January 22, 2024

Lenfi - V2

Contents

1 - Summary

We undertook a thorough examination of Lenfi's V2 validators. Lenfi is a decentralized lending protocol built on Cardano.

The audit is conducted without warranties or guarantees of the quality or security of the code. The investigation spanned several potential vulnerabilities, including scenarios where attackers might exploit the validator to lock up or steal funds. It's important to note that this report only covers identified issues, and we do not claim to have detected all potential vulnerabilities.

1.a - Overview

Lenfi is a decentralized lending protocol built on Cardano. Originally Lenfi supported a P2P model which is considered V1. V2 of Lenfi seeks to add support for pooled lending.

- At launch the pool validator is parameterized by a delegator nft policy and a pool script hash
- A pool is a UTXO at a pool validator address
- Each pool UTXO will be at a unique address with the pool validator hash as the payment credential and a unique stake key.

Users will be able to interact with the protocol through these kinds of actions:

- **Create Pool**: users can create pools and seed them with initial liquidity.
- **Destroy Pool**: users can destroy a pool only if all the assets are out of the pool and there are no liquidity tokens.
- **Deposit**: users can deposit the Pool Asset into a pool and receive liquidity tokens.
- **Withdraw**: users can withdraw the Pool Asset from a pool by burning their liquidity tokens.
- **Borrow**: users can borrow the Pool Asset from a pool by depositing collateral and they will receive a collateral NFT.
- **Repay**: users can repay the Pool Asset to a pool and burn their collateral NFT to get their collateral back.
- **Liquidate**: users can liquidate a loan by repaying the Pool Asset thus receiving the collateral.

1.a.a - Highlevel Protocol Flow

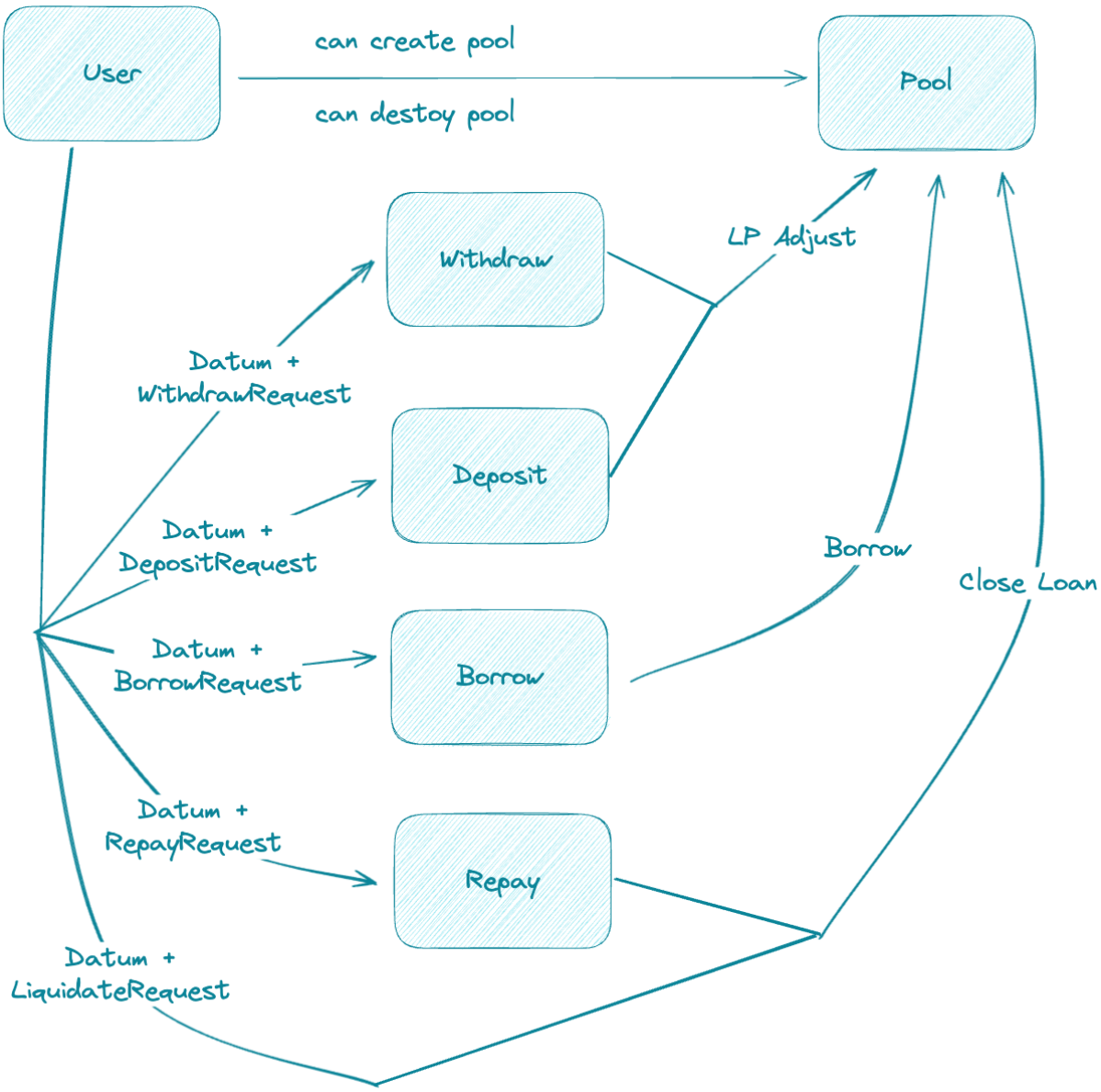


Figure 1: Withdraw, Deposit, Borrow, and Repay are all their own validators that accumulate “orders”.

1.b - Process

The audit process involved a meticulous review of the V2 validators for the Lenfi lending platform. Our team focused on scrutinizing areas susceptible to potential security threats, where attackers could exploit the contract’s functions to lock up or steal funds from the dApp and its users. Our methodical approach encompassed a range of potential attack vectors, including unauthorized minting, funds theft, denial of service, and business requirements violation, among others. The audit, carried out from Nov 13, 2023 to Jan 22, 2024, involved regular interactions on Discord where feedback was submitted. The Lenfi team addressed all issues, as detailed in this report.

1.c - Transactions

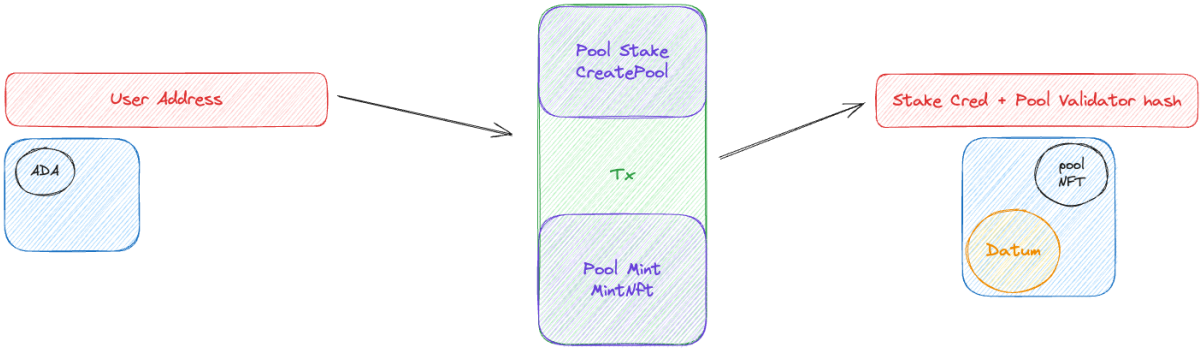
Below are a list of transactions as diagrams that power Lenfi V2.

