Optim Protocol

Audit Report - Revision 1

MLabs Audit Team

December 2, 2022 (Revised December 9, 2022)

Contents

1 Revision Notes 4
1.1 Revision 1 - 2022-12-09 . . . 4

2 Disclaimer 5

3 Contents 5
3.1 Audit Background . . . 5
3.1.1 Scope . . . 5
3.1.2 Methodology . . . 6
3.1.3 Documentation Links . . . 8
3.2 Inter-Audit Fixes . . . 8

4 Audit 11
4.1 Executive Summary . . . 11
4.2 Audit Report Structure . . . 11
4.3 Not Considered Fixed . . . 11
4.3.1 Review of Issues 2.3.1.1 / 2.3.1.2 / 2.3.1.3 . . . 11
4.3.2 Review of Issue 2.3.1.10 . . . 12
4.3.3 Review of Issue 2.3.1.4 . . . 12
4.3.4 Review of Issue 2.3.2.2 . . . 12
4.3.5 Review of Issue 2.3.3.2 . . . 13
4.3.6 Out of date spec (OpenPool Validator) . . . 13
4.3.7 Linting and Formatting . . . 13
4.3.8 Insufficient Comments and Type Annotations . . . 14
4.3.9 Missing Tests . . . 14
4.4 Considered Fixed . . . 15
4.4.1 Review of Issues 2.3.1.8/2.3.1.9 . . . 15
4.4.2 Review of Issue 2.3.3.3 . . . 15
4.4.3 Review of Issue 2.3.1.6 . . . 15
4.4.4 Review of Issues 2.3.1.5 / 2.3.2.1 . . . 15
4.4.5 Review of Issue 2.3.1.7 . . . 16
4.4.6 Review of issue 2.3.3.1 . . . 16
4.4.7 Review of Issue O.V.1 . . . 17
4.4.8 Review of Issue O.V.2 . . . 17
4.4.9 Review of Issue O.V.3 . . . 17
4.4.10 Review of Issue O.V.4 . . . 18
4.4.11 Review of Issue O.V.6 . . . 19
4.4.12 Review of Issue O.V.7 . . . 19
4.4.13 Review of Issue B.F.1 . . . 20
4.4.14 Review of Issue B.F.2 . . . 21
4.4.15 Review of Issue O.V.5 . . . 21
4.5 Recommendations . . . 23
4.5.1 Plutarch Code Recommendations . . . 23
4.5.2 Optimisation Recommendation . . . 23
4.5.3 Removed Tweag Traces . . . 23

5 Conclusion 25

6 Appendix **26**
6.1 Vulnerability types . . . 26
6.1.1 Other redeemer . . . 26
6.1.2 Other token name . . . 26
6.1.3 Unbounded Protocol datum . . . 27
6.1.4 Arbitrary UTxO datum . . . 27
6.1.5 Unbounded protocol value . . . 27
6.1.6 Foreign UTxO tokens . . . 28
6.1.7 Multiple satisfaction . . . 28
6.1.8 Locked Ada . . . 28
6.1.9 Locked non Ada values . . . 29
6.1.10 Missing UTxO authentication . . . 29
6.1.11 Missing incentive . . . 29
6.1.12 Bad incentive . . . 29
6.1.13 UTxO contention . . . 30
6.1.14 Cheap spam . . . 30
6.1.15 Insufficient tests . . . 30
6.1.16 Incorrect documentation . . . 30
6.1.17 Insufficient documentation . . . 31

1 Revision Notes

1.1 Revision 1 - 2022-12-09

1. Added in previously (accidentally) omitted reviews to the original report.

- *Review of Issue B.F.2*
- *Out of date spec (OpenPool Validator)*

3. Removed the *Review of Fix* comments for brevity, at the request of Optim.

4. Minor text changes for readability purposes.

2 Disclaimer

This audit report is presented without warranty or guarantee of any type. Neither MLabs nor its auditors can assume any liability whatsoever for the use, deployment or operations of the audited code. This report lists the most salient concerns that have become apparent to MLabs' auditors after an inspection of the project's codebase and documentation, given the time available for the audit. Corrections may arise, including the revision of incorrectly reported issues. Therefore, MLabs advises against making any business or other decisions based on the contents of this report.

An audit does not guarantee security. Reasoning about security requires careful considerations about the capabilities of the assumed adversaries. These assumptions and the time bounds of the audit can impose realistic constraints on the exhaustiveness of the audit process. Furthermore, the audit process involves, amongst others, manual inspection and work which is subject to human error.

MLabs does not recommend for or against the use of any work or supplier mentioned in this report. This report focuses on the technical implementation provided by the project's contractors and subcontractors, based on the information they provided, and is not meant to assess the concept, mathematical validity, or business validity of their product. This report does not assess the implementation regarding financial viability nor suitability for any purpose. *MLabs does not accept responsibility for any loss or damage howsoever arising which may be suffered as result of using the report nor does it guarantee any particular outcome in respect of using the code on the smart contract.*

3 Contents

3.1 Audit Background

3.1.1 Scope

During the audit, MLabs has inspected the code contained in the provided files and attempted to locate problems that can be found in the following categories:

1. Review of proposed fixes/comments to previous Tweag Audit. The following list is a summary of issues and their considered status in the current implementation - a ticked box represents a fixed issue, while an empty box represents an issue deemed not fixed.

