ANASTASIA LABS

Security Audit Report

Business Confidential

Date: November 30, 2023
Project: Lenfi Pooled Loans
Version 1.1

Copyright © Anastasia Labs

Contents

Copyright © Anastasia Labs

Copyright © Anastasia Labs

Confidentiality statement

This document is the exclusive property of Anastasia Labs and Lenfi . This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both Anastasia Labs and Lenfi .

Lenfi may share this document with third parties under non-disclosure agreements to demonstrate security compliance.

Copyright © Anastasia Labs

Scope and Disclaimer

The scope of this audit is limited to the on-chain code specified in the files audited section. The code responsible for off-chain transaction building was not reviewed in this audit.

Aiken, the language that was used to write the Lenfi Pooled Loans smart contracts is relatively new; as such, the UPLC that it generates may not correctly correspond to the intentions of the Aiken code. The correctness of the Aiken compilation pipeline and the Aiken standard library are both out of the scope of this audit.

Anastasia Labs prioritized identifying the weakest security vectors as well as prominent areas where code quality can be improved.

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

The findings and recommendations contained herein reflect the information gathered by Anastasia Labs during the course of the assessment, and exclude any changes or modifications made outside of that period.

Copyright © Anastasia Labs

Assessment overview

From August 10th, 2023 to September 15th, 2023, Lenfi engaged Anastasia Labs to evaluate and conduct a security assessment of its Lenfi Pooled Loans 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.

Copyright © Anastasia Labs

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

Copyright © Anastasia Labs

Code base

Repository

https://github.com/mcomp-tech/lenfi-smart-contracts-private

Commit

ea2d51626ae1b771096d07b5001b26973e903694

Copyright © Anastasia Labs

Files audited

Copyright © Anastasia Labs

SHA256 Checksum	Files
7dbfd131f53239cf74276b232d03e1ccc089a0031e3d976a6423c79b01654716	validators/collateral.ak
ee03a0bfbfe54df2a51d6c88bc467aa18e626667f42792892bbc0fb578f74885	validators/leftovers.ak
72c8d8f7837a6315ba218227c907086e0f8a0467f6571e9cb474f681a0518b08	validators/liquidation_merge.ak
5e03b087cc6cf8fccb2d8b1d09fcdc6a1a5492c740f14006b5cc5acb4630657e	validators/liquidity_token.ak
9357c15924c30d4d67ca6c9f74bff34aceb86e43a9441fcf97b52c289662add9	validators/oracle_validator.ak
695210d4db5af91d6bcbb9380eee2ebec6dbf67740bc7bb7283fe229c12e5a56	validators/order_contract.ak
4ae815b570b0844a226afcb6c61a5487093017c43323a90a80cbdc23a063e553	validators/placeholder_nft.ak
bea88c0472da25c3f664b9e55ad2fe4e2c5846dbb40dd7345731cf73a221f50a	validators/pool.ak
15cd2b4408fe6019974f2ed4d174ff84840692aef61853384c6bfea7368af3a5	validators/pool_config.ak
49081c8950a2a06498a4630fd7e88f77dd1e0d148af24615d1e04c1b66eb63ee	validators/pool_stake.ak
b38186b02debdec83e4df2f8b9016be773ead9389dbc6ce3df050e89213e5e0f	lib/aada/finance.ak
422d686202c4e610c27fdf081e9ee9df92c796f72155eabde4ccfa20c2bfca53	lib/aada/nft.ak
d555df60f76595f0b03f3cf56f34ec417aeea50ec738c354191e4eaabe9dd20e	lib/aada/nft_pointers.ak
217acac978bc4bb1e638e4937840fe91d3a35ce8d908ae7bf1eebc373bbaec48	lib/aada/types.ak
2b98166ff42f3a5ea3433e6ca57b05ed93cc51d9126906c48b0b3e8cd1c029ec	lib/aada/utils.ak
86dcd1ad9c460ffd92822fe3b0e5e872600e940146edc68fb37d95a4ca82652f	lib/aada/types/collateral.ak
507da425426680ff5728b222a4ef6a8eb79a767445ad86e95774c43628ca3cab	lib/aada/types/leftovers.ak
c4a40670068195b5f235bb6c5cf358b87f2235f313ed79695e4f7b2e207f4e3a	lib/aada/types/oracle.ak
1c6e5eae6093a864196055f737f407026ff9c83ff756d8da517a0f1e690f00b2	lib/aada/types/order.ak
832cd7d9b74d5efe36a65b9fc0649f39cd60c1354c6fcbe25e13c435f3a3a951	lib/aada/types/pool.ak
3d395df6d1e273b6d78fe45c88a936273b119b7268dd8374207c545c3578347e	lib/aada/types/pool_stake.ak