Oracle Info

- The Oracle NFT lives at the oracle validator address, but the datum is not important so it's set to void
- The withdrawal validator's redeemer contains the actual oracle data
- The withdrawal validator checks that the data is signed by a 5 of 9 multisig

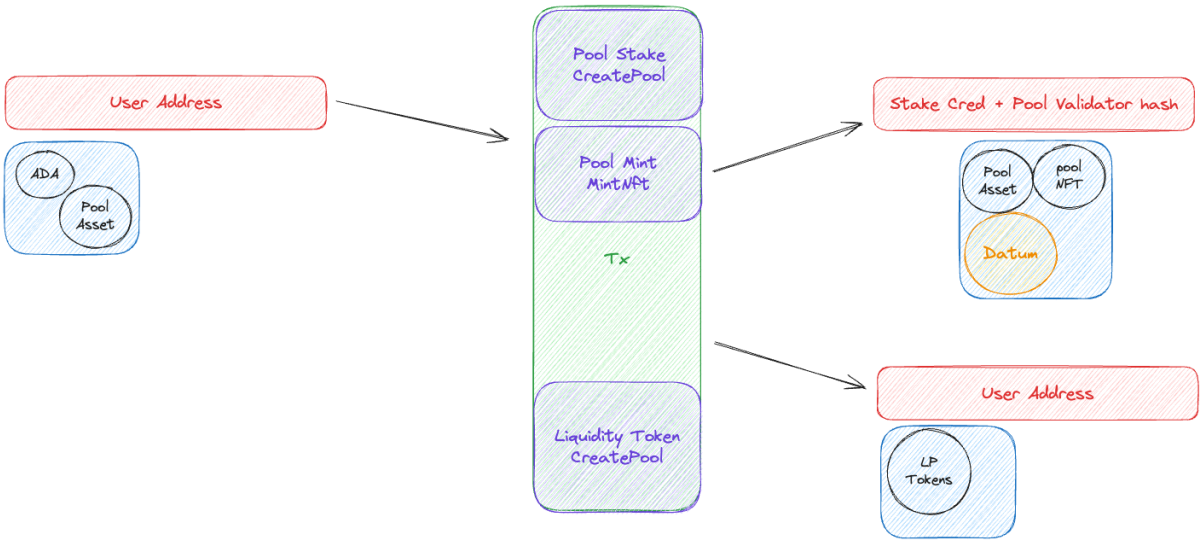
1.c.a - Create Pool

A user can create a pool without seeding it with liquidity. In this case there are two validators involved `pool_stake` and `pool`. `pool_stake` is used with the `delegate` purpose and `pool` is a multivalidator that can be used for both `spend` and `mint`. `pool` mint enforces the creation of only **ONE** pool per transaction. The `pool NFT` asset name is the stake credential of the address the output resides at.



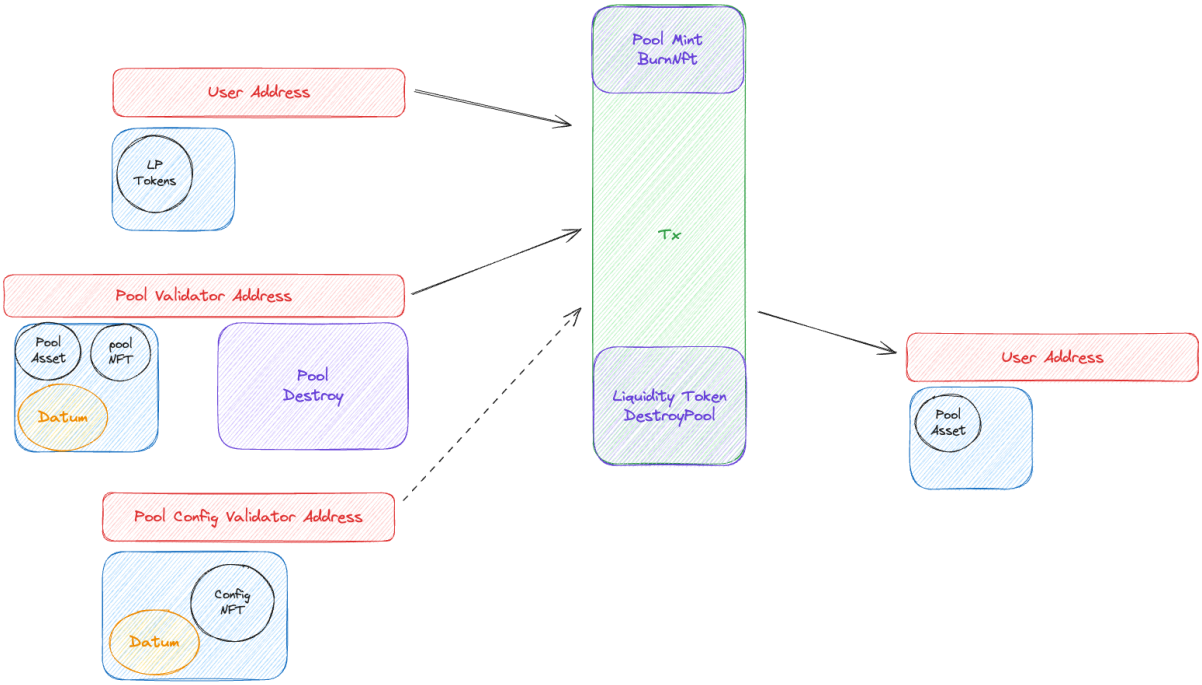
1.c.b - Create Pool - With Seed

Similarly, a pool can be created and seeded with initial liquidity. The only difference is that now the user must include the `pool asset`, there is a `liquidity_token` validator for minting, the user gets LP tokens, and the pool output will contain the pool asset.



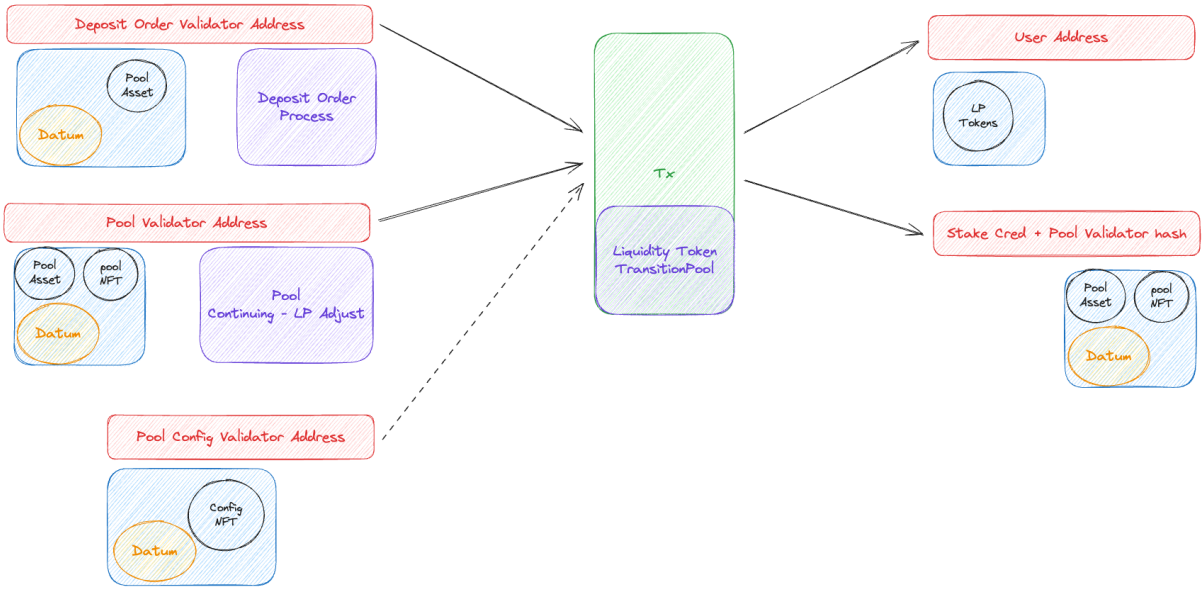
1.c.c - Destroy Pool

A user can destroy a pool if the pool is empty and there are no liquidity tokens. The `pool` validator is used with the `spend` purpose. The `liquidity_token` mint validator is used with a `DestroyPool` redeemer which burns the remaining LP tokens provided by the user.