☒ *Review of Issue 2.3.1.6*
☒ *Review of Issues 2.3.1.8/2.3.1.9*
☒ *Review of Issue 2.3.3.3*
☐ *Review of Issues 2.3.1.1 / 2.3.1.2 / 2.3.1.3*
☐ *Review of Issue 2.3.1.10*
☒ *Review of Issues 2.3.1.5 / 2.3.2.1*
☐ *Review of Issue 2.3.1.4*
☒ *Review of Issue 2.3.1.7*
☒ *Review of issue 2.3.3.1*
☐ *Review of Issue 2.3.2.2*
☐ *Review of Issue 2.3.3.2*

2. Review proposed fixes/comments to communicated findings since Tweag Audit (findings included in the report in section *Inter-Audit Fixes*). The following list should be read in a similar manner to the one above.

☒ *Review of Issue O.V.1*
☒ *Review of Issue O.V.2*

- ☒ *Review of Issue O.V.3*
- ☒ *Review of Issue O.V.4*
- ☒ *Review of Issue O.V.5*
- ☒ *Review of Issue O.V.6*
- ☒ *Review of Issue O.V.7*
- ☒ *Review of Issue B.F.1*
- ☒ *Review of Issue B.F.2*

The original scope is limited to the above two points. Time permitting, MLabs Audit team agreed to undertake the following points in addition to the original scope.

3. Review contracts for any additional vulnerabilities against MLabs list of Vulnerability types. The exploration having identified the following issues:

- ☐ *Missing Tests*
- ☐ *Insufficient Comments and Type Annotations*
- ☐ *Linting and Formatting*
- ☐ *Out of date spec (OpenPool Validator)*
- ☐ *Plutarch Code Recommendations*
- ☐ *Optimisation Recommendation*
- ☐ *Removed Tweag Traces*

Please note that the *Vulnerability Type* exploration was done in a limited capacity and not all the vulnerability types were explored for the protocol.

3.1.2 Methodology

3.1.2.1 Teams Involved

In the following document, we will refer freely to the MLabs Audit Team, as MLabs. This note is made to disambiguate between the Audit and Development teams. Although MLabs (the company) has been actively involved in both the development and audit of the protocol, the two teams (specifically the development and audit teams) are distinct and separate, with no conflicts of interest.

3.1.2.2 Information

MLabs analysed the validators and minting scripts from the `Optim-Onchain` git repository starting with commit `6a16307 - Audit fixes`. MLabs also used as basis for their research the repository `optim-spec`, pinned at commit `eaff83f - Align duration with epoch boundaries`. As a side note, there exists an Optim web interface, which was used to give some indication of UX/UI design - although this was not key to the audit process.

A last mention is the information provided, and included as part of the report in *Inter-Audit Fixes*. These issues were found by the MLabs Development team, and have also been audited as part of the process.

3.1.2.3 Process

During the review process the issues and vulnerabilities were posted directly to a forked repository as GitHub Issues tracked under a meta GH Issue. Some of the findings were accompanied by additional recommendations.

3.1.2.4 Audited Files Checksums

The following checksums are those of files captured by commit `6a16307`, and were generated using the following sha256 binary:

```
$ sha256sum --version
sha256sum (GNU coreutils) 9.0
```

The checksums of the smart-contract implementation code are:

```
b840...a951  ./tryfrom-tests/Test/TryFrom.hs
951a...4cf4  ./optim-common-plutarch/Optim/Onchain/Common/Plutarch/Types.hs
e9d5...a840  ./optim-common-plutarch/Optim/Onchain/Common/Plutarch.hs
f098...e6e2  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Contracts/Open.hs
3615...983a  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Contracts/BondToken.hs
06b0...8251  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Contracts/Closed.hs
da90...ff23  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Contracts/BondWriter.hs
cc92...25e2  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Contracts/ClosedPool.hs
e907...756b  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Contracts/OpenPool.hs
5a5f...d6dc  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Contracts/Nft.hs
fe26...2cba  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Contracts/PoolToken.hs
daac...6121  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Types/Open.hs
821c...372b  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Types/BondToken.hs
42e6...2533  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Types/Closed.hs
bb03...6e5b  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Types/BondWriter.hs
4fb4...66c6  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Types/ClosedPool.hs
8eaf...8e12  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Types/OpenPool.hs
70db...526e  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Types/Nft.hs
40fb...8fda  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Types/PoolToken.hs
fa1f...8d9f  ./optim-bonds/src/Optim/Onchain/Bonds/Plutarch/Contracts.hs
8cfb...d556  ./optim-bonds/app/Main.hs
```

The check-sum of the documentation, audit report, response and documents from `optim-spec eaff83f` are:

```
ecab...f3ca  ./ibo-jit-borrow.md
7a64...1b49  ./spo-bond-pools.md
60df...bb61  ./liquidity-bonds.md
4564...8e90  ./README.md
3de5...77dc  ./audit/tweag-audit-response.md
c55a...c171  ./liqudity-bonds.pdf
dc8d...a499  ./audit/tweag-audit-report.pdf
a734...1a88  ./spo-bond-pools.pdf
```

3.1.2.5 Audit Timeline

MLabs spent three, five day sprints in an exploratory phase of the protocol, auditing all the vulnerabilities and fixes provided. This can be seen as auditors inspecting all the provided modules and documentation, getting acquainted with the protocol and the codebase. This exploratory phase was coupled with a structured phase, focusing on individual smart contract vulnerability exploration, and penetration testing. Each auditor focused on auditing all the previously identified vulnerabilities, their contract implementation, and the proposed fixes. Swapping, and re-reviewing fixes in a *round robin* fashion, each auditor covered the entirety of the code-base. During the final sprint, the team also undertook a process of formalising the findings for the audit report.