Copyright © Anastasia Labs

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	5
4	Major	3
3	Medium	4
2	Minor	8
1	Informational	9
N/A	Notes	2

The audit used the Common Vulnerability Scoring System in conjunction with the NVD Calculator to provide a standardised measure for the severity of the identified vulnerabilities. We recognize that many of the metrics considered in CVSS are not relevant for audits of Cardano Smart Contracts; nonetheless, there is still considerable value in adhering to a standard in that it provides more unbiased severity metrics for the findings.

`https://www.first.org/cvss/v3.1/specification-document#Qualitative-Severity-Rating-Scale`

Copyright © Anastasia Labs

Executive summary

The Lenfi Pooled Lending Protocol is a peer to pool lending system on the Cardano blockchain. In the Lenfi Pooled Lending Protocol, each pool is represented as a single UTxO that resides at a multi-validator script. Likewise, each users collateral is stored in a single UTxO that resides at a multi-validator script.

Security Claims

- Liquidation is only possible if the borrower is below the health factor.
- In the case of liquidation, the borrower will retain everything except the value required to pay back the loan.
- All loaned value is collateralized.
- User's funds are maximally safe against the actions of malicious sequencer agents.

Copyright © Anastasia Labs

Findings

Copyright © Anastasia Labs

ID-501 Token Dust Attack x 4

Level	Severity	Status
5	Critical	Resolved

Description

The current script output value is unbounded, creating a potential vulnerability to a token spam attack. A malicious actor could exploit this by locking an unlimited number of tokens, effectively rendering the script un-spendable. To mitigate this risk, it is essential to implement output value validation based on exact value equality, rather than merely integer equality. Alternatively, the script could employ a mechanism to check the number of tokens contained in the output value.

Recommendation

We strongly recommend implementing robust validation for the output value structure and comparing the expected value with the actual output value to prevent potential token dust attacks.

Resolution

Resolved

Copyright © Anastasia Labs

ID-502 Indistinguishable Fake Pool Attack

Level	Severity	Status
5	Critical	Acknowledged

Description

The pool.ak minting policy has a vulnerability that allows an attacker to mint fake pool NFTs that are completely indistinguishable from real pool NFTs. The malicious actor can then use these fake pool NFTs to steal user funds by liquidating collateral or processing orders that were intended for the real pool.

The following code is the source of the exploit.

```
 let num_minted_check =
      list.length(redeemer) == dict.size(tokens(mint, own_address))
let mints_are_valid =
      list.foldl(
        redeemer,
        Some(inputs),
        fn(
          pool_mint: NFTRedeemerElement<Void>,
          curr_inputs_opt: Option<List<Input>>,
        ) -> Option<List<Input>>
```

The function incorrectly assumes that each element of the `NFTRedeemer<Void>` redeemer (which is a list) must be unique. To exploit this, a malicious actor can craft a transaction with duplicate entries in the `NFTRedeemer<Void>` redeemer such as:

```
[BurnNFT("JUNK_OREF"), BurnNFT("JUNK_OREF")]
```

Thus allowing them to burn one junk pool token, and mint another pool token with whatever token name they want. They could mint another token such that it has the exact same token name of an existing real pool. There are no checks on the output destination of the UTxO containing that minted token, so they can send it to their own wallet or where-ever they desire with any arbitrary datum.

Recommendation

Either enforce that the redeemer is a single mint / burn action instead of a list of mint / burn actions or, before you use the redeemer apply a filter to it that removes duplicate elements.

