Fracada Audit

Audit Report

MLabs Audit Team

January 12, 2023

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 Background

2.1.1 Scope

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

- **Audit the onchain scripts**

2.1.2 Methodology

2.1.2.1 Timeline

In response to the above scope, the Audit started in June 2022 with a two weeks period after which an initial set of findings were informally reported to the Fracada team. The intention behind the informal report was to maximise the utility of the hours spent auditing the scripts, by having the initial findings remediated before continuing with the audit.

Once a set of audit findings remediations were provided, the audit resumed in December 2022. Given the scope of the changes done to address the initial findings, with significant changes to the project's architecture, it was not feasible to properly audit the new design and implementation. Therefore, a mixed approach has been taken, formalising the initial findings and briefly checking the new implementation, with the main purpose of acknowledging that work has been done to address the findings.

2.1.2.2 Information

MLabs analysed the validators and minting scripts from the github.com/dcSpark/fracada-il-primo repository starting at commit `0400e8f`.

It also informally reviewed PR 5 of dcSpark/fracada-il-primo repository starting at commit `a5e8c69`, with the purpose of acknowledging work towards addressing the findings.

2.1.2.3 Audited Files Checksums

The following checksums are those of files captured by commit `0400e8f`, and were generated using the following sha256 binary:

```
$ sha256sum --version
sha256sum (GNU coreutils) 8.32
```

The checksums are:

```
5731...6c3f  src/Fracada/Minting.hs
f839...ed6a  src/Fracada/Utils.hs
491f...a704  src/Fracada/Validator.hs
```

2.1.2.4 Audit Report

The audit report is an aggregation of issues and pull-requests created in the mlabs-haskell/fracada-audit repository.

2.1.2.5 Metrics

2.1.2.5.1 CVSS

In an effort to use a standardised metric for the severity of the vulnerabilities, the open standard Common Vulnerability Scoring System, together with the NVD Calculator, has been leveraged. 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.

2.1.2.5.2 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 Findings

3.1 Summary

The Audit revealed the following five High Severity vulnerabilities:

- *Initial mint of more fraction tokens than declared in the FractionNFTDatum*
- *Negative totalFractions value in FractionNFTDatum allows minting of more fraction tokens*
- *No check to enforce the use of legit fraction token minting policy leads to fraction tokens being leaked*
- *Same fraction token AssetClass can be used for different UTxOs locked in the validator*
- _ Signatures are not unique_

It also revealed the following two Medium Severity vulnerabilities:

- *Use of malicious minting policy for the fraction token*
- *Double statisfaction attack can happen when fungible tokens are being locked in the validator*

The two Medium Severity vulnerabilities can lead to a total loss of funds, but they have been catalogued as Medium severity because they can be avoided by following some basic security guidelines. In a sense, and following the CVSS nomenclature, User Interaction (by not following the guidelines) is needed for the exploit to be successful.

Along each vulnerability, a test has been provided as a proof. The test can be reproduced by checking out to the branch of the audit repository reporting the specific vulnerability and running:

```
cabal run fracada-test
```

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

- *Lack of technical specification*
- *Lack of clarity in the code base*
- *Bad state transitions allowed*
- *There is no limit to the number of NFTs that can be added to a bag*

MLabs acknowledges that work has been done to fix all the vulnerabilities mentioned in this report. A preliminary analysis of the updated implementation suggests that a remediation has been provided for each reported vulnerability. However, the new implementation has not been thoroughly analysed. We recommend that a proper audit for the new implementation takes place, as new vulnerabilities might have been introduced in the process of fixing the reported ones.

3.2 Vulnerabilities

3.2.1 Initial mint of more fraction tokens than declared in the FractionNFTDatum

Severity	CVSS	Vulnerability type
High	8.5	incorrect-logic

3.2.1.1 Description

When locking an NFT in the `fractionValidatorScript` script for the first time, it is enough to include any input with datum (could just come from an always succeed validator) to bypass the mintDeclared check in the minting policy. So one could mint 100 fraction tokens and just record 2 in the datum.

Example steps to a successful exploit:

1. Attacker sends some ADA to an always succeed validator (creating UTxO A).
2. Attacker builds a transaction that consumes UTxO A, locks an NFT in the `fractionValidatorScript` script recording `totalFractions` = 2 and mints 100 corresponding fraction tokens.

Note: to make it more opaque, the attacker could continue the attack by redeeming the NFT and starting the fractionalisation again, this time doing it the correct way.