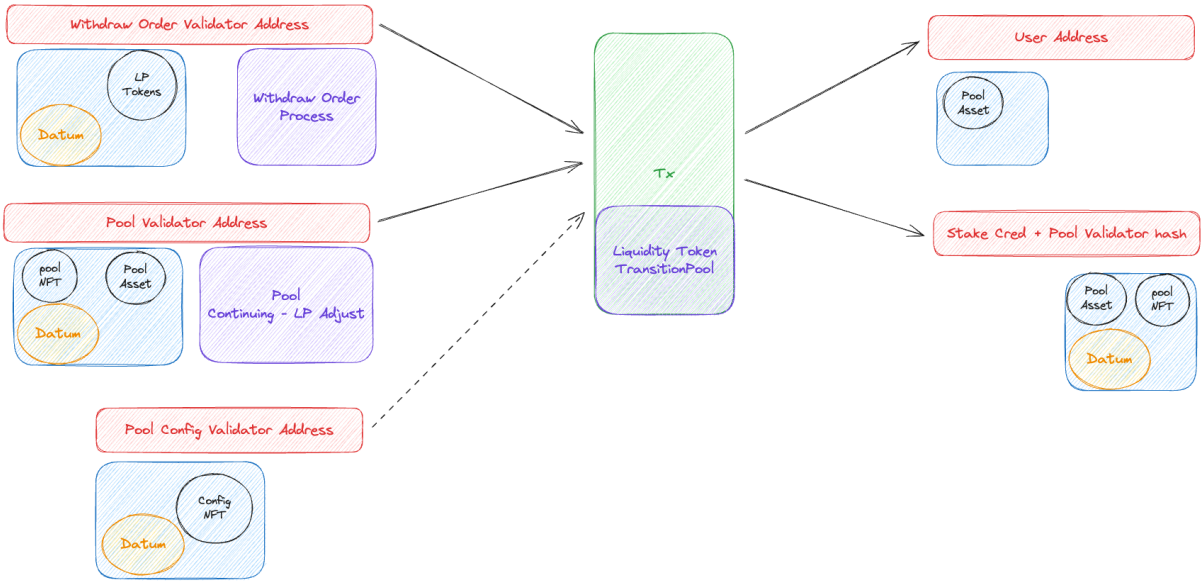
1.c.d - Pool Deposit

A user can deposit the pool asset into a pool and receive liquidity tokens. The `pool` validator is used with the `spend` purpose and the redeemer is set to `Continuing - LP Adjust`. Values from the `pool_config` are used and the pool config is used as a reference input. The `liquidity_token` mint validator is used with a redeemer set to `TransitionPool`.



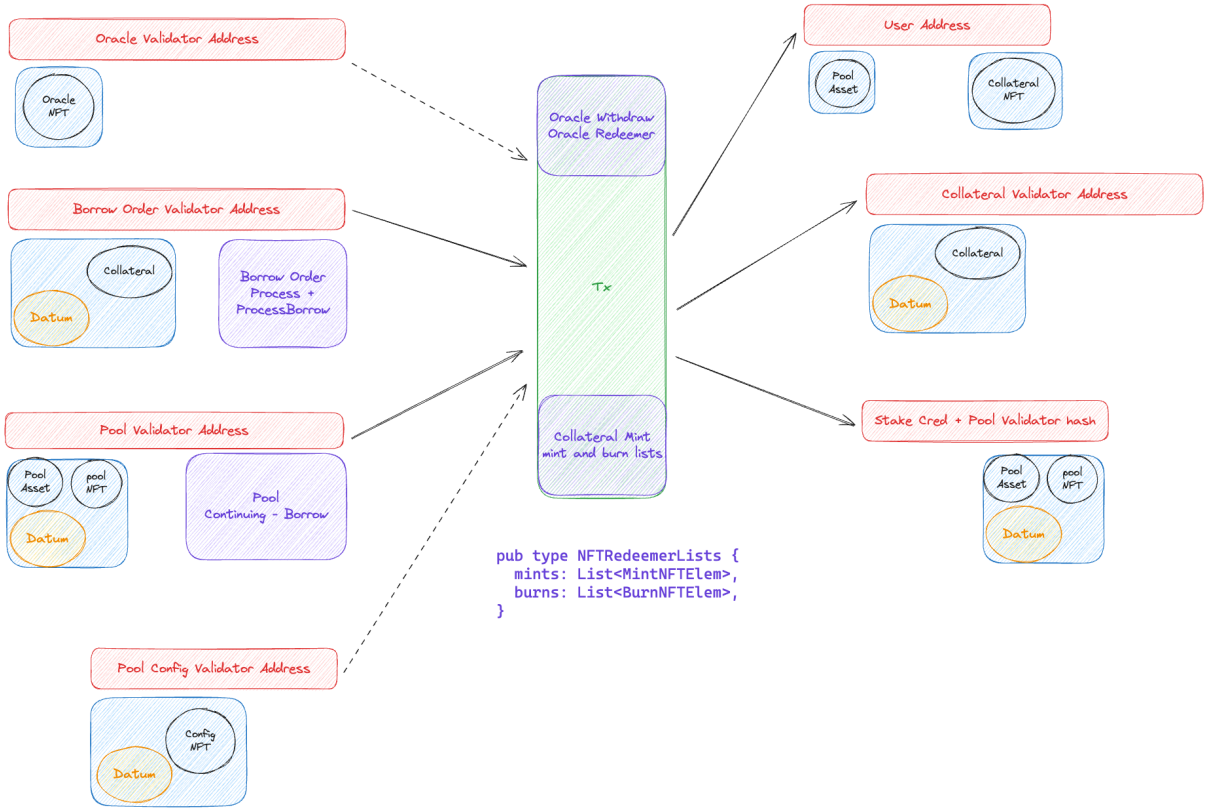
1.c.e - Pool Withdraw

A user can withdraw the pool asset from a pool by burning their liquidity tokens. The `pool` validator is used with the `spend` purpose and the redeemer is set to `Continuing - LP Adjust`. Values from the `pool_config` are used and the pool config is used as a reference input. The `liquidity_token` mint validator is used with a redeemer set to `TransitionPool`.