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-503 Malicious Liquidation to steal funds

Level	Severity	Status
5	Critical	Resolved

Description

An attacker can execute malicious liquidations by executing liquidation transactions with the upper-bound of the transaction validity range set far into the future. `CollateralLiquidate`

Recommendation

Enforce a maximum transaction validity range for the collateral validator that is as low as possible. Additionally, the `calculate_interest_amount` calculation should use the `lower_bound` of the transaction validity range instead of the `upper_bound`.

```
expect Finite ( begin ) = transaction.validity_range.lower_bound.bound_type
expect Finite ( end ) = transaction.validity_range.upper_bound.bound_type
...
and {
end - begin <= 350000,
}
```

Resolution

Resolved

Copyright © Anastasia Labs

ID-504 Infinite Counterfeit LP Tokens

Level	Severity	Status
5	Critical	Acknowledged

Description

The liquidity_token.ak minting policy has a vulnerability that allows a malicious user to mint an infinite number of real LP tokens. This attack is possible because the liquidity_token.ak minting policy does not enforce adequate checks on the total tokens with the LP token currency symbol that are minted or burned in the `DestroyPool` and `CreatePool` redeemer code paths.

For instance the only reference to the number of minted LP tokens in the `CreatePool` redeemer code path is:

```
CreatePool { produced_output } -> {
    ...
    let mint = from_minted_value(ctx.transaction.mint)
    let lp_tokens_minted = quantity_of(mint, own_policy, pool_token_name)
    ...
```

The validator uses `lp_tokens_minted` to perform checks on the number of minted LP tokens with `pool_token_name`; however, no checks are performed on the number of minted LP tokens with other token names. This means a malicious user can mint the correct number of tokens with the token name `pool_token_name`, and in the same transaction, mint an infinite number of real LP tokens with token names that correspond to other existing pools. Similarly, the `DestroyPool` checks that the correct number of LP tokens with the given token name is burned; however, it does not prevent LP tokens with different token names from being minted.

Recommendation

Enforce that only the expected tokens are minted by the liquidity_token.ak minting policy using a method similar to the one used in the `TransitionPool` redeemer code path.

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-505 Negative Interest Rates to Drain Pool

Level	Severity	Status
5	Critical	Resolved

Description

The protocol does not have sufficient checks to prevent malicious actors from creating negative interest rate loans.

The attack can be conducted as follows:

1. Create an invalid repay_order_request locked at the repay_order_request that has no associated collateral UTxO.
2. Process the invalid repay_order_request to the pool with the `pool.CloseLoan` redeemer with a huge `loan_amount` thus paying that huge amount into the pool UTxO and causing the `lent_out` to be negative.
3. Process a borrow_order_request to the pool with the `pool.Borrow` redeemer with a extremely negative interest rate, the provided negative interest rate will match the calculated interest rate because the calculation in `get_interest_rates` can have multiplication with negative values because `lent_out` is now negative.
4. Process a liquidate_order_request to the pool and with a negative interest rate you actually get paid for borrowing money, you get paid more based on how long you borrowed it, borrow it for long duration and eventually liquidate and drain the entire pool.

The attack is made possible because there is no validation logic to prevent negative values for `lent_out` or for the interest rate parameters.

Recommendation

Enforce that `lent_out` must not be negative, and that negative interest rate loans are not allowed.

Resolution

Resolved

Copyright © Anastasia Labs

ID-506 Indistinguishable Fake Collateral Attack

Level	Severity	Status
5	Critical	Acknowledged

Description

The collateral.ak minting policy has a vulnerability that allows an attacker to mint fake collateral NFTs that are completely indistinguishable from real collateral NFTs. The malicious actor can then use these fake collateral NFTs to create under-collateralized borrow requests.

The following code is the source of the exploit.

```
  let num_minted_check =
   list.length(redeemer) == dict.size(tokens(mint, own_address))
let mints_are_valid =
  list.foldl(
    redeemer,
    Some(inputs),
    fn(
      collateral_mint: NFTRedeemerElement<CollateralNFTInner>,
      curr_inputs_opt: Option<List<Input>>,
    ) -> Option<List<Input>>
```

The function incorrectly assumes that each element of the `NFTRedeemer<CollateralNFTInner>` redeemer (which is a list) must be unique. To exploit this, a malicious actor can craft a transaction with duplicate entries in the `NFTRedeemer<CollateralNFTInner>` redeemer such as:

```
[BurnNFT("JUNK_OREF"), BurnNFT("JUNK_OREF")]
```

Recommendation

Either enforce that the redeemer is a single mint / burn action instead of a list of mint / burn actions or, before you use the redeemer apply a filter to it that removes duplicate elements.

