Dependency Assessment: cborg

Prepared by FP Complete

For The Cardano Foundation

CARDANO FOUNDATION

August 2018

Legend

The following report uses a grading system that provides a subjective rating of the impact, significance and execution of the audited aspects. This is intended to assist the audience's understanding.

Flag Severity

Green Well executed in the project

Yellow Potential flaw, or process limitation which may lead to a flaw

Red Confirmed flaw

Question Scoring

Good Meets or exceeds industry standards.

Fair Partial implementation does not meet industry standards.

Poor No implementation.

The questions covered in this report come from a standard set of questions we ask when reviewing any software project. The fact that there is a "poor" response for some question does not mean automatically that something requires fixing but it does warrant further investigation.

This report pertains to the following commit:

`github.com/well-typed/cborg: d7548c03134a494f2c0357d8b6e373d18c3a0871`

Summary

Questions

Title	Links
General	
Is the library mature?	Details
Is the library the standard solution for the problem it solves?	Details
Does the library have a reputation?	Details
Is the library well-maintained?	Details
Quality	
Does it have CI?	Details
Does it have 90-100% code coverage?	Details
A test suite with property-based testing?	Details
Has a test suite that is easy to run?	Details
Aggressive test suite?	Details
Does it have benchmarks?	Details

Flags

Title	Links	Status
Vulnerabilities		
Pervasive use of unsafe functions, unsupported by sufficient tests	Details	
Maintenance		
Casting words to floats	Details	
Low-level and repetitive code	Details	

Questions in Detail

General

Is the library mature?

Answer:

No, it is one year old and first published last year.

Score: Poor

Source: source-code

Is the library the standard solution for the problem it solves?

Answer:

No, there are more widely accepted libraries for the purpose of data serialization in general. If the question is only about CBOR format serialization then this library could be considered the standard solution.

And even if it is not the standard solution for serialization there exist enough of compelling motivation to use this library:

- the more standard solutions (`binary` and `cereal`) have known performance issues and some poor serialization behavior around floating-point values and `cborg` fixes this
- `cborg` provides easier interop with other languages and that could be a very important aspect while integrating with third-party systems
- potentially `cborg` makes it easier to extend the binary format and detect data corruption

Score: Fair

Source: source-code

Does the library have a reputation?

Answer:

No. The audit team could find no reports of use outside of Cardano project.

Score: Poor

Source: source-code

Is the library well-maintained?

Answer:

Yes, work continues to present day.

Score: Good

Source: source-code

Quality

Does it have CI?

Answer:

Yes. However: Linux is tested, against GHC 7.10.3 to GHC 8.0.2. Windows is not tested. 32-bit systems are not tested and it raises additional concern as the library source code contains portions of code which are enabled with CPP directive `#if defined(ARCH_32bit)`. The AppVeyor CI build has been disabled. Their last build was 1 year ago.

Score: Poor

Source: source-code

Does it have 90-100% code coverage?

Answer:

No.

Coverage report:

```
51% expressions used (4869/9465)
10% boolean coverage (23/212)
      9% guards (19/199), 42 always True, 5 always False, 133 unevaluated
     30% 'if' conditions (4/13), 1 always False, 8 unevaluated
    100% qualifiers (0/0)
48% alternatives used (630/1305)
64% local declarations used (60/93)
54% top-level declarations used (208/381)
```

Given the use of unsafe functions (see the flag titled “Pervasive use of unsafe functions”), we expect this library to have a test coverage close to 100%.

Score: Poor

Source: source-code

A test suite with property-based testing?

Answer:

Yes. Each test suite module contains property testing.

Score: Good

Source: source-code

Has a test suite that is easy to run?

Answer:

Yes, it uses standard Cabal test suite stanzas which can be run via common Haskell build tools, including `cabal-install` and Stack.

Score: Good

Source: source-code

Aggressive test suite?

Answer:

The test suite is fairly aggressive. There are:

- Roundtrip tests (`cborg/tests/Tests/CBOR.hs`)
- Reference implementation tests (`cborg/tests/Tests/Reference.hs`)
- Boundary tests (`cborg/tests/Tests/Boundary.hs`)
- Regression tests (`cborg/tests/Tests/Regress.hs`)
- Failure tests (`cborg/tests/Tests/UTF8.hs`)
- Golden tests (`cborg/tests/test-vectors`)

Score: Good

Source: source-code

Does it have benchmarks?

Answer:

No. There isn’t a benchmark stanza in this library cabal file.

However, there is the related `serialise` package, which does have a benchmark suite.

