ANASTASIA LABS

Security Audit Report

Date: July 16, 2024
Project: Ecosystem Marketplace
Version 1.0

ANASTASIA LABS

Contents

Disclosure

This document contains proprietary information belonging to Anastasia Labs. Duplication, redistribution, or use, in whole or in part, in any form, requires explicit consent from Anastasia Labs.

Nonetheless, both the customer Empowa and Anastasia Labs are authorized to share this document with the public to demonstrate security compliance and transparency regarding the outcomes of the Protocol.

Disclaimer and Scope

A code review represents a snapshot in time, and the findings and recommendations presented in this report reflect the information gathered during the assessment period. It is important to note that any modifications made outside of this timeframe will not be captured in this report.

While diligent efforts have been made to uncover potential vulnerabilities, it is essential to recognize that this assessment may not uncover all potential security issues in the protocol.

It is imperative to understand that the findings and recommendations provided in this audit report should not be construed as investment advice.

Furthermore, it is strongly recommended that projects consider undergoing multiple independent audits and/or participating in bug bounty programs to increase their protocol security.

Please be aware that the scope of this security audit does not extend to the compiler layer, such as the UPLC code generated by the compiler or any areas beyond the audited code.

The scope of the audit did not include additional creation of unit testing or property-based testing of the contracts.

Assessment overview

From April 16th, 2024 to July 16, 2024, Empowa engaged Anastasia Labs to evaluate and conduct a security assessment of its Ecosystem Marketplace protocol. All code revision was performed following industry best practices.

Phases of code auditing activities include the following:

- Planning – Customer goals are gathered.
- Discovery – Perform code review to identify potential vulnerabilities, weak areas, and exploits.
- Attack – Confirm potential vulnerabilities through testing and perform additional discovery upon new access.
- Reporting – Document all found vulnerabilities.

The engineering team has also conducted a comprehensive review of protocol optimization strategies.

Each issue was logged and labeled with its corresponding severity level, making it easier for our audit team to manage and tackle each vulnerability.

Assessment components

Manual revision

Our manual code auditing is focused on a wide range of attack vectors, including but not limited to.

- UTXO Value Size Spam (Token Dust Attack)
- Large Datum or Unbounded Protocol Datum
- EUTXO Concurrency DoS
- Unauthorized Data modification
- Multisig PK Attack
- Infinite Mint
- Incorrect Parameterized Scripts
- Other Redeemer
- Other Token Name
- Arbitrary UTXO Datum
- Unbounded protocol value
- Foreign UTXO tokens
- Double or Multiple satisfaction
- Locked Ada
- Locked non Ada values
- Missing UTXO authentication
- UTXO contention

Executive summary

Empowa is a unique decentralised property development and digital collectables platform seeking to harness the power of community to empower people mostly excluded from the financial system. This exclusion may come about from a lack of available funding or lack of access due to gender, financial history, poverty and other forms of discrimination.

Empowa focuses on:

- Economic empowerment, initially through property
- Connecting and empowering people to create a fairer economic system
- Helping to deliver affordable housing to poor and middle income people
- Community building
- Women empowerment
- Environmental impact

The Empowa platform leverages a proprietary technology, namely Empowa Pay, Empowa Trade, digital SDRIs (Secure Defined Return Instrument) and the EMP digital token, to enable financial inclusion through the provision of rent-to-own home loans.

This audit relates to part of the Empowa Trade product.

Code base

Repository

https://github.com/empowa-io/ecosystem-marketplace

Commit

d9d45981fc800f94a2e7302fd9c99098219bb562

Files audited

SHA256 Checksum	Files
ac4f47027ea528628f882f06756e4b175f4d365ef3a162e0df8fe04cb5700dbd	src/utils/pvalueOf.ts
fc698602440c6dbd0706f511080c26911bad377e60bd2c321b24ae200c260e80	src/contracts/feeOracleNftIdPolicy.ts
eafdeb9dbaf379ffa62ca4632976b292a3163277f96875a8a771f9f8d08647ab	src/contracts/feeOracle.ts
bfb5b2ceca8851eb71ea70f1f2e6edb042dad511fd99b7476f7443bfb3937c15	src/contracts/marketplace.ts

Severity Classification

- **Critical**: This vulnerability has the potential to result in significant financial losses to the protocol. They often enable attackers to directly steal assets from contracts or users, or permanently lock funds within the contract.
- **Major**: Can lead to damage to the user or protocol, although the impact may be restricted to specific functionalities or temporal control. Attackers exploiting major vulnerabilities may cause harm or disrupt certain aspects of the protocol.
- **Medium**: May not directly result in financial losses, but they can temporarily impair the protocol's functionality. Examples include susceptibility to front-running attacks, which can undermine the integrity of transactions.
- **Minor**: Minor vulnerabilities do not typically result in financial losses or significant harm to users or the protocol. The attack vector may be inconsequential or the attacker's incentive to exploit it may be minimal.
- **Informational**: These findings do not pose immediate financial risks. These may include protocol optimizations, code style recommendations, alignment with naming conventions, overall contract design suggestions, and documentation discrepancies between the code and protocol specifications.