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-507 Malicious Pool Manager can set Negative Pool Fee in to Drain the Pool

Level	Severity	Status
5	Critical	Resolved

Description

Whoever manages the pool_config can steal all the funds in the pool by updating the pool_fee to be negative, and then using the `PayFee` redeemer and spend the pool UTxO and take `pool\_fee` amount of the loan asset from the pool. Similarly they could make the fees in `loan\_fee\_details` negative and create collateralized loan requests with a large amount of collateral and a negative interest rate.

Recommendation

It makes sense to allow each pool to decide how it wants to handle the pool_config, however, there should be certain in variants. In general, pool_config should not ever allow `pool\_fee` to be negative, `liquidationThreshold` should have some bounded range, the fees in `loan_fee_details` should not ever be negative and should have some reasonable upper bound. And so on.

Furthermore, `PayFee` should check that the value of the continuing pool output is strictly greater than the value of the pool input.

Resolution

Resolved

Copyright © Anastasia Labs

ID-401 DEX UTXOs contention for Oracle Validators

Level	Severity	Status
4	Major	Acknowledged

Description

In the context of oracle validators, a significant risk emerges in high-traffic environments where oracles may not effectively read from the DEX pool by referencing it as an input. This risk stems from the frequent spending of DEX UTXOs, which may lead to a scenario where liquidations cannot be executed when necessary, potentially resulting in the loss of user funds.

Recommendation

Implement strategies that allow for seamless and uninterrupted oracle access to the DEX pool, even in high-traffic environments

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-402 Staking Rewards Theft

Level	Severity	Status
4	Major	Resolved

Description

The pool_stake.ak staking validator has a vulnerability that allows a malicious user to steal staking rewards from the protocol. This is because the validator mistakenly enforces that the amount of staking rewards withdrawn is greater than the amount sent to the pool. This allows an attacker to withdraw a large amount of staking rewards and only send a small portion of it to the pool.

The mistaken check is:

```
let amount_check = current_withdrawal_amount >= fee_amount
```

Recommendation

Modify the check to ensure that the party processing the transaction is only able to take a nominal fee from the staking rewards, and enforce that the rest go to the pool.

```
let current_withdrawal_fee_adjusted = current_withdrawal_amount - 2_000_000
...
let amount_check = fee_amount >= current_withdrawal_fee_adjusted
```

Resolution

Resolved

Copyright © Anastasia Labs

ID-403 Staking Reward Smuggle Fake Pool

Level	Severity	Status
4	Major	Resolved

Description

The pool_stake.ak staking validator has a vulnerability that allows a malicious user to steal staking rewards from the protocol. This is because the validator mistakenly assumes that `pool_oref` and `nft_oref` from the redeemer are equivalent. Since they are not equivalent, a malicious user could pass a real pool output reference for `nft_oref` and a fake pool output reference for `pool_oref` and then withdraw the rewards into their fake pool (ie themselves).

The mistaken check is:

```
  expect Some(pool_input) =
    inputs |> list.find(fn(out) { out.output_reference == nft_oref})
  let nft_check =
    quantity_of(pool_input.output.value, pool_nft_policy, pool_nft_name) == 1
expect Some(raw_pool_redeemer): Option<Data> =
    redeemers |> dict.get(Spend(pool_oref))
```

Recommendation

Modify the validator to enforce that all checks are done with `pool_oref` and remove `nft_oref` from the validator.

```
  expect Some(pool_input) =
    inputs |> list.find(fn(out) { out.output_reference == pool_oref})
  let nft_check =
    quantity_of(pool_input.output.value, pool_nft_policy, pool_nft_name) == 1
expect Some(raw_pool_redeemer): Option<Data> =
    redeemers |> dict.get(Spend(pool_oref))
```

Resolution

Resolved

Copyright © Anastasia Labs

ID-301 Infinite mint x 4

Level	Severity	Status
3	Medium	Acknowledged

Description

A potential vulnerability exists wherein an attacker may deliberately mint an infinite number of tokens. This can be achieved by adding an external minting policy in the transaction field and subsequently locking these assets into the output script.