Score: Poor

Source: source-code

Flags

Detailed flags: Vulnerabilities

● Pervasive use of unsafe functions, unsupported by sufficient tests

The package makes pervasive use of unsafe functions:

- Raw use of the `IO` data constructor to inject things into the `IO` monad.
- Use of explicit pinning, predicates on pinned values, which can affect whether memory is moved or not.
- Use of `unsafePerformIO` and its friends.
- Use of raw pointers, raw byte arrays, etc.
- Use of `unsafeHead`, `unsafeTail`, etc. type of operations which are faster but if called wrongly will memory overrun at worst and segfault at best.

For example, `unsafeHead` from the bytestring package:

```
isBigIntRepCanonical :: ByteString -> Bool
isBigIntRepCanonical bstr = BS.length bstr > 8 && BS.unsafeHead bstr /= 0x00
```

Functions like this, and any defined in terms of them, are vulnerable to causing memory overruns. They do not check the bounds of memory like their counterparts. Thus far, no examples of this precondition not being satisfied have been identified.

It is understandable that a high performance serialization library could have good reasons to use such a low level oriented and potentially unsafe functions but to cover the dangers from that we would highly recommend to use thorough code reviews, extensive tests suite with a good coverage and CI being run for every supported combination of GHC version, operating system and CPU architecture. When looking at the test coverage report, we can clearly see that many of the uses of unsafe functions are never tested.

First occurrence: Thursday 9 August 2018

Detailed flags: Maintenance

● Casting words to floats

In `Codec.CBOR.Magic` there are custom functions that cast words to floats, which is a little dangerous if done wrongly or if GHC's internals change in the wrong way. These custom functions are now provided by GHC in base.

First occurrence: Thursday 9 August 2018

● Low-level and repetitive code

- There is a lot of repetitive code, similar to low-level C code, and therefore plenty of it has "no obvious bugs" rather than "obviously no bugs". Probably the most prominent example of that is the `go_fast` function which spans over 400 of lines and contains code segments which differ only in minor details, e.g.

```
go_fast !bs da@(ConsumeWord8Canonical k) =
    case tryConsumeWord (BS.unsafeHead bs) bs of
```

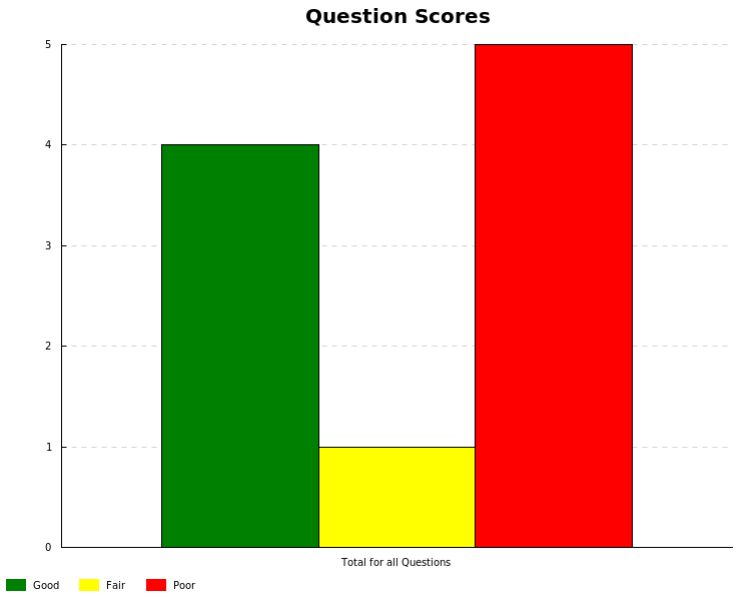
```
      DecodeFailure           -> go_fast_end bs da
      DecodedToken sz (W# w#) ->
        case gtWord# w# 0xff## of
          0# | isWordCanonical sz w# -> k w# >>= go_fast (BS.unsafeDr
op sz bs)
          _                          -> go_fast_end bs da

go_fast !bs da@(ConsumeWord16Canonical k) =
    case tryConsumeWord (BS.unsafeHead bs) bs of
      DecodeFailure           -> go_fast_end bs da
      DecodedToken sz (W# w#) ->
        case gtWord# w# 0xffff## of
          0# | isWordCanonical sz w# -> k w# >>= go_fast (BS.unsafeDr
op sz bs)
          _                          -> go_fast_end bs da
```

- In the `Codec.CBOR.Read` module, there is a “fast path” and “slow path” separation in which the same readers are implemented again in a different way.
- As the reader makes no use of a parsing abstraction (see again `Codec.CBOR.Read`), which would handle things like bounds checks and chunking, this makes the code more prone to potential bugs and maintenance overhead.

First occurrence: Thursday 9 August 2018

Report Statistics

Question Score	Total
Good	4
Fair	1
Poor	5
Un-Scored	0
Total	10



	Public	Private	Total
Green flags	0	0	0
Yellow flags	3	0	3
Red flags	0	0	0
Total Flags	3	0	3