3.1.2.6 Metrics

3.1.2.6.1 CVSS

To leverage a standardised metric for the severity of the open standard Common Vulnerability Scoring System, together with the NVD Calculator. The metrics from the mentioned tools were included with each vulnerability. MLabs recognises that some of the parameters are not conclusive for the protocol - but considers that leveraging such a standard is still valuable to offer a more unbiased severity metric for the found vulnerabilities.

3.1.2.6.2 Severity Levels

The aforementioned CVSS calculations were then benchmarked using the CVSS-Scale metric, receiving a grade spanning from `Low` to `Critical`. This additional metric allows for an easier, human understandable grading, whilst leveraging the CVSS standardised format.

3.1.3 Documentation Links

We include the following links to the aforementioned resources.

1. Optim specification `eaff83fc5ce5544aa5c34d822509189fec336138`
2. Optim implementation `6a16307904f25da1bf22fe1722b02596a6c6ab79`
3. Optim dAPP
4. Inter-Audit Fixes document
5. Optim Tweag Audit Report

3.2 Inter-Audit Fixes

The following is a list of vulnerabilities found in the `optim-onchain` repository and the solutions implemented in response.

The list is divided into two categories:

1. Those identified during the Tweag audit of optim-onchain and included in the resulting report with suggested solutions.
2. Those identified and corrected by Optim independent of the Tweag audit.

3.2.0.1 Vulnerabilities included in Tweag report

Problem: A bond can be written using an `otmFee` value of 100 BasisPoints
Potential vulnerability: This can prevent the fee from being paid and the position from being closed.
Solution: Add Optim fee as parameter to Bond Token script, incorporating this into the bond as an instrument

Problem: The UniqNFT is not burned on Cancel
Potential vulnerability: The same token can be used for two different BWV positions to mislead potential lenders, esp. via pools.
Solution: Add check to burn UniqNFT on Cancel

Problem: The datum is unrestricted between Closed Pool interactions
Potential vulnerability: Alternative Pool Tokens can be minted and specified in the Closed Pool datum, allowing the bonds to be redeemed at no cost.
Solution: Check datum hash in Closed Pool (fix, response)

Problem: The value at the Open position is not verified by the Open Pool
Potential vulnerability: A pool of more tokens than a BWV position can pay only enough to satisfy the Open position and redirect any excess Lovelace elsewhere.
Solution: Check that Open Pool value is no greater than the face value of the number of minted bonds plus one

3.2.0.2 Other vulnerabilities

3.2.0.2.1 O.V.1

Problem: Value comparisons use `punionWith` for comparisons
Potential vulnerability: These checks are semantically unclear and require care to ensure the correct behaviour for asset classes not represented in both values. In addition, excess assets were permitted to potentially inflate UTxO sizes.
Solution: Compare values in a zip-like operation with a predicate, defaulting to validation failure on unmatched asset classes

3.2.0.2.2 O.V.2

Problem: Optim fee on Close is checked as equality
Potential vulnerability: A reward above a certain size may increase the minimum UTxO value beyond the provided Lovelace, making it impossible to close the position.
Solution: Allow the fee paid to Optim to be greater than the minimum, paid by the transaction sender

3.2.0.2.3 O.V.3

Problem: Optim fee UTxO is identified by address only
Potential vulnerability: With no restriction on Open inputs, multiple equal-value positions can be closed simultaneously and identify the same fee UTxO as being paid. The expected fee from all but one of these positions can be redirected.
Solution: Identify the paid fee UTxO by both address and the position's token name in datum

3.2.0.2.4 O.V.4

Problem: Open pool has no restriction on input and output during Match
Potential vulnerability: Multiple Open Pool inputs of equal size, with tokens minted together, to the matching BWV can be matched together, each identifying the same single Closed Pool and Open outputs as correct. In this case, the funds from all but one of the Open Pools can be redirected.
Solution: Restrict the Match operation to a single Open Pool input and a single Closed Pool output

3.2.0.2.5 O.V.5

Problem: Pool output values are only compared on Lovelace and known tokens
Potential vulnerability: Excess assets can be added to pool UTxOs, inflating the associated size of transactions and minimum UTxO value.
Solution: Compare entire input and output values including expected deltas

3.2.0.2.6 O.V.6

Problem: The BWV value is unchecked for excess tokens during Write
Potential vulnerability: A malicious SPO can potentially add enough tokens on issue to prevent the position from ever being closed. In addition, this complicates the check for underperforming positions.
Solution: Require the written bond to have at least 1 of each reward asset with a zip-like operation on the input value and `epoRewards`

3.2.0.2.7 O.V.7

Problem: The amount of asset classes and token units provided as rewards is unbounded
Potential vulnerability: A malicious SPO can potentially offer a bond with a reward token they can freely mint. With unbounded `Integer`s, a suitable amount of this token can be added to the position to increase the transaction size and preventing the position from ever being closed.
Solution: Restrict both the number of asset classes and the amount of each in the rewards

3.2.0.3 Bug fixes/features

3.2.0.3.1 B.F. 1

Problem: An Open Pool with a `minPrepaid` of 0 cannot match a BWV due to a type restriction
Solution: Replace the `'Positive` value guarantee with `'NoGuarantees`

3.2.0.3.2 B.F. 2

Problem: A bond's duration is counted in full epochs rather than epoch boundaries crossed since opening. As a result, the borrower side receives `duration + 1` epochs worth of staking rights while the lender receives `duration` epochs worth of rewards.
Solution: Require rewards for each epoch boundary to be paid in advance of that boundary in order to maintain the position and consider the position matured as soon as `duration` epoch boundaries have been crossed.

