



ANASTASIA LABS

Cerra.io Security Audit Report

Initial Report - Treasury Validators

Date July, 2024
Project Cerra.io - Treasury
Version 0.9

Contents

Disclosure	1
Disclaimer and Scope	2
Assessment overview	3
Assessment components	4
Code base	5
Repository	5
Commit	5
Files Audited	5
Severity Classification	6
Finding severity ratings	7
Findings	8
ID-501 Unrestricted Treasury Withdrawal	9
ID-401 Reduced Staking Rewards	10
ID-402 Unspendable Staked Tokens	11
ID-301 User Reward Deprivation	12
ID-302 Excessive Restriction on Value of Staking UTxOs	13
ID-101 Inefficient Staking UTxO Lookup	14
ID-102 Misleading Variable Names	15
ID-103 Extra Traversal of Inputs	16

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 Cerra.io 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 June, 2024 to July, 2024, Cerra.io engaged Anastasia Labs to evaluate and conduct a security assessment of its treasury validators. 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

Code base

Repository

<https://github.com/cerraio/treasury>

Commit

5090f4aa12b1bbca949f254dd79405e117d064af

Files Audited






SHA256 Checksum	Files
1bc1ad049942b32787806706b0bce4f4df98a89ca9a361dd31215122fe7b5a50	src/Cerra/FactoryFT/OnChain.hs
2cd4a3aac14489026e2aa8e08561f26a74ee8cb38ca822046edf8903816301b	src/Cerra/FactoryFT/Types.hs
9959bbf7d69cd172fd5b4bf5ce965a104b2d469d8865a6b398ac3e8e17f7dfc0	src/Cerra/Staking/OnChain.hs
3e474e2cb2c9ad20565d9fbab33b6997bc96e49eb4282257b44c8594abc2f30f	src/Cerra/Staking/Types.hs
b1f006dfd1fe4585a9b46f829fc0c3ee8b7e98ac9fc220edf9f9de5ff16fb11b	src/Cerra/Treasury/OnChain.hs
6ec5d24ac254045f132530c8a36881a5d81d934cb227231d1455519b0c3eb434	src/Cerra/Treasury/Types.hs
c19da33e5a8ed8108517206e8ef22e8f8ad481e6fbb615950efff676dd9b7906	src/Cerra/Utils/Utils.hs

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	1
	4	Major	2
	3	Medium	2
	2	Minor	0
	1	Informational	3

Findings

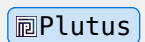
ID-501 Unrestricted Treasury Withdrawal

	Level	Severity	Status
	5	Critical	Pending

Description

The only requirement for spending from the treasury is that a single factory token is getting spent, indirectly ensuring staking script's validation is also present.

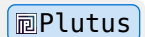
```
1 if assetClassValueOf assets (assetClass tpFactoryFTSymbol  
   tpStakingTokenName) == 1  
2 then ()  
3 else error()
```



However, there is no checks for which endpoint of the staking script (`Unstake` or `Claim`) is invoked. This leaves the door open for arbitrary withdrawals of treasury UTxOs by simply staking some CERRA, and then submitting transactions that spend multiple treasury UTxOs while unstaking the CERRA tokens.

Treasury validates the presence of a factory token and therefore permits the spends, while the staking script validates the unstaking logic of that single UTxO.

```
1 validateUnstake :: ScriptContext -> Bool  
2 validateUnstake ctx = ...
```



Recommendation

A computationally cheap solution is for treasury spends to also make sure no minting/burning are happening in the transaction.

A more explicit—and perhaps future-proof—solution would be to look for the corresponding redeemer of the UTxO containing the factory token, and making sure that it is indeed `Claim`.

ID-401 Reduced Staking Rewards

	Level	Severity	Status
	4	Major	Pending

Description

Currently user reward distribution transaction can be submitted by anyone with as many UTxOs as is required (or is limited by the ledger) to either fully or partially pay out a user's staking rewards.

The fee of this transaction (with a hardcoded upper bound of 1.5 ADA, line 203 in `Staking/OnChain.hs`) is also meant to be covered by the recipient (i.e. no additional UTxOs are required, the treasury UTxOs suffice).

However, since there can't be a way to ensure maximum number of UTxOs are being consumed, an attacker can submit multiple transactions, each of which consumes a single 2 ADA UTxO from the treasury, with their fees set to 1 ADA (i.e. as high as possible, which is limited by the minimum Lovelaces required for the produced output), and deprive users of their full rewards. This also has negligible costs to the attacker.

As an example, consider a scenario where a user is eligible to 200 ADA in rewards. This means 100 UTxOs from the treasury are required to pay staking rewards of this user. If we assume each transaction can spend 10 UTxOs at max, this user should be getting a total of about 185 ADA.

However, if subject to an attack, each transaction will spend a single UTxO while their fees are set to max (currently about 1 ADA), and therefore pays out 1 ADA to the user per transaction. After 100 transactions, the user will be considered fully paid, but the total received rewards will be 100 ADA.

This loss can become more severe if the minimum required Lovelaces for a UTxO goes lower.

