logics helps some of the most exposed organisations secure their software, analyse their code and increase security automation and assurance, and if you would like to consider working with us please reach out to us via our website.

We write about our work on our blog and maintain a Youtube channel with educational videos. You can also follow Ada Logics on Linkedin, X.

Ada Logics ltd
71-75 Shelton Street,
WC2H 9JQ London,
United Kingdom

Contents

About Ada Logics **1**

1 Executive Summary **3**

2 Audit contacts **4**

3 Scope **4**

4 Fluent Bit security audit **5**
4.1 Fuzzing improvements across Fluent Bit 6

5 Issues found and fixed **7**
5.1 Use-After Free in Fluent Bit Monkey server 8
5.2 ctraces Multiple Memory Leaks 10
5.3 in_exec_wasi file descriptor leakage 13
5.4 Dereference of uninitialized memory in config reading 14
5.5 Heap-Buffer-Overflow By Way Of flb_cf_meta_property_add 16
5.6 out_s3 Can Leak File Descriptor 17
5.7 config_format can Trigger NULL Dereference From Config String 19
5.8 SEGV in config reading 20
5.9 Null-dereference READ in ctr_decode_opentelemetry_create 22
5.10 Null-dereference READ in ctr_decode_opentelemetry_create 24
5.11 Null-dereference READ in unpack_cmetric 26

6 Conclusions **28**

1 Executive Summary

This report contains the results from a security audit of Fluent Bit carried out by Ada Logics, and the audit was funded by the Cloud Native Computing Foundation.

Fluent Bit (fluentbit.io), is a performant log processor for a variety of operating systems and is a part of graduated CNCF project Fluentd. Fluent Bit is written in the C programming language and handles significant amounts of data. For this reason, fuzzing was a focus point of this security audit. Manual auditing and analysis by way of code analysis tools were carried out during the security audit and Ada logics developed fixes for each of the issues found and reported.

Fluent Bit has previously had two security audits funded by the CNCF, where the reports are available here and here. In each of these audits, fuzzing was part of the techniques used due to the useful nature of it when matched with Fluent Bit. Since the completion of these two audits, Fluent Bit has developed a mature continuous fuzzing set up that is run daily by OSS-Fuzz and fuzzing introspection for Fluent Bit is available here.

This audit reported a total of 11 issues to Fluent Bit, ranging from memory corruption issues e.g. use-after-free and buffer overflow, to less security severe issues such as file descriptor leaks and memory leaks. Throughout the audit, the focus was to find vulnerabilities and propose fixes, and for all found issues Ada Logics developed according fixes.

2 Audit contacts

Contact	Role	Organisation	Email
Adam Korczynski	Auditor	Ada Logics Ltd	adam@adalogics.com
David Korczynski	Auditor	Ada Logics Ltd	david@adalogics.com
Chris Aniszczyk	CTO	CNCF	caniszczyk@linuxfoundation.org
Leonardo Albertovich	Fluent Bit Maintainer	Chronosphere	leonardo.alminana@chronosphere.io
Eduardo Silva Pereira	Fluent Bit Maintainer	Chronosphere	eduardo@chronosphere.io

3 Scope

The audit included the code in the following code repositories:

1. https://github.com/fluent/fluent-bit

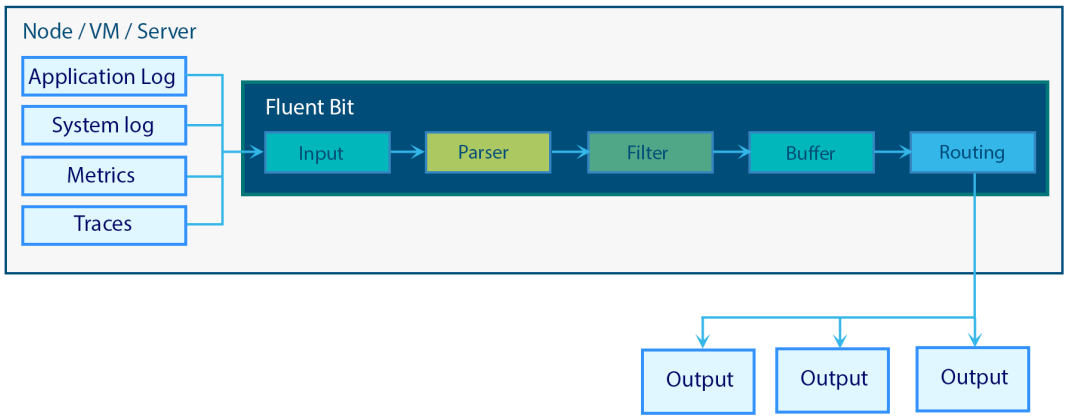
The audit was not fixed to a particular commit; we worked constantly against the latest master branch.

The above repository holds dependencies statically here, which means a lot of projects are in essence in scope here. Several of the issues found and fixed were in projects from inside this lib folder, however, we fixed the issues by submitting patches upstream to ensure wider impact. Links to the patches are below for each issue.

4 Fluent Bit security audit

Fluent Bit is a log processor and forwarder designed to collect, process, and route log data from a wide variety of sources to numerous destinations. It can read logs from local log files, system logs via syslog, journald, or systemd, container log files (such as those generated by Docker or Kubernetes pods), TCP/UDP streams, HTTP endpoints, and Unix sockets. It also supports MQTT and collectd for metrics collection. Once the data is ingested, Fluent Bit can route the processed logs to many output destinations, including databases like Elasticsearch, OpenSearch, InfluxDB, and ClickHouse. It also integrates with cloud logging and observability services such as Amazon CloudWatch, Google Cloud Logging, Azure Monitor, Splunk, and Datadog, and it can send logs to systems like Kafka, HTTP/S endpoints, Fluentd, NATS, and various message queues and storage solutions. This flexibility allows it to act as a connector between a wide array of log producers and consumers in both on-premises and cloud-native environments.