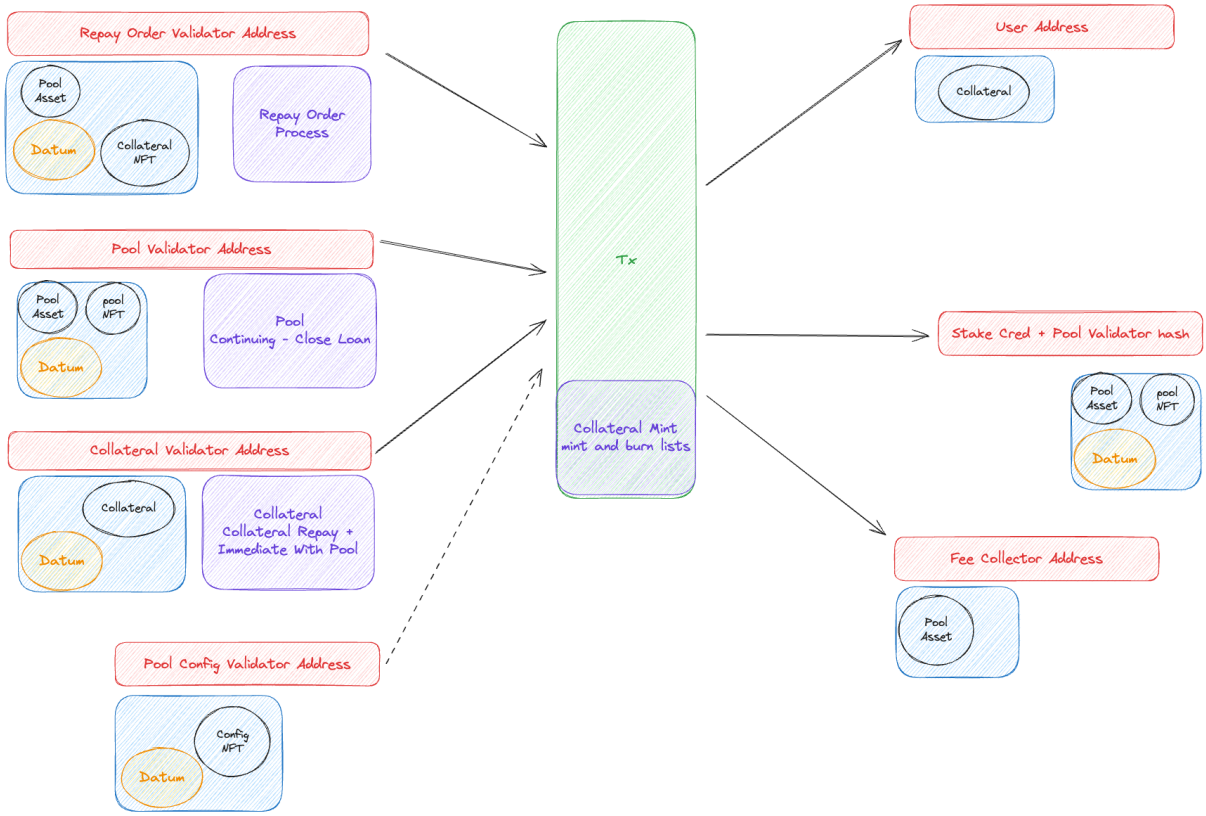
1.c.f - Pool Borrow

A user can borrow pool assets by locking up collateral. The collateral goes to a separate collateral validator with a datum. The user gets the pool asset and a collateral NFT which can be used later during repay or while claiming leftovers after liquidation. The `pool` validator is used with the `spend` purpose and the redeemer is set to `Continuing - Borrow`. Values from the `pool_config` are used and the pool config is used as a reference input. The oracle is used as withdraw script during the process of minting the collateral NFT to make sure that the value of the collateral is greater than the borrowed value. The end (user) address in this case could contain a datum due to how Lenfi chose to define the borrow datum. The borrow datum contains a user specific partial output which could contain a datum. This is a design chosen to promote composability.



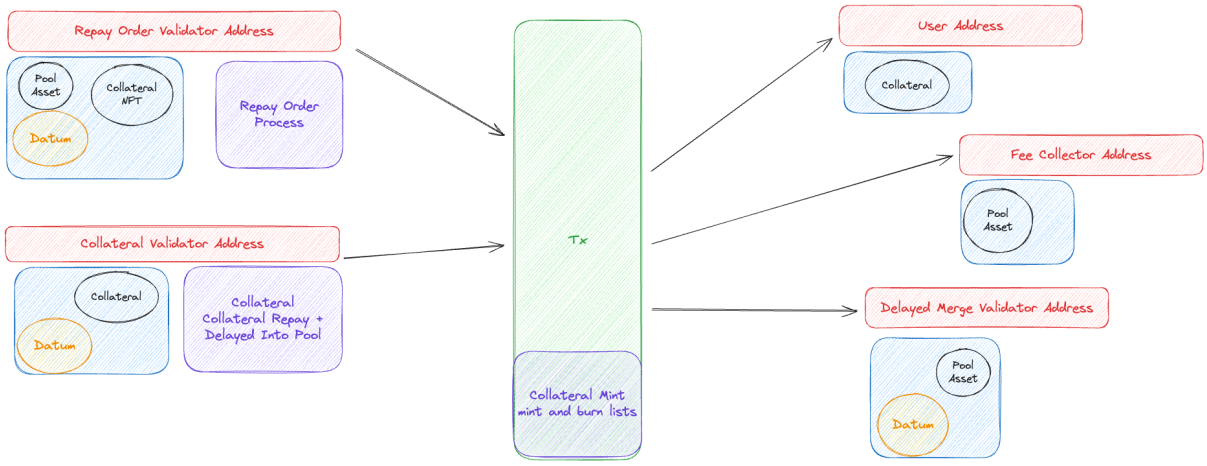
1.c.g - Pool Repay - Immediate With Pool

A user can repay their loan by providing the value of the loan along with the interest. The user gets their collateral back and the collateral NFT is burned in the process. The `pool` validator is used with the `spend` purpose and the redeemer is set to `Continuing - Close Loan`. Values from the `pool_config` are used and the pool config is used as a reference input. The platform fee is paid to the lenfi protocol as part of repaying the loan.



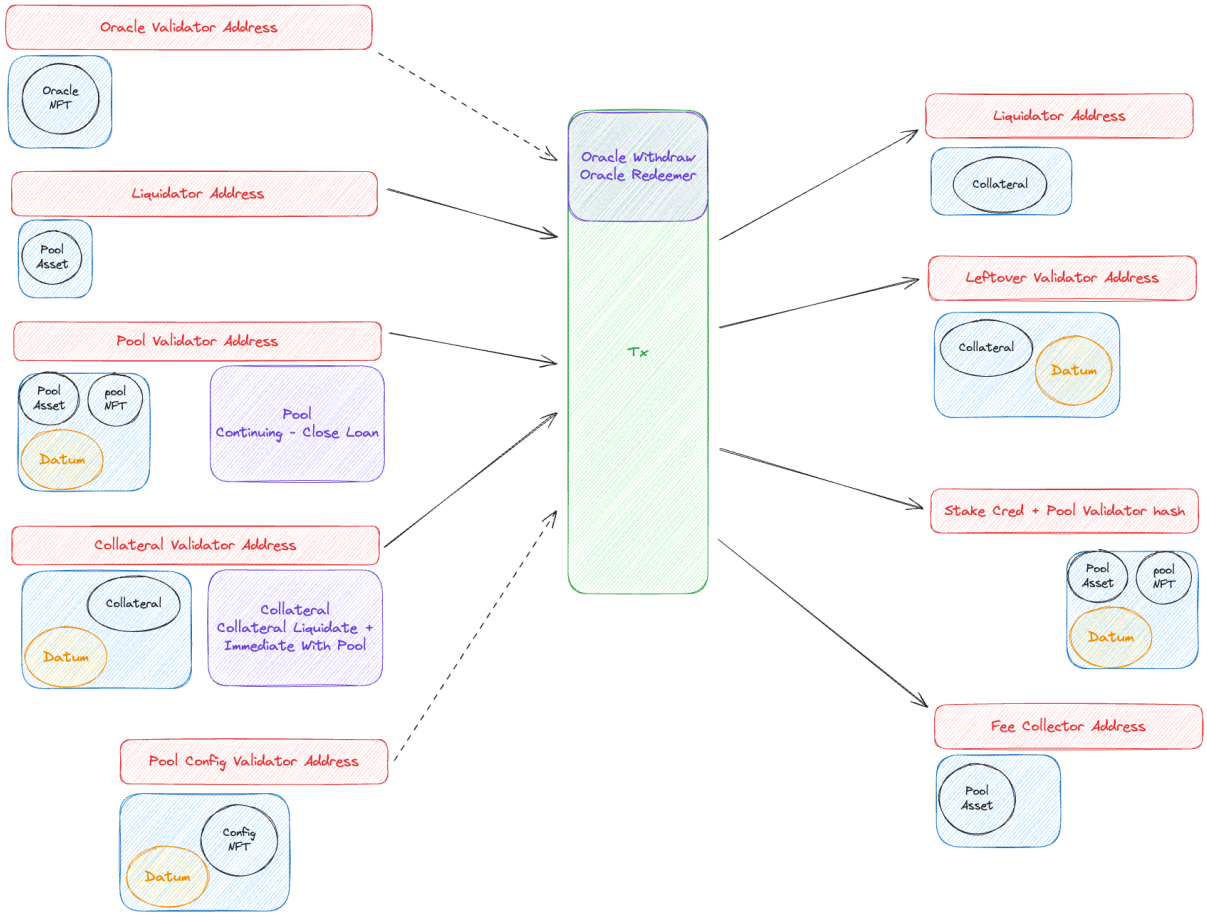
1.c.h - Pool Repay - Delayed Into Pool

Similar to `Pool Repay - Immediate With Pool` except the repaid value goes to a `delayed_merge` validator. This is useful for when the pool is experiencing high volume. The user still gets their collateral right away and the protocol fee is paid but the pool is adjusted later. The user has to pay a delayed merge fee.