The current output value validation does not check against unbounded values. For example, the 'quantity_of' function only verifies whether the minted amount is positive or negative for a specific asset.

Recommendation

We strongly recommend two approaches to mitigate this risk.

First, ensure that the 'mint' field contains only one token policy. Additionally, if uniqueness of minted tokens is required, verify that the token names are unique.

Alternatively, if the protocol intends to allow other minting policies within the same transaction, the script should enforce constraints to ensure that the script output value remains bounded.

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-302 Missing explicit Datum

Level	Severity	Status
3	Medium	Acknowledged

Description

The absence of explicit the output datum values introduces a potential risk wherein the output datum can be arbitrarily set, potentially locking funds indefinitely. This occurs due to unexpected datum parsing when spending the UTXO script.

Recommendation

We strongly recommend conducting a rigorous validation process to ensure that the output datum types are correctly and explicitly applied.

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-303 Liquidation Double Satisfaction

Level	Severity	Status
3	Medium	Resolved

Description

The risk of double satisfaction arises when two or more spend validator checks and validates the same conditions, allowing an attacker to spend multiple inputs from the script by providing just one valid output. This vulnerability can be mitigated by limiting the transaction context to contain only one script input and one script output. Alternatively, the contract could establish a unique datum tag to ensure the uniqueness of outputs. However, this approach is currently compromised by the issue identified in ID-202, which prevents uniqueness in datum.borrower_tn. As a result, there remains the potential for double satisfaction, as no unique output tag is enforced. It is essential to note that datum.borrower_tn should ideally be unique, but its uniqueness is currently not validated, a concern that is linked to the findings in ID-202.

Recommendation

We recommend taking one of two actions to address this issue. First, resolve the matter identified in ID-202 to enforce datum tag uniqueness, particularly with regard to datum.borrower_tn. Alternatively, ensure that each transaction context contains only one script input and one script output.

Resolution

Resolved

Copyright © Anastasia Labs

ID-304 Pool Agent Censorship Prevent Loan Repayment

Level	Severity	Status
3	Medium	Resolved

Description

The protocol relies on a sequencing model instead of the batching architecture. This is possible because many protocol actions do not cause UTxO contention and thus can be done by any number of users up to the blockchain limitations. However, currently loan repayment does involve UTxO contention with the pool. This introduces a lucrative opportunity for malicious agent to deliberately ignore certain user's `repay\_order\_request` in favor of other orders that were placed after. As long as there are other valid orders to be processed, the agent stands to make a lot of money by just processing those orders (or in the absence of those orders they can create their own orders) while ignoring the user's `repay\_order\_request` until the user is ultimately liquidated because they were blocked from paying back their loan in time. The system needs some mechanism to guarantee that once a valid `repay\_order\_request` (corresponding to some valid collateral) that collateral cannot be liquidated by any agent. Since the collateral script is already parallelized (1 UTxO per user borrow) it makes the most sense to just have the collateral UTxO directly be the repayment request (you can even get rid of the `repay\_order\_request` script) There are a few architecture comments that could benefit the protocol greatly. What is the batcher throughput right now? It must be limited by the heavy distribution of logic across scripts and the heavy use of redeemers for message passing and for identifying UTxOs.

Recommendation

Support loan repayment directly in that collateral UTxO so that it does not involve UTxO contention with the lending pool.

Resolution

Resolved

Copyright © Anastasia Labs

ID-201 Missing validation for borrowed_collateral_amount

Level	Severity	Status
2	Minor	Resolved

Description

The 'borrowed_collateral_amount' is a parameter that originates from the pool redeemer during the borrow action and is subsequently set within the collateral output datum. However, there is currently no validation implemented in either the pool or collateral script, allowing a malicious actor to set this value arbitrarily.

Recommendation

We strongly recommend implementing validation to ensure that the value set in the redeemer aligns with the value established in the collateral output datum.