4 Audit

4.1 Executive Summary

The audit findings can be categorised as the following vulnerability types:

1. Issued deemed not fixed from previous audits ×7.
2. `insufficient-tests` ×3.
3. `insufficient-documentation` ×1.
4. `poor-code-standards` ×1.
5. `out-of-date-spec` ×1.
6. `recommendations` ×2.

4.2 Audit Report Structure

The following sections group our findings into issues which are fixed, not fixed, and finally some recommendations from the MLabs Audit team.

4.3 Not Considered Fixed

4.3.1 Review of Issues 2.3.1.1 / 2.3.1.2 / 2.3.1.3

Note: The reviews are grouped together because: the vulnerabilities are transitive, (i.e. 2.3.1.3 is an extension of 2.3.1.2 which depends on 2.3.1.1). In each case the Optim team does *not* address the vulnerabilities onchain.

4.3.1.1 Description

- *uniqNFT* can be redirected when posting a bond (2.3.1.1).
- Open validator can contain unsound bonds (2.3.1.2).
- Funds can be locked with irrevocable staking rights using unsound bonds (2.3.1.3).

4.3.1.2 Fix/Comment

As mentioned above, the Optim team does not address each of these issues. In their response to the Tweag audit, they provide the following reasoning:

> 2.3.1.1
>
> This is intended design, a feature so to say.
>
> 2.3.1.2, 2.3.1.3
>
> The impact of the issue is not felt in the intended behavior. A user is forefitting funds upon donation always.

4.3.1.3 Review of Fix

We deem that the onchain fix in response to the above issue is not adequate.

4.3.1.4 Links

- response to the Tweag audit

4.3.2 Review of Issue 2.3.1.10

4.3.2.1 Description

- Bond tokens can be redeemed with different pool tokens

4.3.2.2 Fix/Comment

Optim team's response to the Tweag audit:

> 2.3.1.10
>
> Fixed by filtering out pools that don't respect the invariant: N*100 = pool tokens* 100 + ada

Note: Although it wasn't mentioned in Optim's response to the Tweag audit, the data tampering fix for the *ClosedPool* validator (see review of issues 2.3.1.8 & 2.3.1.9) addresses this issue as well.

4.3.2.3 Review of Fix

We deem that the onchain fix in response to the above issue is not adequate.

4.3.3 Review of Issue 2.3.1.4

4.3.3.1 Description

Pool tokens target is unchecked.

4.3.3.2 Fix/Comment

Optim team's response to Tweag audit:

> 2.3.1.4
>
> Fixed in 2.3.1.10.

For additional context:

> 2.3.1.10
>
> Fixed by filtering out pools that don't respect the invariant: N*100 = pool tokens* 100 + ada

4.3.3.3 Review of Fix

We deem that the onchain fix in response to the above issue is not adequate.

4.3.4 Review of Issue 2.3.2.2

4.3.4.1 Description

uniqNFT is sometimes required as fees.

4.3.4.2 Fix/Comment

Optim team's response to Tweag audit:

> 2.3.2.2
>
> The % fee is a global constant as our scripts are constant. A non-issue for our purposes.

4.3.4.3 Review of Fix

We deem that the onchain fix in response to the above issue is not adequate.

4.3.5 Review of Issue 2.3.3.2

4.3.5.1 Description

Initial margin is unchecked.

4.3.5.2 Fix/Comment

N/A - The Optim team did not address his issue in their response to the Tweag audit report.

4.3.5.3 Review of Fix

We deem that the onchain fix in response to the above issue is not adequate.

4.3.6 Out of date spec (OpenPool Validator)

Severity	CVSS	Vulnerability type
None	0.0	out-of-date-spec

4.3.6.1 Description

There is a discrepancy between the specification and implementation of the parameters for the `OpenPool` validator.

4.3.7 Linting and Formatting

4.3.7.1 Description

Severity	CVSS	Vulnerability type
None	0.0	poor-code-standards + insufficient-tests

There exists a lack of making use of standard formatting and linting tools. Furthermore, the use of the tools is also absent from the CI tests.

4.3.7.2 Recommendation

1. Make use of formatting and linting in the CI. There are places where CI would have failed due to HLint. Using linting and formatting consistently makes code easier to read, easier to refactor, and eases collaboration.
2. Make use of tools like `haskell-language-server`. Such tools ensure that hints and warnings are resolved and visible during the development process.
3. Keep the `nix` code up-to-date as much as possible. There is a fast-moving `nix` code deprecation cycle with Plutarch - making this difficult. However, in the current situation the audit team could not easily repair the `nix` code as doing so would either take too long or break existing code.

4. Use tools like `pre-commit-hooks.nix` to ensure code quality checks are followed; MLabs internal template provides an easy way to integrate such tools.