Internally, Fluent Bit is built as a modular and event-driven system. It is written in C, enabling it to maintain low memory usage and high performance, especially in environments with limited resources. The internal architecture is centered around a non-blocking event loop and asynchronous I/O, which lets it handle high volumes of data efficiently across multiple sources and destinations. The log processing workflow moves through a configurable pipeline composed of input plugins, parsers, optional filters, and output plugins as shown below:



Fluent can be - and often is - made to process many different logs. This can be application logs from the users own applications, system logs, metrics generated at the operating system level, traces or other sources. Fluent Bit will often run in the same environment as the log source. This is the common use case pattern in a cloud-native context, where Fluent Bit will run in the same node or pod as the log source. In the diagram above, Fluent Bit outputs the processed data to a destination, which varies depending on the use case. Adopters can also chain Fluent Bit such that Fluent Bit outputs to another Fluent Bit instance before outputting the processed data to a destination. In the context of Fluent Bits security model, we consider the logs untrusted. Logs are often generated by way of input from untrusted users, and often also serve the purpose of detecting attacks. As such, Fluent Bit processes untrusted data from the ‘input’ to ‘routing’. Some Fluent Bit adopters may allow authenticated users to add entries to the log, however, in no case do we consider it acceptable for any content from a log source to compromise Fluent Bits security.

Fluent Bits main attack surface is where an attacker places a payload in a log before Fluent Bit parses it. If an attacker is able to inject specially crafted payloads into logs such as malformed JSON, overly long strings, or exploit code targeting memory handling bugs, they may be able to compromise the Fluent Bit process and escalate their privileges. Attackers can obtain such a position by compromising an application that generates logs, sending crafted packets to a Fluent Bit TCP/UDP endpoint or by abusing container logs with escape sequences or payloads designed to exploit vulnerabilities in log parsers. In many use cases, entirely untrusted users will be able to add log entries to the input logs, and these should not be able to negatively affect a running Fluent Bit instance.

An attacker who successfully crashes the Fluent Bit process may achieve secondary malicious objectives by disrupting

secondary observability and detection mechanisms. For instance, if an application relies on Fluent Bit to forward logs for security analysis like tracking failed login attempts or unusual access patterns, an attacker could crash Fluent Bit to create a visibility gap. During the resulting downtime, malicious behavior could go undetected, allowing the attacker to escalate privileges, exfiltrate data, or move laterally within a network without triggering alerts. In such scenarios, denial-of-service against the logging pipeline becomes a strategic tool for evasion, especially in systems where Fluent Bit acts as a first-line log collection and forwarding agent.

Another possible objective for an attacker is to interfere with Fluent Bit's ability to process logs correctly. This can involve manipulating the structure or content of log messages in ways that cause the Fluent Bit parser or filters to misinterpret data. For example, injecting misleading or malformed content might trick Fluent Bit into associating events with the wrong timestamps, truncating important information, or misrouting logs to the wrong outputs. In some cases, this can create persistent parsing issues that not only affect the current log entry but also influence how subsequent logs are interpreted. Over time, such tampering can degrade the reliability of log data, skew metrics, or pollute downstream storage with corrupted or misleading information. These types of attacks can undermine trust in observability systems and complicate forensic investigations by eroding the integrity of the logs themselves.

Because Fluent Bit is written in C - a memory-unsafe language, it is particularly important to audit the codebase for memory corruption vulnerabilities. Memory safety issues, such as buffer overflows, use-after-frees, and out-of-bounds reads or writes, can be exploited by attackers to achieve serious impact from denial-of-service to remote code execution. Attackers who are able to craft malicious input that exploits flaws in memory handling can potentially compromise the Fluent Bit process and, by extension, the broader system it runs on.

4.1 Fuzzing improvements across Fluent Bit

Fluent Bit was integrated into OSS-Fuzz in April 2020 and its project folder is available here and is amongst the OSS-Fuzz projects with most fuzzing harnesses. Specifically, Fluent Bit has as of this writing 29 fuzzing harnesses running continuously, including the harnesses written as part of this engagement.

Fuzzing is a crucial technique for Fluent Bit since it is ideal for finding vulnerabilities in C code bases and data processing applications. For this reason, fuzzing was a key task of this audit, and fuzzing accounted for finding 8 of the 11 issues reported during this audit.

Throughout this audit, we added 5 new fuzzers for Fluent Bit, as well as extended many of the existing harnesses. Below are key pull requests for these additions:

- add cfl_record_accessor fuzzer
- split cmetrics decoding
- extend ctrace fuzzer
- Extend cmetrics fuzzer
- add input fuzzer
- add config yaml fuzzer
- add mp fuzzer
- add fstore fuzzer

5 Issues found and fixed

Throughout the security audit we found a total of 11 security issues. For each of these issues we supplied a corresponding patch to the relevant GitHub repository. In the following sections we present each of these issues, including references to their patches.

ID	Name	Severity	Status
ADA-FLNTBT-01	Use-after-free in Fluent Bit Monkey server	High	Fixed
ADA-FLNTBT-02	ctraces multiple memory leaks	Low	Fixed
ADA-FLNTBT-03	in_exec_wasi file descriptor leakage	Low	Fixed
ADA-FLNTBT-04	Dereference of uninitialized memory in config reading	Moderate	Fixed
ADA-FLNTBT-05	Heap-buffer-overflow By Way Of flb_cf_meta_property_add	High	Fixed
ADA-FLNTBT-06	out_s3 can leak file descriptor	Low	Fixed
ADA-FLNTBT-07	config_format can trigger NULL dereference from config string	Low	Fixed
ADA-FLNTBT-08	SEGV in config reading	Low	Fixed
ADA-FLNTBT-09	Null-dereference READ in ctr_decode_opentelemetry_create	Moderate	Fixed
ADA-FLNTBT-10	Null-dereference READ in ctr_decode_opentelemetry_create	Moderate	Fixed
ADA-FLNTBT-11	Null-dereference READ in unpack_cmetric	Moderate	Fixed

5.1 Use-After Free in Fluent Bit Monkey server

id	ADA-FLNTBT-01
Severity	High
Status	Fixed
CWE	CWE-416: Use After Free
Fix PR	Schedule: fix Use-After frees