Additionally, consider removing the 'borrowed_collateral_amount' parameter from the redeemer, as the validation of borrowed collateral amounts should occur exclusively during script execution against the output datum. Removing this parameter from the redeemer can reduce script overhead.

Resolution

Resolved

Copyright © Anastasia Labs

ID-202 Borrower Token Name inconsistency

Level	Severity	Status
2	Minor	Resolved

Description

borrower_tn is initially set in the redeemer during borrow action , and subsequently assigned to the collateral output datum , However validation for borrower token name uniqueness is only done at during mints_are_valid inside MintNFT as follows:

```
let expected_token_name = id_from_utxo(pool_utxo).
```

This configuration allows the borrower to mint tokens with a unique token name, but it also can inadvertently set the wrong borrower token name (borrower_tn) at collateral datum.

Additionally, this could also open the door for a potential double satisfaction which is related to ID-304

Recommendation

We strongly recommend implementing a validation to ensure that borrower_tn exclusively matches expected_token_name, moreover we discourage the use of borrower_tn at the redeemer level, since it provides no benefits beyond adding unnecessary script overhead.

Resolution

Resolved

Copyright © Anastasia Labs

ID-203 Missing output address x 2

Level	Severity	Status
2	Minor	Resolved

Description

The continuing_output is an index which is employed to reference a particular output, and it is primarily used to enhance script efficiency. However the absence of validation for explicit pool output address, could result in locking the pool assets at an unintended address.

Recommendation

We strongly recommend incorporating the script output address, this will mitigate the risk of locking assets into unintended addresses.

Resolution

Resolved

Copyright © Anastasia Labs

ID-204 Unhandled error with default fallback in loan_value

Level	Severity	Status
2	Minor	Resolved

Description

The function 'do_oracle_calculation' uses the 'loan_amount' as an input to safely compute the 'ada_for_purchase', resulting in two possible outcomes : 'Some(A)' or 'None'. However, the outcome of this computation is not guaranteed to be deterministic. In cases where the return value is 'None,' there should be a defined mechanism in place to handle this error.

Recommendation

Our recommendation is to handle the None cases properly, and avoid situations where the input value is returned with 'option.or_else' when an intermediate computation fails

Resolution

Resolved

Copyright © Anastasia Labs

ID-205 Unhandled error with default fallback in collateral_tokens_value

Level	Severity	Status
2	Minor	Resolved

Description

The function 'do_oracle_calculation' uses the 'collateral_amount' as an input to safely compute the 'calculate_sale', resulting in two possible outcomes : 'Some(A)' or 'None'. However, the outcome of this computation is not guaranteed to be deterministic. In cases where the return value is 'None,' there should be a defined mechanism in place to handle this error. Currently, the system defaults to using 'collateral_amount' as the result, which may not be appropriate.

Recommendation

Our recommendation is to handle the None cases properly, and avoid situations where the input value is returned with 'option.or_else' when an intermediate computation fails

Resolution

Resolved

Copyright © Anastasia Labs

ID-206 Unhandled error with default fallback in tag_check

Level	Severity	Status
2	Minor	Resolved

Description

The function 'list.any' uses the 'inputs' and 'oref' from tag as inputs to safely compute the 'list.any', resulting in two possible outcomes : 'Some(A)' or 'None.' However, the outcome of this computation is not guaranteed to be deterministic. In cases where the return value is 'None,' there should be a defined mechanism in place to handle this error. Currently, the system defaults to using 'True' as the result, which may not be appropriate.

Recommendation

Our recommendation is to handle the None cases properly, and avoid situations where a True value is returned with 'option.or_else' when an intermediate computation fails

Resolution

Resolved

Copyright © Anastasia Labs

ID-207 Potential discrepancy of collateral Loan value

Level	Severity	Status
2	Minor	Resolved

Description

The collateral script is responsible for computing the loan value, and the 'order_contract' stores the 'expected_output' as a datum when a borrower initiates an order.

The 'expected_output' datum is meant to include the anticipated loan value, while the collateral script is also expected to calculate a loan value.

it is crucial to establish a validation mechanism that verifies whether the loan value calculated by the collateral script aligns with the 'expected_output.'

Recommendation

To mitigate potential discrepancies, it is strongly advised to implement a validation step that verifies the consistency of the collateral's loan value with the expected output.