4.3.8 Insufficient Comments and Type Annotations

Severity	CVSS	Vulnerability type
None	0.0	insufficient-documentation

4.3.8.1 Description

There is a general lack of documentation across the repository. We make the following observations:

1. Long do blocks are really hard to read. Comments and type annotations on binds help tremendously.
2. A good example are fixes for the audit: the audit team have spent a considerable amount of time tracking down where the fixes were applied - something that could have been avoided via the addition of a documentation string or similar.
3. A doc-string makes code easier to read and helps with orientation within the code. It also makes compiled documentation readable and serves as an aid for future improvements and maintenance.

4.3.9 Missing Tests

Severity	CVSS	Vulnerability type
None	0.0	insufficient-tests

4.3.9.1 Description

Plutarch helper functions with non-trivial logic should be property, and unit tested.

A relevant is example is the lack of tests for a function like `pfindOutputByAddressWithDatum`. Even if the function is correct, future changes to serialization could easily break it, and would not be easily caught in the absence of a test.

4.3.9.2 Recommendation

Plutarch provides tooling for golden testing, which should be used in this case.

4.4 Considered Fixed

4.4.1 Review of Issues 2.3.1.8/2.3.1.9

NOTE: the two issues were grouped together because the latter depends upon the former and the fix is the same (i.e. in the same place in the same module) for both.

4.4.1.1 Description

- Funds can be locked after datum tampering (2.3.1.8).
- Bond tokens can be stolen after datum tampering (2.3.1.9).

4.4.1.2 Fix/Comment

Optim team's comment: `Fixed by adding selfIn datumHash = selfOut datumHash`

Source link to plutarch implementation of fix

4.4.1.3 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.2 Review of Issue 2.3.3.3

4.4.2.1 Description

Bond tokens and pool tokens are handled differently.

4.4.2.2 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.3 Review of Issue 2.3.1.6

4.4.3.1 Description

Pools can be emptied by matching a small bond.

4.4.3.2 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.4 Review of Issues 2.3.1.5 / 2.3.2.1

Note: 2.3.1.5 and 2.3.2.1 are grouped together because: The bug in 2.3.2.1 enables the exploit in 2.3.1.5, and the fix for 2.3.2.1 mitigates the vulnerability in 2.3.1.5.

4.4.4.1 Description

- Pool target can be compromised using another uniqNFT (2.3.1.5 - Vulnerability).
- uniqNFT can be redirected when cancelling a bond (2.3.2.1 - Implementation bug).

4.4.4.2 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.4.3 Links

- Fix in code

4.4.5 Review of Issue 2.3.1.7

4.4.5.1 Description

Pools can be emptied by matching a bond with the cancel redeemer

4.4.5.2 Fix/Comment

Optim team's response to Tweag audit:

> 2.3.1.7
>
> Fixed by adding selfIn ADA < (bondAmount + 1) * 100 ADA to OpenPool

Refer to issue 2.3.1.6 for a link to the relevant Plutarch source.

4.4.5.3 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.5.4 Links

- Issue 2.3.1.6

4.4.6 Review of issue 2.3.3.1

4.4.6.1 Description

Buffer boundary is open on the right

4.4.6.2 Fix/Comment

Optim team's response to Tweag audit:

> 2.3.3.1
>
> Given that the duration parameter takes priority over buffer, it is not a concern.

4.4.6.3 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.6.4 Links

- Implementation of proposed fix

4.4.7 Review of Issue O.V.1

4.4.7.1 Description

Value comparisons use *punionWith* for comparisons.

> Potential vulnerability: These checks are semantically unclear and require care to ensure the correct behaviour for asset classes not represented in both values. In addition, excess assets were permitted to potentially inflate UTxO sizes.

4.4.7.2 Fix/Comment

> Solution: Compare values in a zip-like operation with a predicate, defaulting to validation failure on unmatched asset classes.

4.4.7.3 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.8 Review of Issue O.V.2

4.4.8.1 Description

Optim fee on Close is checked as equality.

> Potential vulnerability: A reward above a certain size may increase the minimum UTxO value beyond the provided Lovelace, making it impossible to close the position.

4.4.8.2 Fix/Comment

`Solution: Allow the fee paid to Optim to be greater than the minimum, paid by the transaction sender`

```
-- Open.hs 174-
otmPaid =
  zipValueWith # plam (#>=) #
    otmValue #
      (pnormalize # otmFee <> psingleton # padaSymbol # padaToken # 2_000_000)
```

Link to source

4.4.8.3 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.9 Review of Issue O.V.3

4.4.9.1 Description

- Optim fee UTxO is identified by address only

  Potential vulnerability: With no restriction on Open inputs, multiple equal-value positions can be closed simultaneously and identify the same fee UTxO as being paid. The expected fee from all but one of these positions can be redirected.