The following use-after-free issue was reported by one of the developed fuzzing harnesses:

```
==12124==ERROR: AddressSanitizer: heap-use-after-free on address 0x60d000006e28 at pc 0x000000efca80 bp 0x7fffc5976590 sp 0x7fffc5976588
WRITE of size 8 at 0x60d000006e28 thread T0
    #0 0xefca7f in __mk_list_del fluent-bit/lib/monkey/include/monkey/mk_core/mk_list.h:141:16
    #1 0xefca7f in mk_list_del fluent-bit/lib/monkey/include/monkey/mk_core/mk_list.h:147:5
    #2 0xefca7f in _mk_event_del fluent-bit/lib/monkey/mk_core/mk_event_epoll.c:175:9
    #3 0xefca7f in _mk_event_timeout_destroy fluent-bit/lib/monkey/mk_core/mk_event_epoll.c:344:5
    #4 0xefca7f in mk_event_timeout_destroy fluent-bit/lib/monkey/mk_core/mk_event.c:164:12
    #5 0x5ca6a3 in flb_sched_timer_cb_disable fluent-bit/src/flb_scheduler.c:502:9
    #6 0x5ca6a3 in flb_sched_timer_destroy fluent-bit/src/flb_scheduler.c:659:5
    #7 0x5ca6a3 in flb_sched_destroy fluent-bit/src/flb_scheduler.c:606:9
    #8 0x59f789 in flb_config_exit fluent-bit/src/flb_config.c:425:5
    #9 0x570c4a in flb_destroy fluent-bit/src/flb_lib.c:233:9
    #10 0x56cbed in LLVMFuzzerTestOneInput fluent-bit/tests/internal/fuzzers/filter_stdout_fuzzer.c:54:5
    #11 0x440843 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:611:15
    #12 0x441365 in fuzzer::Fuzzer::TryDetectingAMemoryLeak(unsigned char const*, unsigned long, bool) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:687:3
    #13 0x42bfee in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:329:8
    #14 0x43184c in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:860:9
    #15 0x45ad82 in main /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp:20:10
    #16 0x7f71e10020b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/libc-start.c:308:16
    #17 0x42216d in _start
```

The problem in this case is that the allocation of a `flb_sched` may contain dangling pointers because of the use of `malloc` here.

```
    sched = flb_malloc(sizeof(struct flb_sched));
    if (!sched) {
        flb_errno();
        return NULL;
    }
```

The fix in this case is simply to ensure the memory used for `sched` is initialized with zero-ed out memory, by using `calloc` instead of malloc.

5.2 ctraces Multiple Memory Leaks

id	ADA-FLNTBT-02
Severity	Low
Status	Fixed
CWE	CWE-401: Missing Release of Memory after Effective Lifetime, CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection')
Fix PR	ctraces: Multiple memory leaks

The following leaks were found by developed fuzzing harnesses:

```
==2080==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 23 byte(s) in 1 object(s) allocated from:
    #0 0x52ef96 in malloc /src/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.
        cpp:69:3
    #1 0x57b2e8 in sds_alloc fluent-bit/lib/cfl/src/cfl_sds.c:47:11
    #2 0x57b2e8 in cfl_sds_create_size fluent-bit/lib/cfl/src/cfl_sds.c:164:12
    #3 0x579327 in ctr_mpack_consume_string_tag fluent-bit/lib/ctraces/src/
        ctr_mpack_utils.c:244:22
    #4 0x57900e in ctr_mpack_consume_string_or_nil_tag fluent-bit/lib/ctraces/src/
        ctr_mpack_utils.c:29:18
    #5 0x575107 in unpack_span_trace_state fluent-bit/lib/ctraces/src/
        ctr_decode_msgpack.c:400:12
    #6 0x57a38c in ctr_mpack_unpack_map fluent-bit/lib/ctraces/src/ctr_mpack_utils.
        c:378:30
    #7 0x574a95 in unpack_span fluent-bit/lib/ctraces/src/ctr_decode_msgpack.c
        :532:12
    #8 0x57a60b in ctr_mpack_unpack_array fluent-bit/lib/ctraces/src/
        ctr_mpack_utils.c:444:18
    #9 0x5745b6 in unpack_scope_span_spans fluent-bit/lib/ctraces/src/
        ctr_decode_msgpack.c:539:12
    #10 0x57a38c in ctr_mpack_unpack_map fluent-bit/lib/ctraces/src/ctr_mpack_utils
        .c:378:30
    #11 0x57432a in unpack_scope_span fluent-bit/lib/ctraces/src/ctr_decode_msgpack
        .c:572:12
    #12 0x57a60b in ctr_mpack_unpack_array fluent-bit/lib/ctraces/src/
        ctr_mpack_utils.c:444:18
    #13 0x572ad6 in unpack_resource_span_scope_spans fluent-bit/lib/ctraces/src/
        ctr_decode_msgpack.c:579:12
    #14 0x57a38c in ctr_mpack_unpack_map fluent-bit/lib/ctraces/src/ctr_mpack_utils
        .c:378:30
    #15 0x5727fb in unpack_resource_span fluent-bit/lib/ctraces/src/
        ctr_decode_msgpack.c:614:12
    #16 0x57a60b in ctr_mpack_unpack_array fluent-bit/lib/ctraces/src/
        ctr_mpack_utils.c:444:18
    #17 0x572696 in unpack_resource_spans fluent-bit/lib/ctraces/src/
        ctr_decode_msgpack.c:621:12
    #18 0x57a38c in ctr_mpack_unpack_map fluent-bit/lib/ctraces/src/ctr_mpack_utils
        .c:378:30
    #19 0x572487 in unpack_context fluent-bit/lib/ctraces/src/ctr_decode_msgpack.c
        :632:12
    #20 0x572487 in ctr_decode_msgpack_create fluent-bit/lib/ctraces/src/
        ctr_decode_msgpack.c:654:14
```

```
        #21 0x569ebc in LLVMFuzzerTestOneInput fluent-bit/tests/internal/fuzzers/
            ctrace_fuzzer.c:11:5
        #22 0x43de03 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned
            long) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:611:15
        #23 0x429562 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long)
             /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:324:6
        #24 0x42ee0c in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const
            *, unsigned long)) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.
            cpp:860:9
        #25 0x458342 in main /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp
            :20:10
        #26 0x7fd73b07c0b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/libc
            -start.c:308:16
```