3.2.1.2 Expected Behaviour

The extra minting of fraction tokens should not be allowed.

3.2.1.3 Provided proof

The mentioned emulator test that exploits the vulnerability and its transaction spec.

3.2.1.4 Status

MLabs acknowledges that work has been done to fix this vulnerability in PR 5 of the fracada-il-primo repository. Specifically, instead of checking that there are no inputs to the transaction carrying a datum as a proxy for knowing whether the current mint is the initial one, different minting policy redeemers are used for the initial mint and for subsequent mints. The redeemer used for the initial mint verifies that a validity token (unique for each fraction asset) is minted and locked in the Fracada validator, and the redeemer used for subsequent mints verifies that said validity token is present in the transaction input coming from the Fracada validator.

3.2.2 Negative totalFractions value in FractionNFTDatum allows minting of more fraction tokens

Severity	CVSS	Vulnerability type
High	8.5	incorrect-logic

3.2.2.1 Description

`FractionNFTDatum` allows `totalFractions` value to be negative. Therefore, when locking a UTxO in the `fractionValidatorScript` script, it is possible to set the value of `totalFractions` to a negative number. This leads to the minting of more fraction tokens than registered in the `FractionNFTDatum`.

Example steps to a successful exploit:

1. Attacker locks an NFT in the `fractionValidatorScript` script, recording `totalFractions` = -1 without minting any fraction tokens.
2. Attacker consumes the NFT locked in the script and mints 2 fraction tokens, updating `totalFractions` to 1.

Note: to make it more opaque, the attacker could now redeem the NFT and start the fractionalisation again, this time doing it the correct way.

3.2.2.2 Expected Behaviour

Both the negative value in `totalFractions` and the extra minting of fraction tokens should not be allowed.

3.2.2.3 Provided proof

The mentioned emulator test that exploits the vulnerability and its transaction endpoints:

1. corruptDatum
2. mintExtraTokens

3.2.2.4 Status

MLabs acknowledges that work has been done to fix this vulnerability in PR 5 of the fracada-il-primo repository. Specifically, the minting of a fraction token requires the presence of a validity token, which is minted using the same minting policy as the one for the fraction token and ensures a correct initialisation of the NFT bag.

3.2.3 No check to enforce the use of legit fraction token minting policy leads to fraction tokens being leaked

Severity	CVSS	Vulnerability type
High	7.6	missing-utxo-authentication

3.2.3.1 Description

There is no check in the fraction token minting policy to enforce that `FractionNFTDatum` records its own currency symbol as the one corresponding to the policy to mint further fraction tokens. This leads to valid fraction tokens being leaked, as a malicious user can mint fraction tokens when locking an NFT in the `fractionValidatorScript` script while recording a malicious minting policy as the policy for minting further fraction tokens. A buyer that checks that both the script where the NFT bag is locked and the recorded asset class are correct, but does not scan the blockchain to check for pre-minted fraction tokens is vulnerable to scams.

Example steps to a successful exploit:

1. Attacker locks the NFT in the `fractionValidatorScript` script, minting 10 legit fraction tokens, but recording a fraudulent asset class for the fraction token.
2. Attacker mints 5 tokens from the fraudulent minting policy.
3. Attacker redeems the NFT by burning the 5 fraudulent fraction tokens.
4. Attacker locks the NFT in the `fractionValidatorScript` script, minting 2 legit fraction tokens and recording the correct asset class for the fraction tokens.

3.2.3.2 Expected Behaviour

There should be some kind of check to enforce that valid currency symbol is recorded in `FractionNFTDatum`.

3.2.3.3 Provided proof

The mentioned emulator test that exploits the vulnerability and it's transaction spec.

3.2.3.4 Status

MLabs acknowledges that work has been done to fix this vulnerability in PR 5 of the fracada-il-primo repository. Specifically, all fraction tokens share the same minting policy, which ensures the correct initialisation of an NFT bag. Therefore, a user would just need to check that the fraction token has the correct currency symbol, which is a much more reasonable security measure to expect from users.

3.2.4 Same fraction token AssetClass can be used for different UTxOs locked in the validator

Severity	CVSS	Vulnerability type
High	7.7	infinite-minting

3.2.4.1 Description

In the Fracada protocol, it's possible to have the same fraction asset class for two different UTxOs holding two different NFT bags.