4.4.9.2 Fix/Comment

Solution: Identify the paid fee UTxO by both address and the position's token name in datum

```
-- Open.hs 157
PJust otmOut <- pmatch $ pfindOutputByAddressWithDatum # otmAddr # pto tokenName # infoFs.outputs
```

Link to source

where

```
-- Plutarch.hs 312-329
pfindOutputByAddressWithDatum :: ClosedTerm (PAddress
                                            :--> PByteString
                                            :--> PBuiltinList (PAsData PTxOut)
                                            :--> PMaybe (PAsData PTxOut))
pfindOutputByAddressWithDatum = phoistAcyclic $
    plam $ \(targetAddress :: Term s PAddress) targetDatum txOuts -> P.do
      justDatumHash <- plet . pcon . PDJust $
        pdcons # pdata (pcon . PDatumHash $ pblake2b_256 #$ bsCbor # targetDatum) # pdnil
      let pred :: Term s (PAsData PTxOut :--> PBool)
          pred = plam $ \actual ->
            (#&&)
              (pdata targetAddress #== pfield @"address" # actual)
              (justDatumHash #== pfield @"datumHash" # actual)
      pfind # pred # txOuts
  where
    bsCbor :: ClosedTerm (PByteString :--> PByteString)
    bsCbor = plam $ \bs ->
      pconsBS # 0x58 #$ pconsBS # (plengthBS # bs) # bs
```

Link to source

4.4.9.3 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.10 Review of Issue O.V.4

4.4.10.1 Description

Problem: Open pool has no restriction on input and output during Match

Potential vulnerability: Multiple Open Pool inputs of equal size, with tokens minted together, to the matching BWV can be matched together, each identifying the same single Closed Pool and Open outputs as correct. In this case, the funds from all but one of the Open Pools can be redirected.

4.4.10.2 Fix/Comment

Solution: Restrict the Match operation to a single Open Pool input and a single Closed Pool output

```
-- OpenPool.hs 159-166
          openPoolInputs =
            pfilterInputByValidatorHash # (pfield @"_0" # selfInValidatorHash)
```

```
                                        # inputs
          poolInputUnique = plength # openPoolInputs #== 1

        PJust closedPoolOut' <-
          pmatch $ punique #$ pfilterOutputByValidatorHash # closedPoolVh # outputs
        closedPoolOut :: Term s PTxOut <- plet $ pfromData closedPoolOut'
```

Link to source

4.4.10.3 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.11 Review of Issue O.V.6

4.4.11.1 Description

The BWV value is unchecked for excess tokens during *Write* operation.

> Potential vulnerability: A malicious SPO can potentially add enough tokens on issue to prevent the position from ever being closed. In addition, this complicates the check for underperforming positions.

4.4.11.2 Fix/Comment

> Solution: Require the written bond to have at least 1 of each reward asset with a zip-like operation on the input value and `epoRewards`

```
-- BondWriter.hs 146-153
          exactAssetClasses =
            zipValueWith
              # plam
                (\valueAmount rewardAmount ->
                  valueAmount #< maxAssetAmount #&&
                  rewardAmount * pto duration #< maxAssetAmount)
              # inValue
              # (epoRewards <> uniqNftValue)
```

Link to source

4.4.11.3 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.12 Review of Issue O.V.7

4.4.12.1 Description

The amount of asset classes and token units provided as rewards is unbounded.

> Potential vulnerability: A malicious SPO can potentially offer a bond with a reward token they can freely mint. With unbounded Integers, a suitable amount of this token can be added to the position to increase the transaction size and preventing the position from ever being closed.
>
> Solution: Restrict both the number of asset classes and the amount of each in the rewards

4.4.12.2 Fix/Comment

```
-- BondWriter.hs 146-159
          exactAssetClasses =
            zipValueWith
              # plam
                (\valueAmount rewardAmount ->
                  valueAmount #< maxAssetAmount #&&
                  rewardAmount * pto duration #< maxAssetAmount)
              # inValue
              # (epoRewards <> uniqNftValue)
          numRewardAssets =
            pfoldr # plam (+) # 0 #$
              pmap #
                plam (\xs -> plength #$ pto . pfromData $ psndBuiltin # xs) #
                pto (pto epoRewards)
          rewardAssetsBounded = numRewardAssets #<= 32
```

Link to source

4.4.12.3 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.13 Review of Issue B.F.1

4.4.13.1 Description

Problem: An Open Pool with a minPrepaid of 0 cannot match a BWV due to a type restriction

4.4.13.2 Fix/Comment

Solution: Replace the 'Positive value guarantee with 'NoGuarantees.

```
-- OpenPool.hs 238-245
  where
    rewardsPrepaid :: Term s (
                        PValue 'Sorted 'Positive
                      :--> PInteger
                      :--> PValue 'Sorted 'NonZero
                      :--> PBool)
    rewardsPrepaid = plam $ \rewards epochs value -> P.do
      rewards' <- plet $ pmapAmounts # plam (* epochs) # rewards
      zipValueWith # plam (#<=) # rewards' # pforgetNonZero value
```

4.4.13.3 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.13.4 Links

- Link to source

4.4.14 Review of Issue B.F.2

4.4.14.1 Description

Problem: A bond's duration is counted in full epochs rather than epoch boundaries crossed since opening. As a result, the borrower side receives duration + 1 epochs worth of staking rights while the lender receives duration epochs worth of rewards.

4.4.14.2 Fix/Comment

Solution: Require rewards for each epoch boundary to be paid in advance of that boundary in order to maintain the position and consider the position matured as soon as duration epoch boundaries have been crossed.