and

```
    ==1269==ERROR: LeakSanitizer: detected memory leaks

    Direct leak of 8 byte(s) in 1 object(s) allocated from:
        #0 0x52ef96 in malloc /src/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.
            cpp:69:3
        #1 0x56f0d6 in ctr_attributes_create fluent-bit/lib/ctraces/src/ctr_attributes.
            c:26:12
        #2 0x5747cb in unpack_instrumentation_scope_attributes fluent-bit/lib/ctraces/
            src/ctr_decode_msgpack.c:111:22
        #3 0x57a38c in ctr_mpack_unpack_map fluent-bit/lib/ctraces/src/ctr_mpack_utils.
            c:378:30
        #4 0x5744f7 in unpack_scope_span_instrumentation_scope fluent-bit/lib/ctraces/
            src/ctr_decode_msgpack.c:154:12
        #5 0x57a38c in ctr_mpack_unpack_map fluent-bit/lib/ctraces/src/ctr_mpack_utils.
            c:378:30
        #6 0x57432a in unpack_scope_span fluent-bit/lib/ctraces/src/ctr_decode_msgpack.
            c:572:12
        #7 0x57a60b in ctr_mpack_unpack_array fluent-bit/lib/ctraces/src/
            ctr_mpack_utils.c:444:18
        #8 0x572ad6 in unpack_resource_span_scope_spans fluent-bit/lib/ctraces/src/
            ctr_decode_msgpack.c:579:12
        #9 0x57a38c in ctr_mpack_unpack_map fluent-bit/lib/ctraces/src/ctr_mpack_utils.
            c:378:30
        #10 0x5727fb in unpack_resource_span fluent-bit/lib/ctraces/src/
            ctr_decode_msgpack.c:614:12
        #11 0x57a60b in ctr_mpack_unpack_array fluent-bit/lib/ctraces/src/
            ctr_mpack_utils.c:444:18
        #12 0x572696 in unpack_resource_spans fluent-bit/lib/ctraces/src/
            ctr_decode_msgpack.c:621:12
        #13 0x57a38c in ctr_mpack_unpack_map fluent-bit/lib/ctraces/src/ctr_mpack_utils
            .c:378:30
        #14 0x572487 in unpack_context fluent-bit/lib/ctraces/src/ctr_decode_msgpack.c
            :632:12
        #15 0x572487 in ctr_decode_msgpack_create fluent-bit/lib/ctraces/src/
            ctr_decode_msgpack.c:654:14
        #16 0x569ebc in LLVMFuzzerTestOneInput fluent-bit/tests/internal/fuzzers/
            ctrace_fuzzer.c:11:5
        #17 0x43de03 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned
            long) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:611:15
        #18 0x429562 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long)
             /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:324:6
        #19 0x42ee0c in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const
            *, unsigned long)) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.
            cpp:860:9
        #20 0x458342 in main /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp
            :20:10
        #21 0x7fe288b110b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/libc
            -start.c:308:16
```

and

```
1
2       Direct leak of 8 byte(s) in 1 object(s) allocated from:
3           #0 0x52f16e in __interceptor_calloc /src/llvm-project/compiler-rt/lib/asan/
                asan_malloc_linux.cpp:77:3
4           #1 0x578985 in ctr_id_create fluent-bit/lib/ctraces/src/ctr_id.c:66:11
5           #2 0x57720b in ctr_span_set_span_id fluent-bit/lib/ctraces/src/ctr_span.c
                :120:21
6           #3 0x57720b in ctr_span_set_span_id_with_cid fluent-bit/lib/ctraces/src/
                ctr_span.c:131:12
7           #4 0x574e90 in unpack_span_span_id fluent-bit/lib/ctraces/src/
                ctr_decode_msgpack.c:386:13
8           #5 0x57a73c in ctr_mpack_unpack_map fluent-bit/lib/ctraces/src/ctr_mpack_utils.
                c:378:30
9           #6 0x574b05 in unpack_span fluent-bit/lib/ctraces/src/ctr_decode_msgpack.c
                :569:12
10          #7 0x57a9bb in ctr_mpack_unpack_array fluent-bit/lib/ctraces/src/
                ctr_mpack_utils.c:444:18
11          #8 0x574626 in unpack_scope_span_spans fluent-bit/lib/ctraces/src/
                ctr_decode_msgpack.c:576:12
12          #9 0x57a73c in ctr_mpack_unpack_map fluent-bit/lib/ctraces/src/ctr_mpack_utils.
                c:378:30
13          #10 0x57439a in unpack_scope_span fluent-bit/lib/ctraces/src/ctr_decode_msgpack
                .c:609:12
14          #11 0x57a9bb in ctr_mpack_unpack_array fluent-bit/lib/ctraces/src/
                ctr_mpack_utils.c:444:18
15          #12 0x572b46 in unpack_resource_span_scope_spans fluent-bit/lib/ctraces/src/
                ctr_decode_msgpack.c:616:12
16          #13 0x57a73c in ctr_mpack_unpack_map fluent-bit/lib/ctraces/src/ctr_mpack_utils
                .c:378:30
17          #14 0x57286b in unpack_resource_span fluent-bit/lib/ctraces/src/
                ctr_decode_msgpack.c:651:12
18          #15 0x57a9bb in ctr_mpack_unpack_array fluent-bit/lib/ctraces/src/
                ctr_mpack_utils.c:444:18
19          #16 0x572706 in unpack_resource_spans fluent-bit/lib/ctraces/src/
                ctr_decode_msgpack.c:658:12
20          #17 0x57a73c in ctr_mpack_unpack_map fluent-bit/lib/ctraces/src/ctr_mpack_utils
                .c:378:30
21          #18 0x5724f7 in unpack_context fluent-bit/lib/ctraces/src/ctr_decode_msgpack.c
                :669:12
22          #19 0x5724f7 in ctr_decode_msgpack_create fluent-bit/lib/ctraces/src/
                ctr_decode_msgpack.c:691:14
23          #20 0x569f05 in LLVMFuzzerTestOneInput fluent-bit/tests/internal/fuzzers/
                ctrace_fuzzer.c:16:5
24          #21 0x43de03 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned
                long) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:611:15
25          #22 0x429562 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long)
                 /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:324:6
26          #23 0x42ee0c in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const
                *, unsigned long)) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.
                cpp:860:9
27          #24 0x458342 in main /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp
                :20:10
28          #25 0x7f328e7a7082 in __libc_start_main /build/glibc-SzIz7B/glibc-2.31/csu/libc
                -start.c:308:16
```