Example steps to a successful exploit:

1. Attacker locks two different UTxO(s) that has same `FractionNFTParameters` and same fraction token name.
2. Attacker mints 10 fraction tokens while locking each UTxO, hence attacker has 20 fraction token.
3. Attacker deposits an NFT with high value in one of the created UTxO, increasing the value of the fraction token.
4. Attacker sells 10 fraction tokens. `FractionNFTDatum` of the UTxO that has the NFT has a recorded value of only 10 minted fraction tokens. But the attacker has 20 such fraction tokens, hence a buyer who does not check for the existence of extra fraction tokens is vulnerable to the scam.

3.2.4.2 Expected Behaviour

A duplicate fraction token should not be allowed.

3.2.4.3 Provided proof

The mentioned emulator test that exploits the vulnerability and the assertion after running the emulator trace.

3.2.4.4 Status

MLabs acknowledges that work has been done to fix this vulnerability in PR 5 of the fracada-il-primo repository. Specifically, all fraction tokens share the same currency symbol, which comes from a minting policy that ensures that all token names are unique for each initialisation of an NFT bag. The uniqueness of the token name is guaranteed by using as the name the hash of the reference of a UTxO and ensuring that the referenced UTxO is consumed by the transaction.

3.2.5 Signatures are not unique

Severity	CVSS	Vulnerability type
High	8.9	incorrect-logic

3.2.5.1 Description

To verify legit transactions when minting extra fraction tokens or adding new NFTs to the bag, it is checked that the datum of the continuing output (which contains the last added asset class and the number of fractions in circulation) is signed by a minimum number of authorised parties. These messages are not unique, so the signature could be reused by anyone in the future.

Example steps to a successful attack:

1. Owner of the NFT locks it in a script with the owner's key as the only one authorised, sets `totalFractions` = 2, and mints two fraction tokens.
2. Owner mints an extra fraction. To authorise it, includes as a redeemer the signature to the message: `Datum $ FractionNFTDatum {tokensClass = myAsset, totalFractions = 3, newNftClass = myAsset}`.
3. Owner redeems the NFT, burning the 3 tokens and unlocking the NFT.
4. Owner wants to fractionalise the NFT again, so owner repeats step 1.
5. Attacker mints a fraction using the signature submitted in step 2 in a redeemer.

3.2.5.2 Expected behaviour

A signature should uniquely identify a single action being authorised by signing the message in a way that cannot be reused in the future.

3.2.5.3 Provided proof

The mentioned emulator test that exploits the vulnerability and the assertion after running the emulator trace.

3.2.5.4 Status

MLabs acknowledges that work has been done to fix this vulnerability in PR 5 of the fracada-il-primo repository. Specifically, instead of requiring signatures for output datum, which are not unique, signatures for the whole transaction are required, which are unique.

3.2.6 Use of malicious minting policy for the fraction token

Severity	CVSS	Vulnerability type
Medium	6.1	missing-utxo-authentication

3.2.6.1 Description

There is no check in the Fracada protocol to enforce that the `mintFractionTokensPolicy` minting policy is used for the fraction tokens when locking an NFT at the `fractionValidatorScript` script. This can result in the use of a malicious minting policy for fraction tokens that a malicious user has complete control over. A buyer that trusts that all NFTs locked in the legit script are protected by the Fracada protocol is vulnerable to scams. In order to verify legitimacy of a fraction token, the buyer would need to check that the asset class of the

fraction token corresponds to a version of the Fracada minting policy after applying the relevant arguments, which might be non-trivial for many users.

Example steps to a successful exploit:

1. Attacker locks the NFT in the `fractionValidatorScript`, recording a fraudulent asset class for the fraction tokens.
2. Attacker has complete control over the minting and burning of fraction tokens, so attacker mints 100 tokens without recording it in the datum of the UTxO locking the NFT.

3.2.6.2 Expected Behaviour

There should be some kind of check to ensure that the desired minting policy is used for fraction tokens.

3.2.6.3 Provided proof

The mentioned emulator test that exploits the vulnerability and it's transaction spec.

3.2.6.4 Status

MLabs acknowledges that work has been done to fix this vulnerability in PR 5 of the fracada-il-primo repository for this vulnerability. Specifically, all fraction tokens share the same minting policy, which ensures a correct initialisation of an NFT bag. Therefore, a user would just need to check that the fraction token has the correct currency symbol, which is a much more reasonable security measure to expect from users.