The implementation of the fix consists in changing (Open.hs, inside `onClose`):

```
        epochsOwed <- plet $ pmin # datumFs.duration #$
                                    currentEpoch - datumFs.start + datumFs.buffer
-- (...)

          matured =
            datumFs.duration #< currentEpoch - datumFs.start
```

to:

```
        epochsOwed <- plet $ pmin # datumFs.duration #$
                                    currentEpoch - datumFs.start + datumFs.buffer + 1
-- (...)

          matured =
            datumFs.duration #<= currentEpoch - datumFs.start
```

Link to source

4.4.14.3 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.14.4 Links

- Link to source

4.4.15 Review of Issue O.V.5

4.4.15.1 Description

- Problem: Pool output values are only compared on Lovelace and known tokens

  Potential vulnerability: Excess assets can be added to pool UTxOs, inflating the associated size of transactions and minimum UTxO value.

4.4.15.2 Fix/Comment

Solution: Compare entire input and output values including expected deltas

```
-- OpenPool.hs 134-135
          ptraceIfFalse "n*100 ADA added to CO.defStk and n (Pool Token, poolTkn) removed from CO.def
            (pforgetPositive inValue <> paidLovelace #== pforgetPositive outValue <> outTokens)
```

4.4.15.3 Review of Fix

We deem the fix to be appropriate, solving the stated issue.

4.4.15.4 Links

- Link to source

4.5 Recommendations

4.5.1 Plutarch Code Recommendations

4.5.1.1 Description

Recommendations stemming from review of: *Review of Issue O.V.1*.

4.5.1.2 Recommendations

1. Don't pass a function along the recursive call if the function is constant. Instead, introduce another lambda that floats out this function
2. Don't `pfromData` and then compare for equality, checking for data equality is very cheap and is almost always preferred. `pfromData` also has an associated costs for primitive types, so don't do that if you can avoid it
3. *Minor*: Don't unnecessarily introduce let bindings if you only use the value once
4. If you are sure that if this function doesn't hold, the validator/ mintingpolicy can never succeed, then don't return `PBool` but instead return `POpaque` on success and `perror` on failure.
5. In Plutarch there exist optimized higher order recursion functions specialised to lists. Use of those should be preferred if possible. (We did not check whether use of those functions makes sense in this case, this is a general tip)
6. The documentation on those functions could be more extensive. For example, the toplevel functions docstring could have been:

   Tests that for a pair of $Value$ s xs and ys, they - contain the same assets - a predicate p holds for each pair of values in the underlying `AssocMap` - *Most important*: There are not any tests (neither property nor unit tests). It is recommended that non-trivial functions (like this) should be tested.

4.5.2 Optimisation Recommendation

4.5.2.1 Description

Recommendation from *Review of Issue O.V.2*.

4.5.2.2 Recommendation

Note, however, that here `zipValueWith` does not need to return a boolean but can *instantly* `perror` as noted in *Plutarch Code Recommendations*.

4.5.2.3 Links

- Link to source

4.5.3 Removed Tweag Traces

Severity	CVSS	Vulnerability type
None	0.0	insufficient-tests

4.5.3.1 Description

After making changes to address the Tweag audit findings, Optim team removed the Tweag traces used to identify the presence of vulnerabilities in the onchain code.

4.5.3.2 Recommendation

The Tweag traces depend upon a complex set of dependencies and a particular nix setup, which the Optim team (understandably) may not wish to maintain. We suggest that the Optim team re-implement the logic of the Tweag tests using their own testing framework to guard against regressions (and, in cases where vulnerabilities are not addressed, to serve as a reminder that they are still present in the onchain code).

5 Conclusion

During the three week period MLabs inspected the on-chain code of the Optim protocol revealing seven fixes which are not considered appropriate and six vulnerabilities and recommendations. The list is not exhaustive, containing only issues identified by the team in the limited time-frame of the audit. MLabs has summarised the findings in the given report, outlining potential dangers and offering recommendations. The report is provided without a guarantee of any kind, but we hope it proves useful.

6 Appendix

6.1 Vulnerability types

The following list of vulnerability types represents a list of commonly found vulnerabilities in Cardano smart contract protocol designs or implementations. The list of types is actively updated and added to as new vulnerabilities are found.

6.1.1 Other redeemer

ID: other-redeemer

Test: Transaction can avoid some checks when it can successfully spend a UTxO or mint a token with a redeemer that some script logic didn't expect to be used.

Property: A validator/policy should check explicitly whether the 'other' validator/policy is invoked with the expected redeemer.

Impacts:

- Bypassing checks

6.1.2 Other token name

ID: other-token-names

Test: Transaction can mint additional tokens with some 'other' token name of 'own' currency alongside the intended token name.

Property: A policy should check that the total value minted of their 'own' currency symbol doesn't include unintended token names.

Impacts:

- Stealing protocol tokens
- Unauthorised protocol actions

Example:

A common coding pattern that introduces such a vulnerability can be observed in the following excerpt:

```
vulnPolicy rmr ctx = do
  ...
  assetClassValueOf txInfoMint ownAssetClass == someQuantity
  ...
```

The recommended coding pattern to use in order to prevent such a vulnerability can be observed in the following excerpt:

```
safePolicy rmr ctx = do
  ...
  txInfoMint == (assetClassValue ownAssetClass someQuantity)
  ...
```

6.1.3 Unbounded Protocol datum

ID: unbounded-protocol-datum

Test: Transaction can create protocol UTxOs with increasingly bigger protocol datums.

Property: A protocol should ensure that all protocol datums are bounded within reasonable limits.