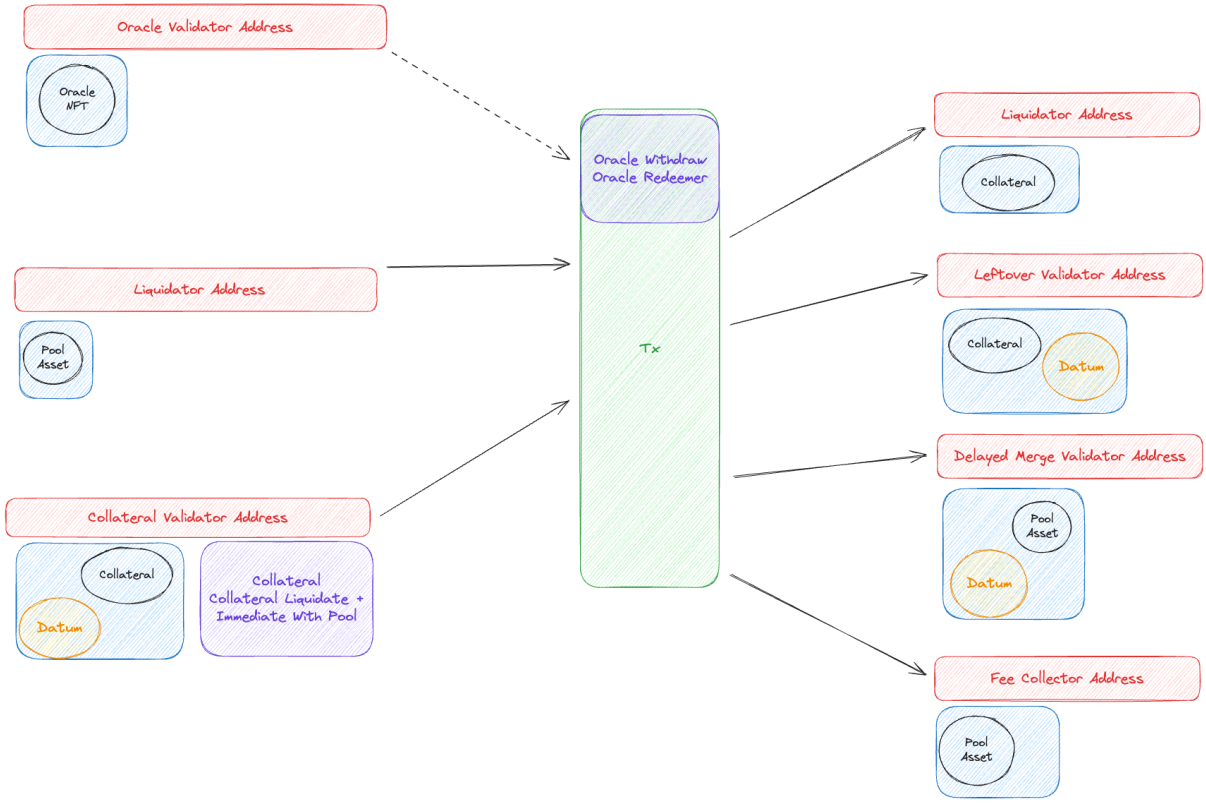
1.c.i - Pool Liquidate - Immediate With Pool

When a collateral position becomes less than the value of the loan amount plus a buffer based on the collateral ratio in the pool config. The liquidator gets 100% of the collateral ratio and the liquidator gets 2.5% of the leftovers. The remaining amount gets sent to the `leftover` validator whose datum contains a reference to the original collateral NFT allowing the borrower to get their final remaining collateral back. The pool liquidator has to pay a protocol fee.



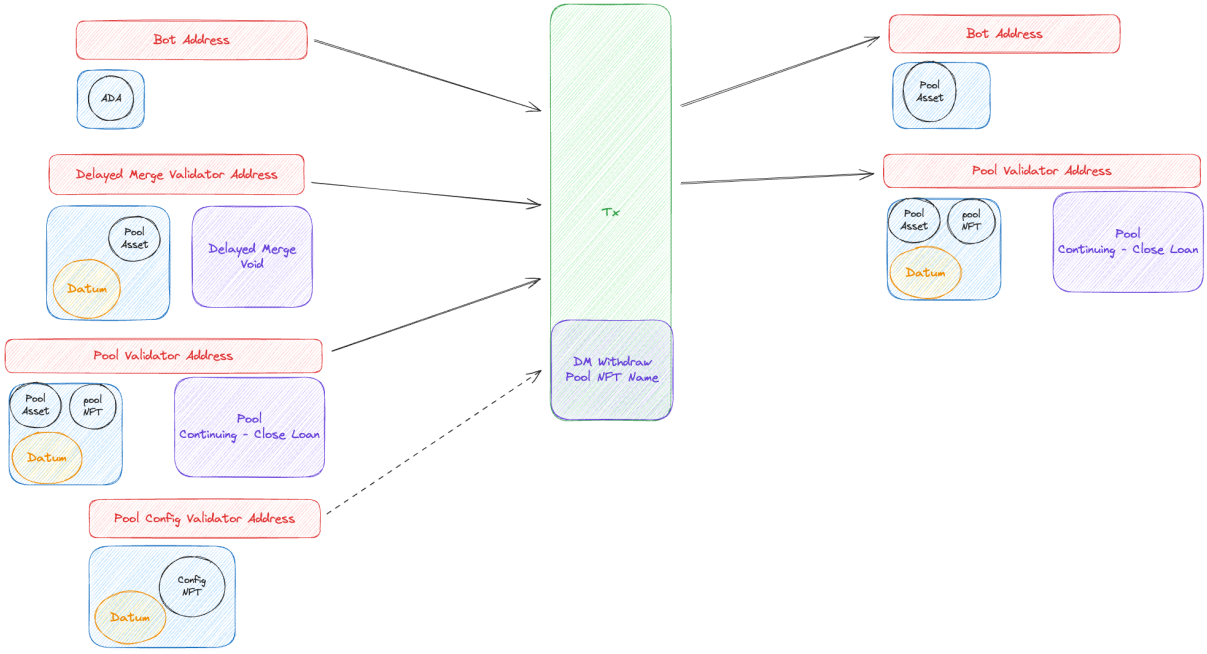
1.c.j - Pool Liquidate - Delayed Into Pool

Similar to `Pool Liquidate - Immediate With Pool` except the repaid value goes to a `delayed_merge` validator. This is useful for when the pool is experiencing high volume. The liquidator gets the same ratios right away and the leftovers are sent to the same place. The liquidator has to pay a delayed merge fee.



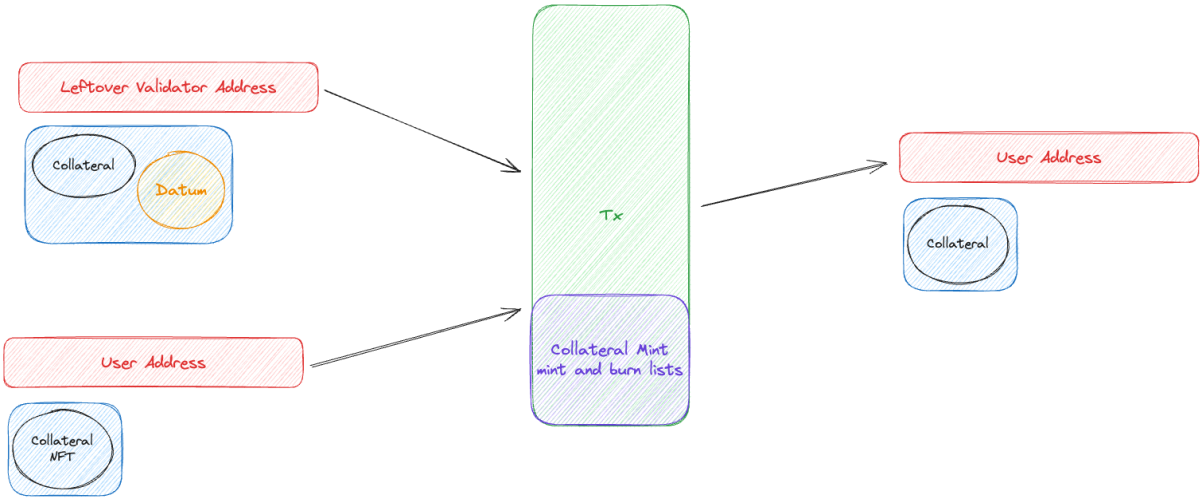
1.c.k - Delayed Merge

The `delayed_merge` validator accumulates loan repayments and liquidations that have not been merged into the pool. This transaction can have more than one input from the delayed merge validator address. The Batcher/Sequencer gets a fee for performing this transaction which is determined by the pool config.



