JPG Vesting Contract

Audit Report - Final Draft

MLabs Audit Team

December 14, 2022

Contents

1 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.*

2 Background

2.1 Scope

During the audit, MLabs Audit Team (from now on referred to as MLabs) have used the provided files for the following scope:

- **A. Audit the onchain contract**

☒ A.1. Integrate testing frameworks (Plutip and PSM) to allow reproducible testing of hypotheses.

☒ A.2. Write tests to prove the well functioning of the on-chain components.

- **B. Audit of the offchain components**

☐ B.1. Test the well functioning of the provided shell scripts.

☒ B.2. Audit the shell scripts for any malicious or not intended behaviour.

2.2 Methodology

2.2.1 Timeline

In response to the above scope, the Audit process took three (one week) sprints and it can be summarised to the following actions:

1. Review and test the onchain/offchain components against the MLabs Vulnerability types.
2. Write test scenarios for the implementations, and run some of them against a Cardano node (via Plutip).
3. Find optimisations, code quality improvements, or recommendations.
4. Capture the findings in an Audit Report.

2.2.2 Information

MLabs analysed the validators and minting scripts from the github.com/jpg-store/vesting-contract repository starting at commit `703566b`.

2.2.3 Audited Files Checksums

The following checksums are those of files captured by commit `703566b`, and were generated using the following sha256 binary:

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

The checksums are:

```
34ed...a6a6  app/Main.hs
6ef0...aa6c  src/Canonical/Shared.hs
ee50...1886  src/Canonical/DebugUtilities.h
add6...b689  src/Canonical/Vesting.hs
```

2.2.4 Audit Report

The audit report is an aggregation of issue, tickets and pull-requests created in the jpg-store/vesting-contract repository.

3 Findings

3.1 Summary

The Audit revealed one `unbounded-protocol-datum` vulnerability. With this exception, there were no other vulnerabilities found during the Audit process.

Furthermore, the audit puts forward the following optimisations / recommendations:

- *Coding Standards - Linting*
- *Coding Standards - Formatters*
- *Optimisation - Calculating the Total Unvested Value*

3.2 Input Datum - Unbounded List

3.2.0.1 Description

The Input datum of the vesting contract is the product of two unbounded lists of beneficiaries and schedule. Due to the unbounded nature of the datum, it is possible to make arbitrary long vesting schedules for number of arbitrary beneficiaries, however doing so may result in permanently locking the native tokens and ADA in the vesting contract. This is because when contract is executed while spending it, it exceeds the resource limit which is imposed on the transaction.

3.2.0.2 How To Reproduce

Checkout branch audit/unbounded-datum and run:

```
$ cabal run vesting-tests
```

3.2.0.3 Expected behaviour

The values locked in the contract should not be locked permanently.

3.2.0.4 Error

While spending the contract it results in the following error:

```
ContractExecutionError "WalletContractError (OtherError \"ScriptFailure
(ScriptErrorEvaluationFailed
(CekError An error has occurred:  User error:
The machine terminated part way through evaluation due to overspending the budget.\
The budget when the machine terminated was:
({ cpu: 7057872449
| mem: -852\
})
Negative numbers indicate the overspent budget; note that this only indicates the budget that was
needed for the next step, not to run the program to completion.) [])\")"
```

3.2.0.5 Provided proof

The following plutip test demonstrate this vulnerability: unboundedDatum.

3.3 Coding Standards - Linting

3.3.1 Description

Use shellcheck to perform static analysis of the `shell` scripts used in the offchain of the protocol to find optimisations, and underlying bugs. Use `hlint` to lint the Haskell code.

3.3.2 Vulnerabilities

1. Please refer to ./docs/audit/shell-check-report.txt for a detailed list of all the bugs, vulnerabilties, and optimisations. Note that the `shellcheck` analysis is based on the assumption that the scripts are intended to be interpreted via `/bin/bash`.
2. Please refer to ./docs/audit/hlint-check-report.txt for linting suggestions.

3.4 Coding Standards - Formatters

3.4.1 Description

We recommend the use of standardising code format to minimise change noise, and easier long term maintenance of the code.

3.4.2 Recommendation

Make use of tools like: - shfmt - fourmolu

MLabs team provides the currently open PR 5 as a POC.

3.5 Optimisation - Calculating the Total Unvested Value

In the current version, we have to iterate the `Portion` list multiple times to calculate the total unvested value:

1. Filter the unvested `Portion` from the given `Schedule`
2. Iterate again to get `amount` from the filtered `Portion`
3. Iterate again to concat all the `Value`(s) present in the list.

```
unvested :: Value
unvested = mconcat . fmap amount . filter (not . isVested) . schedule $ datum
```

The same thing can be accomplished using `foldr` which only requires a single iteration of the list.

```
unvested :: Value
unvested = foldr (\portion totalAmt ->
                    if not (isVested portion)
                    then amount portion <> totalAmt
                    else totalAmt
                  ) mempty
         . schedule $ datum
```

4 Testing Summary

For report brevity (and clarity), the testing framework and associated PRs will be mentioned here, and not included in the Audit Report. For in-depth explanation about what each PR is aiming to prove, please refer to the linked ticket description on GitHub.

Therefore, we would first mention the tickets outlining the hypotheses that the tests were trying to prove/disprove:

- *Vesting Contract Onchain Hypotheses*,
- *Vesting Contract Offchain Hypotheses*.

Secondly, we would like to mention the PRs proposing the tests to be integrated into the main repository.

- *Test - Not Enough Signatures*,
- *Test - Empty Beneficiaries on Input*,
- *Test - Early Withdraw*,
- *Test - Empty Beneficiaries on Withdraw*,
- *Test - Multi User Vesting*,
- *Test - Happy Path Test Generator*.

We recommend for the aforementioned tests to be included in the main branch and made visible in the CI.

5 Conclusion

MLabs inspected the onchain and offchain code of the *JPG Vesting Contract* over a three week period and discovered one vulnerability and three recommendations. Additionally, MLabs has provided a testing framework and tests written to verify the claims made in the report (available on GitHub). This list is not exhaustive, as the team only had a limited amount of time to conduct the audit.

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

6.1.18 Poor Code Standards

Test: Missing the use of code quality and stadardisation tools.

Property: Code is properly formatted, linted, and uses an adequate code standard.

Impacts:

- Codebase Maintainability
- Comprehension