Impacts:

- Script XU and/or size overflow
- Unspendable outputs
- Protocol halting

Example:

A common design pattern that introduces such vulnerability can be observed in the following excerpt:

```
data MyDatum = Foo {
  users :: [String],
  userToPkh :: Map String PubKeyHash
}
```

If the protocol allows these datums to grow indefinitely, eventually XU and/or size limits imposed by the Plutus interpreter will be reached, rendering the output unspendable.

The recommended design patterns is either to limit the growth of such datums in validators/policies or to split the datum across different outputs.

6.1.4 Arbitrary UTxO datum

ID: arbitrary-utxo-datum

Test: Transaction can create protocol UTxOs with arbitrary datums.

Property: A protocol should ensure that all protocol UTxOs hold intended datums.

Impacts:

- Script XU overflow
- Unspendable outputs
- Protocol halting

6.1.5 Unbounded protocol value

ID: unbounded-protocol-value

Test: Transaction can create increasingly more protocol tokens in protocol UTxOs.

Property: A protocol should ensure that protocol values held in protocol UTxOs are bounded within reasonable limits.

Impacts:

- Script XU overflow
- Unspendable outputs
- Protocol halting

6.1.6 Foreign UTxO tokens

ID: foreign-utxo-tokens

Test: Transaction can create protocol UTxOs with foreign tokens attached alongside the protocol tokens.

Property: A protocol should ensure that protocol UTxOs only hold the tokens used by the protocol.

Impacts:

- Script XU overflow
- Unspendable outputs
- Protocol halting

6.1.7 Multiple satisfaction

ID: multiple-satisfaction

Test: Transaction can spend multiple UTxOs from a validator by satisfying burning and/or paying requirements for a single input while paying the rest of the unaccounted input value to a foreign address.

Property: A validator/policy should ensure that all burning and paying requirements consider all relevant inputs in aggregate.

Impacts:

- Stealing protocol tokens
- Unauthorised protocol actions
- Integrity

Example:

A common coding pattern that introduces such a vulnerability can be observed in the following excerpt:

```
vulnValidator _ _ ctx =
  ownInput ← findOwnInput ctx
  ownOutput ← findContinuingOutput ctx
  traceIfFalse "Must continue tokens" (valueIn ownInput == valueIn ownOutput)
```

Imagine two outputs at `vulnValidator` holding the same values

A. `TxOut ($FOO x 1 + $ADA x 2)` B. `TxOut ($FOO x 1 + $ADA x 2)`

A transaction that spends both of these outputs can steal value from one spent output by simply paying `$FOO x 1 + $ADA x 2` to the 'correct' address of the `vulnValidator`, and paying the rest `$FOO x 1 + $ADA x 2` to an arbitrary address.

6.1.8 Locked Ada

ID: locked-ada

Test: Protocol locks Ada value indefinitely in obsolete validator outputs.

Property: Protocol should include mechanisms to enable redeeming any Ada value stored at obsolete validator outputs.

Impacts:

- Financial sustainability

- Cardano halting

6.1.9 Locked non Ada values

ID: locked-nonada-values

Test: Protocol indefinitely locks some non-Ada values that ought to be circulating in the economy.

Property: Protocol should include mechanisms to enable redeeming any non-Ada value stored at obsolete validator outputs.

Impacts:

- Financial sustainability
- Protocol halting

6.1.10 Missing UTxO authentication

ID: missing-utxo-authentication

Test: Transaction can perform a protocol action by spending or referencing an illegitimate output of a protocol validator.

Property: All spending and referencing of protocol outputs should be authenticated.

Impacts:

- Unauthorised protocol actions

Example:

Checking only for validator address and not checking for an authentication token.,

6.1.11 Missing incentive

ID: missing-incentive

Test: There is no incentive for users to participate in the protocol to maintain the intended goals of the protocol.

Property: All users in the Protocol should have an incentive to maintain the intended goals of the protocol

Impacts:

- Protocol stalling
- Protocol halting

6.1.12 Bad incentive

ID: bad-incentive

Test: There is an incentive for users to participate in the protocol that compromises the intended goals of the protocol.

Property: No users of the protocol should have an incentive to compromise the intended goals of the protocol.

Impacts:

- Protocol stalling
- Protocol halting

6.1.13 UTxO contention

ID: utxo-contention

Test: The protocol requires that transactions spend a globally shared UTxO(s) thereby introducing a contention point.

Property: The protocol should enable parallel transactions and contention-less global state management if possible.

Impacts:

- Protocol stalling
- Protocol halting

6.1.14 Cheap spam

ID: cheap-spam

Test: A transaction can introduce an idempotent or useless action/effect in the protocol for a low cost that can compromise protocol operations.

Property: The protocol should ensure that the cost for introducing a salient action is sufficient to deter spamming.

Severity increases when compounded with the `utxo-contention` vulnerability.

Impacts:

- Protocol stalling
- Protocol halting

6.1.15 Insufficient tests

ID: insufficient-tests

Test: There is piece of validation logic that tests do not attempt to verify.

Property: Every piece of validator code gets meaningfully executed during tests.

Impacts:

- Correctness

6.1.16 Incorrect documentation

ID: incorrect-documentation

Test: There is a mistake or something confusing in existing documentation.

Property: Everything documented is clear and correct.

Impacts:

- Correctness
- Maintainability

6.1.17 Insufficient documentation

ID: insufficient-documentation

Test: There is a lack of important documentation.

Property: Everything of importance is documented.

Impacts:

- Comprehension
- Correctness