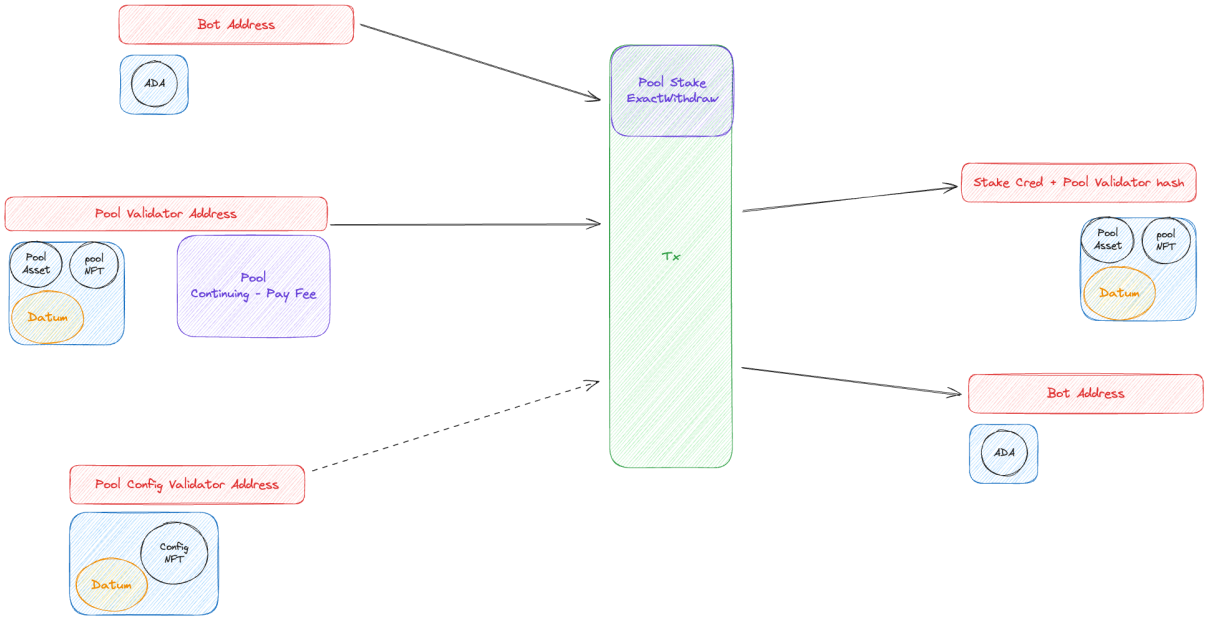
1.c.l - Leftover

A user burns their collateral NFT to claim any leftover collateral that remained after their loan was liquidated.



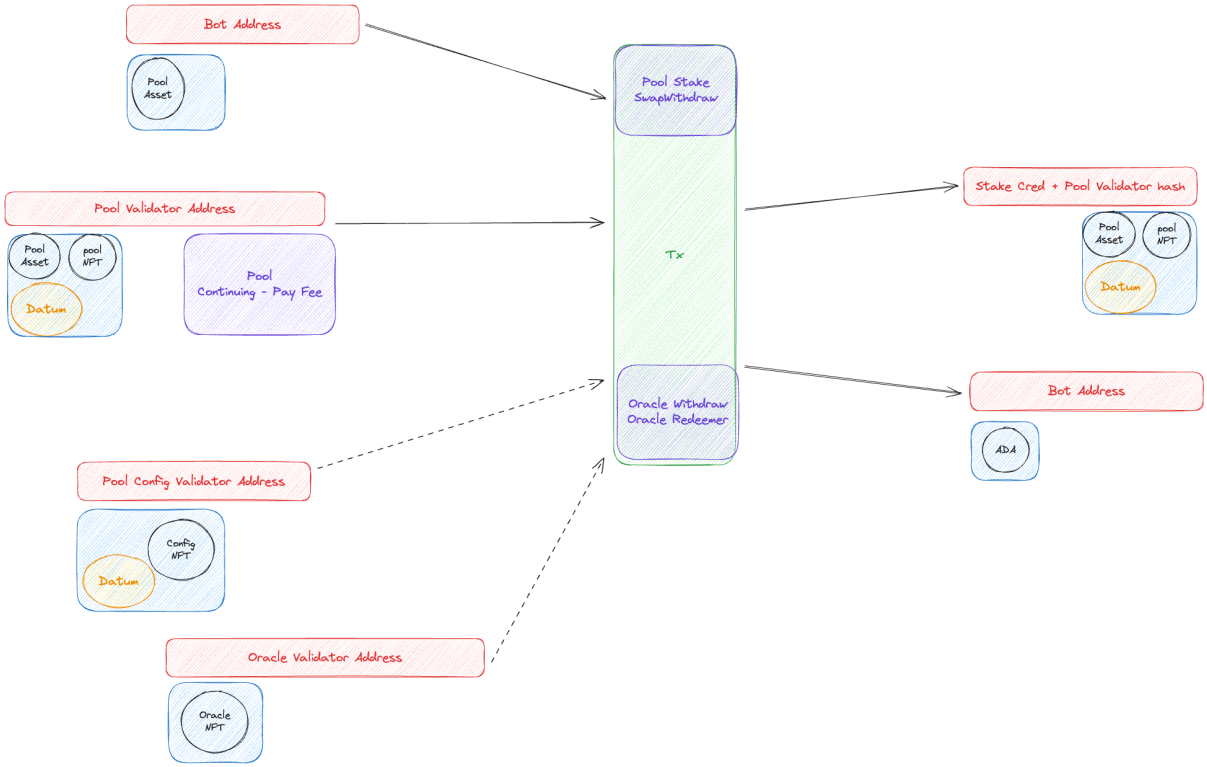
1.c.m - Pool Stake - Exact Withdraw

Any entity can run this transaction which will withdraw staking rewards. This is only valid for pools where the loanable pool asset is ADA. The withdraw reward must be greater than the minimum set in the pool config. The withdrawer gets to keep exactly two ADA as long as the minimum is met. The rest of the rewards go right into the pool.



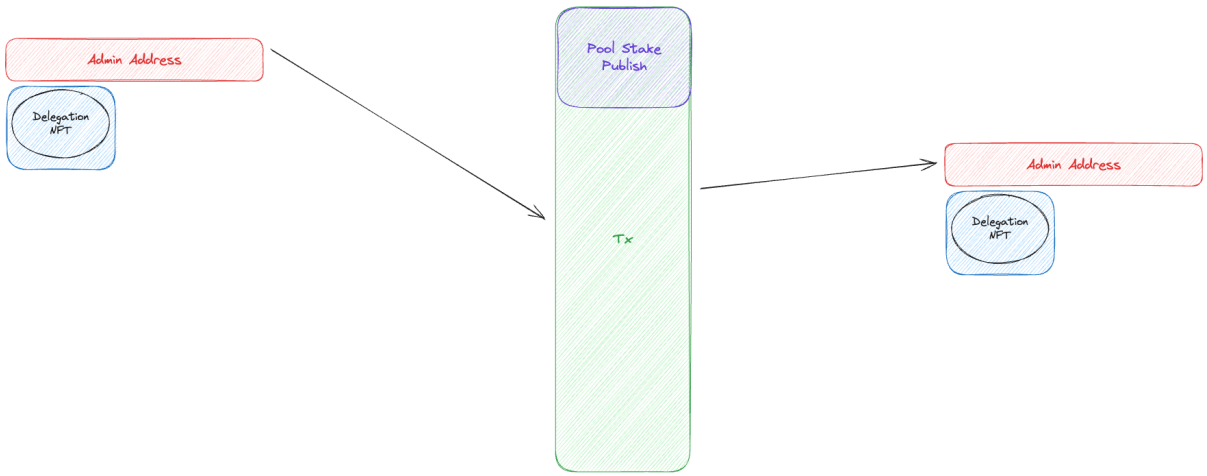
1.c.n - Pool Stake - Swap Withdraw

This transaction is similar to `Pool Stake - Exact Withdraw` but since the pool asset is not ADA, the oracle is used to calculate the ADA value compared to the pool asset found in the pool. The entity must deposit the pool asset where the amount is equal to the amount of withdrawn ADA minus two. The pool config also has a minimum set for this situation.