The fixes to the above PRs are similar in nature and includes ensuring proper checking of return values and additional cleaning up logic. The full PR for mitigating the above leaks is available here

5.3 in_exec_wasi file descriptor leakage

id	ADA-FLNTBT-03
Severity	Low
Status	Fixed
CWE	CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime
Fix PR	in_exec_wasi: fix possible file descriptor leak

A file descriptor leak was found in `in_exec_wasi.c`. The following code:

```
54      FILE *stdoutp = tmpfile();
55
56      /* variables for parser */
57      int parser_ret = -1;
58      void *out_buf = NULL;
59      size_t out_size = 0;
60      struct flb_time out_time;
61
62      if (ctx->oneshot == FLB_TRUE) {
63          ret = flb_pipe_r(ctx->ch_manager[0], &val, sizeof(val));
64          if (ret == -1) {
65              flb_errno();
66              return -1;
67          }
68      }
```

can leak `stdoutp` if `ctx->oneshot` is true and `ret == -1` is true.

The proposed fix:

```
68      if (ctx->oneshot == FLB_TRUE) {
69          ret = flb_pipe_r(ctx->ch_manager[0], &val, sizeof(val));
70          if (ret == -1) {
71              fclose(stdoutp);
72              flb_errno();
73              return -1;
74          }
```

5.4 Dereference of uninitialized memory in config reading

id	ADA-FLNTBT-04
Severity	Moderate
Status	Fixed
CWE	CWE-908: Use of Uninitialized Resource, CWE-457: Use of Uninitialized Variable, CWE-824: Access of Uninitialized Pointer
Fix PR	decode: opentelemetry: fix NULL deref

One of the developed fuzzing harnesses reported the following issue:

```
    AddressSanitizer:DEADLYSIGNAL
=================================================================
==396==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000018 (pc 0
    x5693b908d889 bp 0x7ffde211bc30 sp 0x7ffde211bb40 T0)
==396==The signal is caused by a READ memory access.
==396==Hint: address points to the zero page.
    #0 0x5693b908d889 in decode_data_point_labels fluent-bit/lib/cmetrics/src/
        cmt_decode_opentelemetry.c:364:35
    #1 0x5693b908d047 in decode_numerical_data_point fluent-bit/lib/cmetrics/src/
        cmt_decode_opentelemetry.c:426:18
    #2 0x5693b908d047 in decode_numerical_data_point_list fluent-bit/lib/cmetrics/src/
        cmt_decode_opentelemetry.c:481:18
    #3 0x5693b908a374 in decode_counter_entry fluent-bit/lib/cmetrics/src/
        cmt_decode_opentelemetry.c:727:14
    #4 0x5693b908a374 in decode_metrics_entry fluent-bit/lib/cmetrics/src/
        cmt_decode_opentelemetry.c:861:18
    #5 0x5693b908a374 in decode_scope_metrics_entry fluent-bit/lib/cmetrics/src/
        cmt_decode_opentelemetry.c:1082:18
    #6 0x5693b908a374 in decode_resource_metrics_entry fluent-bit/lib/cmetrics/src/
        cmt_decode_opentelemetry.c:1190:18
    #7 0x5693b908a374 in decode_service_request fluent-bit/lib/cmetrics/src/
        cmt_decode_opentelemetry.c:1250:22
    #8 0x5693b908a374 in cmt_decode_opentelemetry_create fluent-bit/lib/cmetrics/src/
        cmt_decode_opentelemetry.c:1273:18
    #9 0x5693b90869ac in LLVMFuzzerTestOneInput fluent-bit/tests/internal/fuzzers/
        cmetrics_decode_fuzz.c:47:18
    #10 0x5693b8f3b330 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*,
        unsigned long) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:614:13
    #11 0x5693b8f265a5 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned
        long) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:327:6
    #12 0x5693b8f2c03f in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char
        const*, unsigned long)) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.
        cpp:862:9
    #13 0x5693b8f572e2 in main /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp
        :20:10
    #14 0x7c9c43fa5082 in __libc_start_main /build/glibc-LcI20x/glibc-2.31/csu/libc-
        start.c:308:16
    #15 0x5693b8f1e78d in _start
```

This was reported by OSS-Fuzz in https://issues.oss-fuzz.com/issues/42531451.

The problem is that in `cmt_decode_opentelemetry.c` a missing NULL pointer check happens on the following lines:

```
61  attribute = (Opentelemetry__Proto__Common__V1__KeyValue *)
62                  value_index_list[map_label_index];
63
64  if (attribute->value->value_case ==
        OPENTELEMETRY__PROTO__COMMON__V1__ANY_VALUE__VALUE_STRING_VALUE) {
65      result = append_new_metric_label_value(metric, attribute->value->string_value, 0);
66  }
```

The proposed fix:

```
61  attribute = (Opentelemetry__Proto__Common__V1__KeyValue *)
62                  value_index_list[map_label_index];
63
64  if (attribute->value->value_case ==
        OPENTELEMETRY__PROTO__COMMON__V1__ANY_VALUE__VALUE_STRING_VALUE) {
65      result = append_new_metric_label_value(metric, attribute->value->string_value, 0);
66  }
```

5.5 Heap-Buffer-Overflow By Way Of flb_cf_meta_property_add

id	ADA-FLNTBT-05
Severity	High
Status	Fixed
CWE	CWE-122: Heap-based Buffer Overflow
Fix PR	config_format: fix possible heap overflow

The following code in `flb_config_format.c` leads to possible buffer overflows:

```
416      if (meta[0] != '@') {
417          flb_cf_error_set(cf, FLB_CF_ERROR_META_CHAR);
418          return NULL;
419      }
420
421      p = meta;
422      tmp = strchr(p, ' ');
423      xlen = (tmp - p);
424
425      /* create k/v pair */
426      return meta_property_add(cf,
427                               meta + 1, xlen - 1,
428                               meta + xlen + 1, len - xlen - 1);
429  }
```

The return value of `strchr` is not checked for failure. If it's failure then `tmp` will be `0` in the `(tmp-p)` calculation, causing `xlen` to be `p`. `xlen` is later used for copying memory by way of `memcpy` in string creation using `flb_sds_create_len`. This fixes it.