Recommendation

Limit the authority of this transaction to interested parties, i.e. Cerra.io and the eligible user.

With this approach, it is also advisable to lift the upper bound on the transaction fee in order to both avoid dependence on protocol parameters, and also allow the most number of UTxOs to be spent in a single transaction.

We would also recommend Cerra.io's authentication to be validated via an NFT.

ID-402 Unspendable Staked Tokens

	Level	Severity	Status
	4	Major	Pending

Description

Factory token policy only validates that the produced UTxO has a proper datum, and that it contains at least 1 CERRA, without further restrictions on the value.

On the other hand, `Unstake` endpoint of the staking script requires all the NFTs (and/or FTs with quantities of 1) included in the UTxO to be burnt.

This leads to a stake UTxO getting permanently locked if, for any reason, one or more NFTs are stored next to its staked CERRA during minting of the factory token.

Another situation that leads to the UTxO becoming unspendable, is when the staked CERRA quantity is exactly 1 (as it needs to be burnt along with the factory token in order to be unstaked).

Recommendation

Factory token policy should ensure that the produced UTxO has exactly 3 tokens: ADA, CERRA, and the minted token itself.

This allows the `Unstake` endpoint to validate by making sure a single token (that is not CERRA) is getting burnt in the transaction.

ID-301 User Reward Deprivation

	Level	Severity	Status
	3	Medium	Pending

Description

The `claim` endpoint relies on the input UTxOs to find how much ADA is being sent to the user. It also validates that this input ADA equals all the ADA sent to user, plus transaction fee.

A costly attack (e.g. carried out by a competitor) would be providing a single UTxO with minimum ADA, and spending it all on the transaction fee.

This leads to increased `sdAdaPaid` in the datum, without the user receiving any rewards.

Recommendation

Similar to the recommendation in ID-401, limiting the authority of this transaction to interested parties would be a simple solution.

If this is not an option, an extra validation that can make this attack costlier would be ensuring input ADA (minus ADA coming from staking UTxO) is greater than the transaction fee.

ID-302 Excessive Restriction on Value of Staking UTxOs

	Level	Severity	Status
	3	Medium	Pending

Description

The `claim` endpoint relies on the input UTxOs to find how much ADA is being sent to the user. It also validates that this input ADA equals all the ADA sent to user, plus transaction fee.

A costly attack (e.g. carried out by a competitor) would be providing a single UTxO with minimum ADA, and spending it all on the transaction fee.

This leads to increased `sdAdaPaid` in the datum, without the user receiving any rewards.

Recommendation

Similar to the recommendation in Vulnerability 01, limiting the authority of this transaction to interested parties would be a simple solution.

If this is not an option, an extra validation that can make this attack costlier would be ensuring input ADA (minus ADA coming from staking UTxO) is greater than the transaction fee.

ID-101 Inefficient Staking UTxO Lookup

	Level	Severity	Status
	1	Informational	Pending

Description

As already highlighted in the technical document, it is of importance to make the treasury script's logic as lightweight as possible.

The utility function currently used (`valueSpent`), traverses all inputs and the value fields of each of those inputs, and also performs concatenation between each value it passes.

This is somewhat expensive and can lead to higher execution budgets.

Recommendation

Consider using a filter function that looks for the factory token, and making sure it'll only find a single input.

This is a more optimized approach and can potentially increase the number of treasury UTxOs that can be spent in a single transaction.

ID-102 Misleading Variable Names

	Level	Severity	Status
	1	Informational	Pending

Description

Throughout the staking script, multiple field and variable names use “Ada” to address Lovelace values. This can potentially lead to some hinderance in code maintenance down the line.

Recommendation

Consider renaming “Ada” to “Lovelace” for a more explicit reading experience.

ID-103 Extra Traversal of Inputs

	Level	Severity	Status
	1	Informational	Pending

Description

Burning endpoint of the factory FT validates owner's signature by going through the transaction's inputs, making sure it finds a single UTxO, and then reads owner's address from the datum.

This additional traversal can increase transaction's fee.

Recommendation

Delegate this logic to the staking script itself, as it has a very cheap access to the spending datum. In other words, simply remove the call to `validateBurn` in `mkFTValidator`:

```
1 -- ...
2 in
3 if assetClassValueOf minted fTAssetClass == 1
4 then validateMint info fTAssetClass fpCerraAssetClass fpStakingValidatorHash
5 else assetClassValueOf minted fTAssetClass == -1
```

The reasons this is possible are:

- The minting logic only allows single factory tokens to be produced in a minting transaction, and it's enforced to be included in one UTxO at the staking script with a quantity of 1
- The requirement of a UTxO's epoch to be at least 2 behind that of the oracle's prevents any factory tokens produced outside the staking script, and therefore it is guaranteed that burning one factory token is accompanied by spending a single UTxO from the staking script
- The burning endpoint of the factory policy requires a single token burnt
- Staking script doesn't allow staking UTxOs' values to change (specifically, included factory token quantity can not change)
- The `Unstake` endpoint already validates a single UTxO being spent from the script via the `getStakingContractInput` utility function