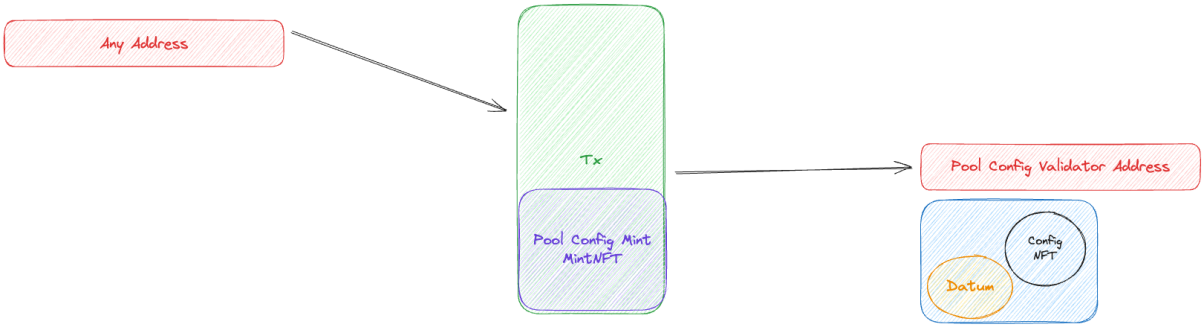
1.c.o - Pool Stake - Delegate/Deregister

The pool manager is allowed to delegate or deregister at anytime by spending the delegation NFT.



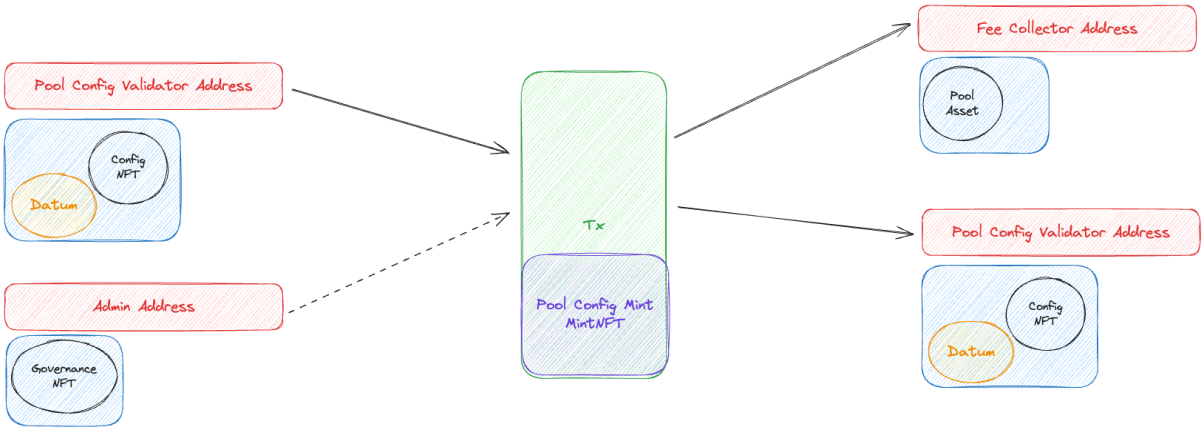
1.c.p - Pool Config - Create

This transaction just uses the `pool_config` mint validator with a redeemer of `MintNFT` to mint a config NFT and puts it in the `pool_config` validator address along with a datum. Other than checking the uniqueness of the NFT this validator performs no other validations.



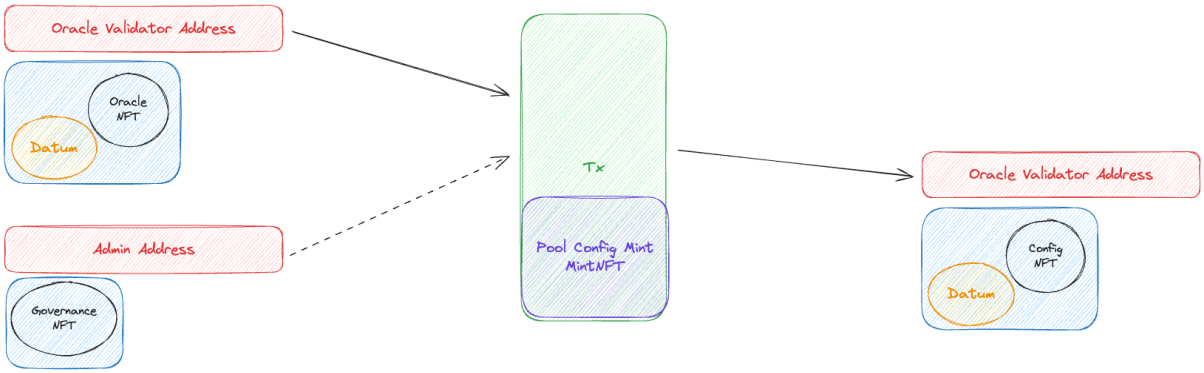
1.c.q - Pool Config - Update

A transaction to change the pool config parameters. If the governance NFT is at an address who's payment credential is a public key hash then the public key hash must be a signer of the transaction. If the governance NFT is at an address who's payment credential is a script hash then a withdraw validator matching that script hash must run in the same transaction. The `pool_config` validator will check that the fee collector address is a valid address.



1.c.r - Oracle - Update

A transaction to change the entire oracle validator. If the governance NFT is at an address who's payment credential is a public key hash then the public key hash must be a signer of the transaction. If the governance NFT is at an address who's payment credential is a script hash then a withdraw validator matching that script hash must run in the same transaction. The new oracle validator address can be any address and it does not need to be the same as the origin.



1.c.s - Files Audited

Below is a list of all files audited in this report, any files **not** listed here were **not** audited. The final state of the files for the purposes of this report is considered to be commit `c3899a600ca8c96df6d27e6f0ab6343e48fdabe7`.

Filename
validators/collateral.ak
validators/delayed_merge.ak
validators/leftovers.ak
validators/liquidity_token.ak
validators/oracle_validator.ak
validators/order_contract.ak
validators/pool_config.ak
validators/pool_stake.ak
validators/pool.ak
lib/aada/finance.ak
lib/aada/nft_pointers.ak
lib/aada/nft.ak
lib/aada/tests.ak
lib/aada/types.ak
lib/aada/utils.ak

2 - Findings

ID	Title	Severity	Status
LF-001	Multiple Pool NFTs in the same utxo in pool validator	Critical	Resolved
LF-002	DestroyPool action can be used to mint any amount of liquidity tokens	Critical	Resolved
LF-003	Partial Output validations from Order datums doesn't prevent Double Satisfaction	Critical	Resolved
LF-004	Values paid to delayed merge should check for repay amt	Critical	Resolved
LF-005	Pool deposit does not check that liquidity tokens go to user	Critical	Resolved
LF-006	Delayed Merge is vulnerable to double satisfaction	Critical	Resolved
LF-007	Collateral paying out to Delayed Merge contract is vulnerable to double satisfaction	Critical	Resolved
LF-008	Collateral Liquidate should use amount_to_repay instead of 1	Critical	Resolved
LF-009	Withdraw Exact does not check the pool asset to make sure its ada	Critical	Resolved
LF-100	Liquidations should be incentivized paying to the pool over delayed merge	Major	Resolved
LF-200	Fulfilling borrow order contract does account for lost ada due to min utxo constraint	Minor	Acknowledged
LF-201	More than one utxo at an address can be valid for the pool validator	Minor	Resolved

LF-300	Redundant `check_withdrawal_amount` in pool validator	Info	Resolved
LF-301	Use binop chain for `sign_check` in the pool validator	Info	Resolved
LF-302	Remove unused liquidation order contract	Info	Resolved
LF-303	Replace usage of list.length checks of 3 or less with when statements	Info	Resolved
LF-304	Redundant `consumed_utxo_check` in pool validator	Info	Resolved
LF-305	Can use an expect list_at instead of list.at in `output_correct` in pool validator	Info	Resolved
LF-306	Can simplify `id_from_utxo` to use serialise_data in nft.ak	Info	Resolved
LF-307	Check both for quantity and token name in liquidity validator	Info	Resolved
LF-308	BurnNFT action can hold duplicated items allowing for minting any collateral NFT	Info	Resolved
LF-309	`spend_check` is redundant since input is checked for existence	Info	Resolved
LF-311	Can replace usages of expect Some on list.head with expect [list_item, ..]	Info	Acknowledged
LF-312	`utils.do_oracle_calculation` does not need to return an option	Info	Resolved

3 - LF-001 Multiple Pool NFTs in the same utxo in pool validator

Category	Commit	Severity	Status
Exploit	286d46429c64340c5b0341e2d05a7fc57f3680a6	Critical	Resolved

3.a - Description

Redeemer output indexes can be duplicated allowing for multiple Pool NFTs in the same utxo in pool validator. This allows Pool NFTs to be stolen and then put into other validators.