Resolution

Resolved

Copyright © Anastasia Labs

ID-208 merge_script_hash Placeholder

Level	Severity	Status
2	Minor	Resolved

Description

merge_script_hash is empty

Recommendation

Add merge_script_hash

Resolution

Resolved

Copyright © Anastasia Labs

ID-209 Sequencer can set borrow order deposit time far into the future

Level	Severity	Status
1	Informational	Resolved

Description

A sequencer who executes a borrow order request can set the `deposit_time` of the collateral datum to any time in the future. This is because the collateral minting policy enforces that the `deposit_time` must be set to the upper bound of the transaction validity range thus enforcing that `deposit_time` cannot be set to the past; however since no constraints are set on the duration of the validity range, the agent executing the borrow request could build a transaction such that the upper bound of the validity range is far into the future, thus enabling them to set the `deposit_time` to that time.

Recommendation

Enforce a maximum length for the transaction validity range in the `borrow_order_contract` validator.

```
expect Finite(begin) = transaction.validity_range.lower_bound.bound_type
expect Finite(end) = transaction.validity_range.upper_bound.bound_type
...
and {
    valid_interest_rate,
    valid_pool_collateral_amount,
    borrower_token_minted,
    recipient_got_borrow_nft,
    user_received_value,
    collateral_was_tagged,
    end - begin <= 3600000,
  }
```

Resolution

Resolved

Copyright © Anastasia Labs

ID-101 Shadowed variable x 3

Level	Severity	Status
1	Informational	Acknowledged

Description

The variable pool_nft_name is declared and used by actual_datum_output, but it is also shadowed by raw_pool_datum.

The variable mint is declared and used by ctx, but it is also shadowed by from_minted_value function.

Recommendation

To maintain code readability and prevent unexpected outcomes, it is recommended to avoid shadowing variables. Consider using unique variables names or restructuring the code to eliminate shadowing.

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-102 Redeemer interest discrepancy

Level	Severity	Status
1	Informational	Acknowledged

Description

There is a potential discrepancy between variables interest_amount and calculate_interest_amount, since both are used interchangeably in different functions. This inconsistency can lead to errors when validating the output value correctness.

Recommendation

To improve code readability and reduce redundancy, it is advisable to exclusively use ‘calculate_interest_amount‘ and remove ‘interest_amount‘ from the redeemer. This simplification will ensure consistent interest calculations throughout the codebase

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-103 Rational interest rounding

Level	Severity	Status
1	Informational	Resolved

Description

It is important to note that interest originates from a rational number which is then truncated. The truncate function rounds the number down and it could potentially cause unexpected interest outcomes, such as situations when the interest is rounded down to zero.

Recommendation

It is recommended to use a ceil function to round the number up for a more safety interest calculation

Resolution

Resolved

Copyright © Anastasia Labs

ID-104 Pool NFT Redundant validation

Level	Severity	Status
1	Informational	Resolved

Description

The nfts_check is redundantly computed in two separate locations.

First, nfts_check validation is computed in liquidity_token policy, and secondly it is repeated within the pool spending validator during the execution of validate_transition, where it is referred to as pool_output_nft_check

Recommendation

In order to optimize transaction efficiency, we recommend consolidating pool nft check validation within pool spending validator.

Furthermore, it appears that there is a parameter dependency issue where minting policy liquidity_token requires pool_hash, and additionally datum parameter (pool.Datum) at pool spending validator requires the liquidity_token policy (params.lp_token.policy_id).

Resolution

Resolved

Copyright © Anastasia Labs

ID-105 Lack of Input Pool NFT Validation

Level	Severity	Status
1	Informational	Resolved

Description

The current codebase allows spending without requiring the presence of pool NFT at the input level. This means that the script could pass validation by incorporating an external input containing the pool NFT with pool_nft_policy and pool_nft_name.

It is worth noting that pool input and output value validation is also done at liquidity_token policy.

Recommendation

In order to enhance the protocol security we strongly suggest implementing a validation in the input script to validate whether the pool nft is locked, before spending it, and once spent this can be locked back into the script.