3.2.7 Double statisfaction attack can happen when fungible tokens are being locked in the validator

Severity	CVSS	Vulnerability type
Medium	6.1	multiple-satisfaction

3.2.7.1 Description

A double satisfaction attack can be used to steal funds from the UTxOs locked in the `fractionValidatorScript` script when fractionalising fungible tokens.

Example steps to a successful attack:

1. Owner of a fungible token locks it in the `fractionValidatorScript` script, minting 5 fraction tokens.
2. (Optional) An extra NFT is added to the bag.
3. Attacker locks the same fungible token in the script, minting 5 fraction tokens.
4. Attacker redeems the fraction token and the NFT by burning the 5 fraction tokens

This vulnerability is a consequence of *Same fraction token AssetClass can be used for different UTxOs locked in the validator* vulnerability. It can be avoided by not locking fungible tokens in Fracada.

3.2.7.2 Expected Behaviour

Double statisfaction attack should not be allowed even when locking fungible token in the `fractionValidatorScript` validator.

3.2.7.3 Provided proof

The mentioned emulator test that exploits the vulnerability, it's transaction spec and the assertion after running the said emulator trace.

3.2.7.4 Status

MLabs acknowledges that work has been done to fix this vulnerability in PR 5 of the fracada-il-primo repository. Specifically, it is checked that in each transaction there is only one transaction input coming from the Fracada script.

3.3 Recommendations

3.3.1 Lack of technical specification

Severity	CVSS	Vulnerability type
None	0.0	insufficient-documentation

3.3.1.1 Description

There is no technical specification describing the workings of the Fracada validator and minting policy. The comments provided throughout the code and the README are not enough to have a full picture of how the protocol is intended to work or what assumptions are being made.

3.3.2 Lack of clarity in the code base

Severity	CVSS	Vulnerability type
None	0.0	poor-code-standards

3.3.2.1 Description

1. There are few instances of code duplicity. For example, findDatumTyped is defined in `Fracada.Minting` module but is re-written in the `Fracada.Validator` module again instead of using the already existing function.
2. In addition, there are no code standards in place, which increases code noise and affects the maintainability of the code base:
   - Some unnecessary duplicity, like checking `nftc'` both with `assetClass` and `unAssetClass`.
   - No setup for proper linter and formatter in the code base.
3. Setting the `newNFTClass` to the fraction tokens asset class (and not an NFT) when minting more fraction tokens for an NFT bag.

3.3.3 Bad state transitions allowed

Severity	CVSS	Vulnerability type
None	0.0	incorrect-logic

Some of the state transitions in the fracada validator do not make sense and can result in unintented behavior. Therefore, it would be beneficial to assert for those transitions and prevent them from occurring.

- Fraction tokens can be added to the UTxO locked at the `fractionValidatorScript` script.
- The locked NFT at the `fractionValidatorScript` script cannot be redeemed if no fraction tokens have been minted.

3.3.4 There is no limit to the number of NFTs that can be added to a bag

Severity	CVSS	Vulnerability type
None	0.0	unbounded-protocol-value

3.3.4.1 Description

There are no limits to the number of NFTs that can be added to one UTxO locked in the `fractionValidatorScript` script. While in the Fracada protocol this should not lead to a vulnerability, it can result in unexpected errors when hitting the network limits.

4 Conclusion

MLabs inspected the onchain code of Fracada in two phases. In the first phase the findings were informally reported and in the second phase they have been formalised. A total of seven vulnerabilities have been found and reported, as well as additional recommendations. Additionally, MLabs has written tests 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.

Additionally, MLabs did a preliminary inspection of the new implementation of Fracada, which aims to remediate the reported vulnerabilities. MLabs acknowledges that work has been done towards fixing all the reported vulnerabilities. However, we advise on pursuing a full audit for the new implementation, as it has significant architecture changes.

5 Appendix

5.1 Vulnerability types

The following list is a description of the common vulnerability types found during this audit.

5.1.1 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.

5.1.2 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.

5.1.3 Infinite minting

ID: infinite-minting

Test: Tokens can only be minted in the specific situations where they are intended to be minted."

Property: A protocol should ensure that no extra tokens can be minted.

Impacts:

- Stealing protocol tokens
- Unauthorised protocol actions

5.1.4 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

5.1.5 Insufficient documentation

ID: insufficient-documentation

Test: There is a lack of important documentation.

Property: Everything of importance is documented.

Impacts:

- Comprehension
- Correctness

5.1.6 Poor Code Standards

ID: 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