Finding severity ratings

The following table defines levels of severity and score range that are used throughout the document to assess vulnerability and risk impact.

	Level	Severity	Findings
🟥	5	Critical	0
🟥	4	Major	1
🟧	3	Medium	0
🟨	2	Minor	3
🟦	1	Informational	5

Findings

ID-401 Token Dust Attack

	Level	Severity	Status
🟥	4	Major	Resolved

Description

At `feeOracle.ts`, since the continuing output is only validated via `pvalueOf`, it can be subject to being filled with random tokens which can increase the transaction fees of subsequent transactions, and even brick the UTxO.

This is a common vulnerability known as "Token Dust Attack."

Recommendation

It is recommended to check UTxO's value consists of exactly 2 assets, one of which is Ada, and the other is the authentication NFT.

Note that ordering of assets is also lexicographic (similar to input UTxOs). However this might depend on the framework (`plu-ts` in this case) and how it gives access to assets and/or flattens a value.

It is also not advised to validate the Lovelace count in this instance as its minimum amount depends on network's protocol parameters.

Resolution

Resolved in commit `4bac97d`, continuing output is now forced to carry only two policy IDs (one of which is Ada). Along with validation for presence of the beacon NFT, token dust attacks are now fully prevented.

ID-201 Oracle is Controlled by a Single Signature

	Level	Severity	Status
🟨	2	Minor	Acknowledged

Description

At line 23 of `feeOracle.ts` updating datum of the oracle UTxO is permitted by `owner`, which can be considered a single point of failure, i.e. if access to `owner` wallet is lost, fee value is locked permanently.

Recommendation

Switch the authorization to the presence of a specified NFT. This way the NFT can be stored in a multi-sig wallet, which alleviates the oracle from having a single point of failure.

Resolution

Acknowledged

ID-202 Staking Control

	Level	Severity	Status
🟨	2	Minor	Resolved

Description

Whenever a validation needs to check where a UTxO is coming from (or going to), an important consideration is the staking part of the address. If only the payment part of the address is checked, it can potentially allow the staking part of the UTxOs to be changed arbitrarily.

We spotted 4 occurrences:

1. `ownOutToSelf` at `feeOracle.ts` does not check the staking part of the continuing output (line 61). This, however, is quite minor as the locked Ada is small.
2. `NFTSale` datum only stores the seller's public key hash, and therefore doesn't allow any validations on the staking part (line 8).
3. Point 2 also applies to `owner`, a parameter to `marketplace.ts` (line 37).
4. Continuing output of `marketplace.ts` is also prone to this vulnerability (`validOutAddress`).

Recommendation

In all cases, consider implementing validations for staking parts, i.e. instead of checking the payment part, check the whole address.

Resolution

All findings resolved in commit `1ef3b1f`.

ID-203 Excessive Restriction on Inputs and Outputs

	Level	Severity	Status
🟨	2	Minor	Resolved

Description

The `Update` endpoint of `marketplace.ts` requires exactly one input and one output to the transaction (lines 77 and 80). This means that the single input UTxO has to provide the fee to the transaction, and therefore be reproduced with less Ada.

Performing this action multiple times can deplete the UTxO up to a point where there won't be enough Ada to cover the minimum required in the reproduced UTxO.

Recommendation

Instead of making this restriction on all the inputs and outputs, impose it on the continuing inputs and outputs, i.e. ensure a single UTxO from the script is getting spent and only one UTxO is reproduced.

Resolution

Resolved in commit `8c96457`.

ID-101 Redundant Redeemer

	Level	Severity	Status
🟦	1	Informational	Acknowledged

Description

Since Cardano transactions contain old state, update message (i.e. redeemer), and the new state, in logics where simply a new value should replace its older counterpart, providing the new value by the redeemer is redundant and can be directly specified as the new state.