Resolution

Resolved

Copyright © Anastasia Labs

ID-106 Duplicated pool output value validation

Level	Severity	Status
1	Informational	Acknowledged

Description

The correct_pooltoken_out is redundantly computed in two separate locations.

First, correct_pooltoken_out is computed in liquidity_token policy,and secondly it is repeated within the pool minting policy during the execution of mints_are_valid, where it is referred to as correct_quantity

Recommendation

In order to optimize transaction efficiency, we recommend consolidating pool output value validation within the pool policy. Furthermore, it appears that there is a parameter dependency issue where minting policy liquidity_token requires pool_hash, and additionally pool policy datum parameter (pool.Datum) requires the liquidity_token policy (params.lp_token.policy_id).

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-107 Minimum ADA for Pools with ADA Loans

Level	Severity	Status
1	Informational	Acknowledged

Description

As per the ledger protocol, it is required to lock a minimum ADA in every UTXO. This requirement is determined by the following formula.
`minUTxoVal = (160 + sizeInBytes (TxOut)) * coinsPerUTxOByte`.
It is important to note that coinsPerUTxOByte is a ledger parameter that could change in the future, therefore Pools where loans are defined in ADA should take into account this minimum ADA requirement.

Recommendation

In the context of ADA pool creation, it is advisable to implement a conditional validation to ensure compliance with the minimum ADA requirement.

The function which determines LP token, should exclude the minimum ADA from its calculation.

In order to simplify the LP token calculations, the developer should consider establishing a constant minimum ADA value, such as 4 ADA during pool creation. Subsequently, upon closure of the pool, this minimum ADA can be returned to the original pool creator.

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-108 Duplicated variable ‘borrower_nft_policy‘

Level	Severity	Status
1	Informational	Acknowledged

Description

The variable ’borrower_nft_policy’ is found to be duplicated within the code base.

Recommendation

It is advised to eliminate the duplication of the ’borrower_nft_policy’ variable to maintain code clarity and efficiency.

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-109 Two outputs with identical destination

Level	Severity	Status
1	Informational	Acknowledged

Description

There exist two outputs directed to the same borrower address ‘expected_output‘ and ‘expected_recipient_output‘.

It is advisable to consolidate these outputs to streamline the transaction, leading to cost savings in script execution and reduced transaction fees. Moreover, this consolidation contributes to heightened security and code readability.

It is noteworthy that ‘expected_output‘ lacks validation checks specific to the borrower address, which adds an element of risk to the transaction.

Recommendation

It is strongly recommended to consolidate the two borrower outputs into a single output directed to the borrower’s address, and ensuring that the output address matches the borrower’s address.

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-N/A Lack of unit test for helper functions

Level	Severity	Status
N/A	N/A	Acknowledged

Description

There is a lack of unit testing for helper functions. the absence of unit tests raises concerns regarding the reliability and predictability of these functions, without them it is challenging to asses how the functions will perform under different conditions.

Recommendation

It is strongly advised to incorporate unit tests for the helper functions.

Resolution

Acknowledged

Copyright © Anastasia Labs

ID-N/A Documentation enhancement for protocol and functions

Level	Severity	Status
N/A	N/A	Acknowledged

Description

There is a clear need for substantial documentation enhancements, particularly in the context of contract interactions and functions. While some documentation exists, it suffers from incompleteness and a lack of clarity, making it challenging to understand the correctness of the protocol.

Several notable examples of undocumented functions include:

- ada_for_purchase
- calculate_sale
- asset_gain_adasale
- calculate_health_factor
- get_interest_rates
- calculate_tokenswap_discount
- check_is_overcollaterized
- check_is_undercollaterized

In addition to improving function documentation, it is imperative to enhance protocol documentation for the following contracts:

- collateral.ak
- leftovers.ak
- liquidation_merge.ak
- liquidity_token.ak
- oracle_validator.ak
- order_contract.ak
- placeholder_nft.ak
- pool.ak
- pool_config.ak
- pool_stake.ak

Copyright © Anastasia Labs

Recommendation

Enhance documentation protocol and functions It is highly recommended to address the documentation gaps. Enhancing documentation for both functions and contracts will significantly improve the the protocol's correctness and reliability.

Resolution

Acknowledged

Copyright © Anastasia Labs