The proposed fix:

```
416      if (meta[0] != '@') {
417          flb_cf_error_set(cf, FLB_CF_ERROR_META_CHAR);
418          return NULL;
419      }
420
421      p = meta;
422      tmp = strchr(p, ' ');
423      if (tmp == NULL) {
424          return NULL;
425      }
426      xlen = (tmp - p);
427
428      /* create k/v pair */
429      return meta_property_add(cf,
430                               meta + 1, xlen - 1,
431                               meta + xlen + 1, len - xlen - 1);
432  }
```

5.6 out_s3 Can Leak File Descriptor

id	ADA-FLNTBT-06
Severity	Low
Status	Fixed
CWE	CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime
Fix PR	out_s3: fix potential file descriptor leak

Two possible file descriptor leaks exist in the below code

```
284 static int read_seq_index(char *seq_index_file, uint64_t *seq_index)
285 {
286     FILE *fp;
287     int ret;
288
289     fp = fopen(seq_index_file, "r");
290     if (fp == NULL) {
291         flb_errno();
292         return -1;
293     }
294
295     ret = fscanf(fp, "%"PRIu64, seq_index);
296     if (ret != 1) {
297         flb_errno();
298         return -1;
299     }
300
301     fclose(fp);
302     return 0;
303 }
304
305 /* Writes index value to metadata file */
306 static int write_seq_index(char *seq_index_file, uint64_t seq_index)
307 {
308     FILE *fp;
309     int ret;
310
311     fp = fopen(seq_index_file, "w+");
312     if (fp == NULL) {
313         flb_errno();
314         return -1;
315     }
316
317     ret = fprintf(fp, "%"PRIu64, seq_index);
318     if (ret < 0) {
319         flb_errno();
320         return -1;
321     }
322
323     fclose(fp);
324     return 0;
325 }
```

If `ret - 1` holds true in `read_seq_index` a file descriptor leak will occur, and, similarly, if `ret < 0` holds true in `write_seq_index` a file descriptor leak will occur.

The proposed fix is to simply close the file descriptor in each of the given cases.

5.7 config_format can Trigger NULL Dereference From Config String

id	ADA-FLNTBT-07
Severity	Low
Status	Fixed
CWE	CWE-476: NULL Pointer Dereference
Fix PR	config_format: yaml: fix null dereference

A NULL-dereference issue exists in `flb_cf_yaml.c` in the following code

```
1288            if (strcmp(state->key, "processors") == 0) {
1289                yaml_error_event(ctx, state, event);
1290                return YAML_FAILURE;
1291            }
1292
1293            if (state->key == NULL) {
1294                flb_error("no key");
1295                return YAML_FAILURE;
1296            }
```

The problem is that the `state->key == NULL` check needs to occur before the `strcmp` check, as otherwise `strcmp` causes a NULL-dereference.

The fix is to switch the conditionals so the NULL-dereference check happens first.

5.8 SEGV in config reading

id	ADA-FLNTBT-08
Severity	Low
Status	Fixed
CWE	CWE-824: Access of Uninitialized Pointer, CWE-457: Use of Uninitialized Variable, CWE-908: Use of Uninitialized Resource
Fix PR	config_format: fix SEGV from missing return check

One of the developed fuzzing harnesses reported the following issue:

```
    AddressSanitizer:DEADLYSIGNAL
=================================================================
==399==ERROR: AddressSanitizer: SEGV on unknown address (pc 0x565e0eb7eaf7 bp 0x7ffd35e0ff70 sp 0x7ffd35e0ff30 T0)
==399==The signal is caused by a READ memory access.
==399==Hint: this fault was caused by a dereference of a high value address (see register values below).  Disassemble the provided pc to learn which register was used.
    #0 0x565e0eb7eaf7 in flb_sds_avail fluent-bit/include/fluent-bit/flb_sds.h:77:25
    #1 0x565e0eb7eaf7 in flb_sds_cat fluent-bit/src/flb_sds.c:126:13
    #2 0x565e0eb7eaf7 in flb_sds_cat_safe fluent-bit/src/flb_sds.c:208:11
    #3 0x565e0eb5a716 in flb_cf_key_translate fluent-bit/src/config_format/flb_config_format.c:74:5
    #4 0x565e0eb5bdd7 in flb_cf_section_property_add fluent-bit/src/config_format/flb_config_format.c:275:11
    #5 0x565e0eb67c91 in consume_event fluent-bit/src/config_format/flb_cf_yaml.c:2341:17
    #6 0x565e0eb6539f in read_config fluent-bit/src/config_format/flb_cf_yaml.c:2892:18
    #7 0x565e0eb645b3 in flb_cf_yaml_create fluent-bit/src/config_format/flb_cf_yaml.c:2973:11
    #8 0x565e0eb57928 in LLVMFuzzerTestOneInput fluent-bit/tests/internal/fuzzers/config_yaml_fuzzer.c:56:10
    #9 0x565e0eb574f9 in ExecuteFilesOnyByOne /src/aflplusplus/utils/aflpp_driver/aflpp_driver.c:255:7
    #10 0x565e0eb572f5 in LLVMFuzzerRunDriver /src/aflplusplus/utils/aflpp_driver/aflpp_driver.c:0
    #11 0x565e0eb56ead in main /src/aflplusplus/utils/aflpp_driver/aflpp_driver.c:311:10
    #12 0x7d2c5b5a8082 in __libc_start_main /build/glibc-LcI20x/glibc-2.31/csu/libc-start.c:308:16
    #13 0x565e0ea7e6ad in _start
```

This was reported by OSS-Fuzz in https://issues.oss-fuzz.com/issues/42530021.