In our review we encountered two instances of this form of redundancy:

1. In the single endpoint of `feeOracle.ts`.
2. In the `Update` endpoint of `marketplace.ts`.

Recommendation

By omitting the new value from the redeemers of these two endpoints, their corresponding transactions can have smaller sizes.

As a consequence, required validations for the new values must be performed after grabbing them from the updated datums.

Resolution

Acknowledged

ID-102 Costly Value Lookups

	Level	Severity	Status
🟦	1	Informational	Acknowledged

Description

Input and output UTxOs in Cardano transactions typically depend upon the states of users' wallets. Therefore employing functions that their execution cost varies based on UTxO arrangements are not advisable.

We have spotted 4 occurrences:

1. In `feeOracle.ts`, the value of input beacon UTxO is validated via `getNftQty` which is a partial application of `pvalueOf` (line 40).
2. In `feeOracle.ts`, the continuing output is identified using `getNftQty` (line 54). However, this becomes moot if the recommendation at "ID-103 Inefficient List Traversals" is followed.
3. In `marketplace.ts`, validation for payout to seller is done by `paidAmtToHash` which uses `pvalueOf` under the hood (line 176).
4. Similar to point 3, owner's fee payout is validated via `paidAmtToHash` (line 187).

Recommendation

The general solution for this potential issue is to be a bit more restrictive on allowed values. However this is not applicable to every instance.

Here are our recommended alternatives for each of the cases:

1. Expect the input value to have exactly 2 tokens, one of which is going to be Ada. Note that the amount of Ada should NOT be validated as it may depend on protocol parameters.
2. Please refer to "ID-103 Inefficient List Traversals".
3. Expect the payout UTxO at a specific output index (either hardcoded, or specified via the redeemer). This allows a sort of decoupling from wallets.
4. Similar to 3.

Resolution

Acknowledged

ID-103 Inefficient List Traversals

	Level	Severity	Status
🟦	1	Informational	Acknowledged

Description

Using functions like `find` and `filter`—which can perform arbitrary logics on list elements as they traverse—can have a negative impact on the required execution budgets.

While using these functions can sometimes be inevitable, we have spotted a few instances where we believe these traversals can be optimized:

1. `ownInput` in `feeOracle.ts`, optimization benefit here is quite small (lines 25 to 38).

2. `ownOut` in `feeOracle.ts`, this is costlier than `ownInput` as the traversal logic goes through the value field of each element it checks (lines 52 to 58).

3. `Buy` endpoint of `marketplace.ts`, finding oracle UTxO among the reference inputs (lines 123 to 125).

Recommendation

The general solution for this kind of logic is to either provide the index of the intended element via the redeemer, or expecting the element at an hardcoded index.

Here are our suggested solutions to optimize the 3 instances:

1. Provide the index of `ownInput` in the redeemer (if the recommendation at "ID-101 Redundant Redeemer" is followed, it frees up the redeemer to be used for this index).

2. Expect `ownOut` at index `0` of the outputs, make sure it's going back to the script (preferably validating the whole address, point 1 of "ID-202 Staking Control" delves deeper into this aspect), and finally expect it to have exactly 1 asset (the beacon NFT) apart from its Lovelaces (much like the recommended solution for point 1 from "ID-102 Costly Value Lookups").

3. Provide the index of the beacon UTxO in the redeemer.

Note that further validations are required to make sure the element at specified index is in fact the intended one.

Also, both inputs and reference inputs are lexicographically ordered based on their transaction hashes first, and second on their output indices. This ordering needs to be done beforehand in the off-chain code in order to find the proper indices.

Resolution

Acknowledged

ID-104 Lack of a Minimum Fee for Owner

	Level	Severity	Status
🟦	1	Informational	Resolved

Description

Since owner fee at the `Buy` endpoint of `marketplace.md` is found as a portion of the listed price, there is a possibility for this value to fall below the minimum required Ada for a UTxO. This prevents invocation of this endpoint for UTxOs with low prices.

Recommendation

As the low price of a listing can be updated or reclaimed, a convenient solution is to set a minimum fee to reduce the probability of this failure for users.

Resolution

Addressed in commit `4bac97d`, the new logic makes the fee payment optional if the datum received from `feeOracle` is set to zero.

ID-105 Incorrect Imports

	Level	Severity	Status
🟦	1	Informational	Resolved

Description

Utility module `pvalueOf.ts` has imports that are not exposed by `plu-ts`, namely `PAssetsEntryT` and `PValueEntryT`.

Recommendation

Define the required `plu-ts` types locally.

Resolution

Resolved in commit `61710b3`.