The problem is that in `flb_cf_key_translate` a string is created but is missing a validity check on success. The fuzzer managed to create an input that would cause the string creation to fail, which then later meant bogus memory would be used for a dereference operation.

https://github.com/fluent/fluent-bit/blob/e2cee6a8c0c8a70303937e2ee519ac2f730eacfb/src/config_format/flb_config_format.c#L72-L74 ~~~~ {.c .numberLines startFrom="72"} /* copy content and check if we have underscores / *out = flb_sds_create_size(len* 2); flb_sds_cat_safe(&out, key, len); ~~~~~~

The proposed fix:

```
72      /* copy content and check if we have underscores */
73      out = flb_sds_create_size(len * 2);
74      if (out == NULL) {
75          return NULL;
76      }
77      flb_sds_cat_safe(&out, key, len);
```

5.9 Null-dereference READ in ctr_decode_opentelemetry_create

id	ADA-FLNTBT-09
Severity	Moderate
Status	Fixed
CWE	CWE-476: NULL Pointer Dereference
Fix PR	decode_opentelemetry: fix NULL deref

One of the developed fuzzing harnesses reported the following issue:

```
1  =================================================================
2  ==399==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000018 (pc 0
       x560e24a79262 bp 0x7ffec2362670 sp 0x7ffec2362570 T0)
3  ==399==The signal is caused by a READ memory access.
4  ==399==Hint: address points to the zero page.
5      #0 0x560e24a79262 in resource_set_data fluent-bit/lib/ctraces/src/
           ctr_decode_opentelemetry.c:445
6      #1 0x560e24a79262 in ctr_decode_opentelemetry_create fluent-bit/lib/ctraces/src/
           ctr_decode_opentelemetry.c:555:9
7      #2 0x560e24a6a59d in LLVMFuzzerTestOneInput fluent-bit/tests/internal/fuzzers/
           ctrace_fuzzer.c:26:5
8      #3 0x560e24a6a339 in ExecuteFilesOnyByOne /src/aflplusplus/utils/aflpp_driver/
           aflpp_driver.c:255:7
9      #4 0x560e24a6a135 in LLVMFuzzerRunDriver /src/aflplusplus/utils/aflpp_driver/
           aflpp_driver.c:0
10     #5 0x560e24a69ced in main /src/aflplusplus/utils/aflpp_driver/aflpp_driver.c:311:10
11     #6 0x7b638d18e082 in __libc_start_main /build/glibc-LcI20x/glibc-2.31/csu/libc-
           start.c:308:16
12     #7 0x560e249914ed in _start
```

This was reported by OSS-Fuzz in https://issues.oss-fuzz.com/issues/412744484.

A fix is proposed in https://github.com/fluent/ctraces/pull/70

The problem is that in the below code:

```
524  otel_resource_span = service_request->resource_spans[resource_span_index];
525  if (otel_resource_span == NULL) {
526      opentelemetry__proto__collector__trace__v1__export_trace_service_request__free_unpacked
           (service_request, NULL);
527      ctr_destroy(ctr);
528      return CTR_DECODE_OPENTELEMETRY_INVALID_PAYLOAD;
529  }
530
531  /* resource span */
532  resource_span = ctr_resource_span_create(ctr);
533  ctr_resource_span_set_schema_url(resource_span, otel_resource_span->schema_url);
534
535  /* resource */
536  resource = ctr_resource_span_get_resource(resource_span);
537  resource_set_data(resource, otel_resource_span->resource);
538
539  ctr_resource_set_dropped_attr_count(resource, otel_resource_span->resource->
       dropped_attributes_count);
```

The problem is that `otel_resource_span->resource` may be NULL, and when used in `resource_set_data` it is dereferenced, causing a NULL pointer dereference.

The proposed fix is to check for NULL conditions in the first conditional:

```
524  otel_resource_span = service_request->resource_spans[resource_span_index];
525  if (otel_resource_span == NULL || otel_resource_span->resource == NULL) {
526      opentelemetry__proto__collector__trace__v1__export_trace_service_request__free_unpacked
         (service_request, NULL);
527      ctr_destroy(ctr);
528      return CTR_DECODE_OPENTELEMETRY_INVALID_PAYLOAD;
529  }
530
531  /* resource span */
532  resource_span = ctr_resource_span_create(ctr);
533  ctr_resource_span_set_schema_url(resource_span, otel_resource_span->schema_url);
534
535  /* resource */
536  resource = ctr_resource_span_get_resource(resource_span);
537  resource_set_data(resource, otel_resource_span->resource);
538
539  ctr_resource_set_dropped_attr_count(resource, otel_resource_span->resource->
       dropped_attributes_count);
```

5.10 Null-dereference READ in ctr_decode_opentelemetry_create

id	ADA-FLNTBT-10
Severity	Moderate
Status	Fixed
CWE	CWE-476: NULL Pointer Dereference
Fix PR	decode_opentelemetry: fix NULL deref

One of the developed fuzzing harnesses reported the following issue:

```
    =================================================================
==397==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000018 (pc 0
    x570dc6952d4f bp 0x7ffc19033ee0 sp 0x7ffc19033e80 T0)
==397==The signal is caused by a READ memory access.
==397==Hint: address points to the zero page.
    #0 0x570dc6952d4f in convert_any_value fluent-bit/lib/ctraces/src/
        ctr_decode_opentelemetry.c:302:18
    #1 0x570dc69518de in convert_otel_attrs fluent-bit/lib/ctraces/src/
        ctr_decode_opentelemetry.c:372:18
    #2 0x570dc6951d91 in resource_set_data fluent-bit/lib/ctraces/src/
        ctr_decode_opentelemetry.c:445:18
    #3 0x570dc6951d91 in ctr_decode_opentelemetry_create fluent-bit/lib/ctraces/src/
        ctr_decode_opentelemetry.c:555:9
    #4 0x570dc694697c in LLVMFuzzerTestOneInput fluent-bit/tests/internal/fuzzers/
        ctrace_fuzzer.c:26:5
    #5 0x570dc67fb320 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned
         long) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:614:13
    #6 0x570dc67e6595 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long
        ) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:327:6
    #7 0x570dc67ec02f in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char
        const*, unsigned long)) /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.
        cpp:862:9
    #8 0x570dc68172d2 in main /src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp
        :20:10
    #9 0x785e3a80f082 in __libc_start_main /build/glibc-LcI20x/glibc-2.31/csu/libc-
        start.c:308:16
    #10 0x570dc67de77d in _start
```

This was reported by OSS-Fuzz in https://issues.oss-fuzz.com/issues/414273107.

A fix is proposed in https://github.com/fluent/ctraces/pull/71

The problem is that in the below code:

```
276 static int convert_any_value(struct opentelemetry_decode_value *ctr_val,
277                               opentelemetry_decode_value_type value_type, char *key,
278                               Opentelemetry__Proto__Common__V1__AnyValue *val)
279 {
280     int result;
281
282     switch (val->value_case) {
```

The problem is that `val` may be NULL.

The proposed fix is to check for NULL conditions in the first conditional:

```
276 static int convert_any_value(struct opentelemetry_decode_value *ctr_val,
277                                opentelemetry_decode_value_type value_type, char *key,
278                                Opentelemetry__Proto__Common__V1__AnyValue *val)
279 {
280     int result;
281
282     if (val == NULL) {
283         return -1;
284     }
285     switch (val->value_case) {
```

5.11 Null-dereference READ in unpack_cmetric

id	ADA-FLNTBT-11
Severity	Moderate
Status	Fixed
CWE	CWE-476: NULL Pointer Dereference
Fix PR	unpack_metric: fix NULL deref

One of the developed fuzzing harnesses reported the following issue:

```
    =================================================================
==405==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000000 (pc 0
    x5ae0d9347bc0 bp 0x7ffed5c37660 sp 0x7ffed5c375c0 T0)
==405==The signal is caused by a READ memory access.
==405==Hint: address points to the zero page.
    #0 0x5ae0d9347bc0 in unpack_metric fluent-bit/lib/cmetrics/src/cmt_decode_msgpack.c
        :672:59
    #1 0x5ae0d9347bc0 in unpack_metric_array_entry fluent-bit/lib/cmetrics/src/
        cmt_decode_msgpack.c:737:14
    #2 0x5ae0d934a43c in cmt_mpack_unpack_array fluent-bit/lib/cmetrics/src/
        cmt_mpack_utils.c:255:18
    #3 0x5ae0d9349ffb in cmt_mpack_unpack_map fluent-bit/lib/cmetrics/src/
        cmt_mpack_utils.c:189:30
    #4 0x5ae0d9344967 in unpack_basic_type fluent-bit/lib/cmetrics/src/
        cmt_decode_msgpack.c:1091:14
    #5 0x5ae0d9344967 in unpack_basic_type_entry fluent-bit/lib/cmetrics/src/
        cmt_decode_msgpack.c:1309:14
    #6 0x5ae0d934a43c in cmt_mpack_unpack_array fluent-bit/lib/cmetrics/src/
        cmt_mpack_utils.c:255:18
    #7 0x5ae0d9349ffb in cmt_mpack_unpack_map fluent-bit/lib/cmetrics/src/
        cmt_mpack_utils.c:189:30
    #8 0x5ae0d9342459 in unpack_context fluent-bit/lib/cmetrics/src/cmt_decode_msgpack.
        c:1442:12
    #9 0x5ae0d9342459 in cmt_decode_msgpack_create fluent-bit/lib/cmetrics/src/
        cmt_decode_msgpack.c:1476:14
    #10 0x5ae0d933a4dd in LLVMFuzzerTestOneInput fluent-bit/tests/internal/fuzzers/
        cmetrics_decode_fuzz.c:53:18
    #11 0x5ae0d933a329 in ExecuteFilesOnyByOne /src/aflplusplus/utils/aflpp_driver/
        aflpp_driver.c:255:7
    #12 0x5ae0d933a125 in LLVMFuzzerRunDriver /src/aflplusplus/utils/aflpp_driver/
        aflpp_driver.c:0
    #13 0x5ae0d9339cdd in main /src/aflplusplus/utils/aflpp_driver/aflpp_driver.c
        :311:10
    #14 0x7bb394535082 in __libc_start_main /build/glibc-LcI20x/glibc-2.31/csu/libc-
        start.c:308:16
    #15 0x5ae0d92614dd in _start
```

This was reported by OSS-Fuzz in https://issues.oss-fuzz.com/issues/429003372.

A fix is proposed in https://github.com/fluent/cmetrics/pull/237

The problem is that in the below code:

```
669     if (decode_context->map->type == CMT_HISTOGRAM) {
670         histogram = decode_context->map->parent;
671         metric->hist_buckets = calloc(histogram->buckets->count + 1, sizeof(uint64_t));
```

```
672
673         if (metric->hist_buckets == NULL) {
674             cmt_errno();
675
676             free(metric);
677
678             return CMT_DECODE_MSGPACK_ALLOCATION_ERROR;
679         }
680     }
```

The problem is that `histogram->buckets` may be NULL.

The proposed fix is to check for NULL conditions in the first conditional:

```
669     if (decode_context->map->type == CMT_HISTOGRAM) {
670         histogram = decode_context->map->parent;
671         if (histogram == NULL || histogram->buckets == NULL) {
672             free(metric);
673             cmt_errno();
674             return CMT_DECODE_MSGPACK_ALLOCATION_ERROR;
675         }
676         metric->hist_buckets = calloc(histogram->buckets->count + 1, sizeof(uint64_t));
677
678         if (metric->hist_buckets == NULL) {
679             cmt_errno();
680
681             free(metric);
682
683             return CMT_DECODE_MSGPACK_ALLOCATION_ERROR;
684         }
685     }
```

6 Conclusions

Fluent Bit is a log processor written in C that handles a lot of data flow effectively and efficiently. In this security audit of Fluent Bit we looked at the Fluent Bit core and relied on various techniques to analyse the code. The primary component was fuzzing, which is highly effective against a codebase like Fluent Bit.

Fluent Bit has now a total of 29 fuzzing harnesses running on OSS-Fuzz, as can be seen from its Fuzz Introspector report here. Additionally, Fluent Bit has run on OSS-Fuzz for around 5 years, which is an impressive effort from Fluent Bit's perspective. A total of 11 issues were uncovered during the audit. In addition to reporting the issues, Ada Logics also supplied fixes for each of them, all of which have been merged into Fluent Bit and its dependencies.

Fluent Bit has a strong fuzzing stance, including running fuzzing harness on PRs before they are merged in. We applaud this but also encourage Fluent Bit's maintainers to continue putting effort into expanding the fuzzing coverage as well as ensuring new code is appropriately fuzzed. We consider this a central part of Fluent Bit's future security work.

We would like to thank Eduardo Silva Pereira, Leonardo Albertovich and the rest of the Fluent Bit maintainers for the collaboration, as well as the CNCF for sponsoring this security audit.