3.b - Recommendation

First change the NFTRedeemer type to

```
NFTRedeemer {
  mints: List<Mints<CollateralNFTInner>>,
  burns: List<Burns<CollateralNFTInner>>,
}
```

For mints you would fold over the mints field and default value is the inputs. Mints would need to check the exact value output to the output index. Burn would fold over the burn field and the default value would be the transactions mint value specifically the mints under the own_address policy. This enforces that the burns are popped off from value one by one, preventing using the same burn validation for multiple redeemers. By checking exact value in the output index you prevent repeated output indexes from being valid.

The team resolved this by changing the pool mint validator to only allow one mint or burn per tx. This limits the pool to one destroy or create per tx which is not a bottleneck.

3.c - Resolution

Resolved in commit `0f7cd71a2f6eace839506f7b65a8e1c868a03a5a`

4 - LF-002 DestroyPool action can be used to mint any amount of liquidity tokens

Category	Commit	Severity	Status
Exploit	286d46429c64340c5b0341e2d05a7fc57f3680a6	Critical	Resolved

4.a - Description

The DestroyPool relies on the datum value to compare against minted liquidity tokens. However, the datum value can be chosen by a user if a pool token is stolen from the pool as in the vulnerabilities found above. This means that the DestroyPool action in the liquidity token validator can be used to mint any amount of liquidity tokens.

4.b - Recommendation

Change liquidity token validator to be per pool utxo instead of being universal for all pools.

4.c - Resolution

Resolved in commit `907e1c46045b82cf3bc3e3ce3c8cdafa215a685d`

5 - LF-003 Partial Output validations from Order datums doesn't prevent Double Satisfaction

Category	Commit	Severity	Status
Exploit	286d46429c64340c5b0341e2d05a7fc57f3680a6	Critical	Resolved

5.a - Description

A user with multiple orders can have one output that satisfies the partial output check from the order datum. The main concern is users with the public key can have identical partial outputs or partial outputs with a value that validates with a single output.

5.b - Recommendation

If the address contains a Verification key payment credential then the contract should check for a unique OutputReference in the datum to that output. For example:

```
let user_double_satisfaction_check =
  when address.payment_credential is {
    VerificationKey(_) -> {
      expect InlineDatum(datum_out) = datum
      expect out_ref: OutputReference = datum_out
      out_ref == this_oref
    }
    _ -> True
  }
```

5.c - Resolution

Resolved in commit `a7bed43818c717e88d0e5ec415bf6911c4ee43c6`

6 - LF-004 Values paid to delayed merge should check for repay amt

Category	Commit	Severity	Status
Exploit	286d46429c64340c5b0341e2d05a7fc57f3680a6	Critical	Resolved

6.a - Description

Currently there is no check for the delayed merge output to contain the repaid fees by the liquidator.

6.b - Recommendation

6.c - Resolution

Resolved in commit `f1e0cf1dcb6f7bb313075201f4d4a69001f75db4`

7 - LF-005 Pool deposit does not check that liquidity tokens go to user

Category	Commit	Severity	Status
Exploit	286d46429c64340c5b0341e2d05a7fc57f3680a6	Critical	Resolved

7.a - Description

Not checking that the liquidity tokens are sent to the address in the partial output specified in the datum would allow entities placing Pool deposits to steal liquidity tokens. The check is being done for withdrawal and borrow but not for deposit.

7.b - Recommendation

Check that the correct amount of liquidity tokens go to the partial output specified in the datum.

7.c - Resolution

Resolved in commit `83be0ce7b58def7b97dbae81e7cb3a15fd3629af`

8 - LF-006 Delayed Merge is vulnerable to double satisfaction

Category	Commit	Severity	Status
Exploit	286d46429c64340c5b0341e2d05a7fc57f3680a6	Critical	Resolved

8.a - Description

Delayed merges with matching utxo values are vulnerable to double satisfaction.

8.b - Recommendation

Use a withdraw validator to handle merging in multiple utxos into the pool at once.

8.c - Resolution

Resolved in commit `d7e2e92f367b4e1e2d87a9dfc6f7a035c5eb0f29`

9 - LF-007 Collateral paying out to Delayed Merge contract is vulnerable to double satisfaction

Category	Commit	Severity	Status
Exploit	286d46429c64340c5b0341e2d05a7fc57f3680a6	Critical	Resolved

9.a - Description

The collateral output to delayed merge can have multiple inputs that match to a single output.

9.b - Recommendation

Include the input output reference in the DelayedMerge datum.

9.c - Resolution

Resolved in commit `781fab8b2a4db1495802dee18fc4cb9adcd590f6`

10 - LF-008 Collateral Liquidate should use amount_to_repay instead of 1

Category	Commit	Severity	Status
Exploit	286d46429c64340c5b0341e2d05a7fc57f3680a6	Critical	Resolved

10.a - Description

The amount_to_repay for liquidate returns 1 instead of the actual amount_to_repay

10.b - Recommendation

Change `Some(1 |> rational.from_int)` to `Some(amount_to_repay)`

10.c - Resolution

Resolved in commit `06302e75cc796e22fd7429adcf1b98b2d823d9ab`

11 - LF-009 Withdraw Exact does not check the pool asset to make sure its ada

Category	Commit	Severity	Status
Exploit	286d46429c64340c5b0341e2d05a7fc57f3680a6	Critical	Resolved

11.a - Description

Withdraw exact can be used on non-ADA lending pools which for assets with a high supply it allows the withdrawer to use the high supply asset to satisfy the fee amount. This would be far less than the ADA they could withdraw.

11.b - Recommendation

Check that the pool asset is ADA. Do not allow withdraw exact on non-ADA pools.

11.c - Resolution

Resolved in commit `76fe120fb1007d96fcfb2b3fe8f974a2dd31305d`

12 - LF-100 Liquidations should be incentivized paying to the pool over delayed merge

Category	Commit	Severity	Status
Bug	286d46429c64340c5b0341e2d05a7fc57f3680a6	Major	Resolved

12.a - Description

A liquidation can always pay to delayed merge over the pool. This could possibly put the pool in a place of bad debt as lenders are delayed in receiving payment for their lends.

12.b - Recommendation

Pay an extra fee when using delayed merge to incentivize paying to the pool if possible. Also possible (and recommended over the above sentence) is to use a withdraw validator to handle merging in multiple utxos into the pool at once.

12.c - Resolution

Resolved in commit `2244539419b2b28356f2139ded0599ee6c7efe18`

13 - LF-200 Fulfilling borrow order contract does account for lost ada due to min utxo constraint

Category	Commit	Severity	Status
Bug	286d46429c64340c5b0341e2d05a7fc57f3680a6	Minor	Acknowledged

13.a - Description

The borrow order contract does not account for the min utxo constraint when fulfilling the order. This could result in the batcher/sequencer using their own ADA to fulfill the order if they are not careful.

13.b - Recommendation

Enforce that some ADA is set aside so that the batcher/sequencer can have a buffer for the transaction fee.

13.c - Resolution

Resolved in commit `N/A`

14 - LF-201 More than one utxo at an address can be valid for the pool validator

Category	Commit	Severity	Status
Bug	286d46429c64340c5b0341e2d05a7fc57f3680a6	Minor	Resolved

14.a - Description

Currently a user could create a new pool utxo at the same address. While this does not have direct effect on funds store at the utxo, this could be used to confuse the backend or users. I would advise to enforce that an address can only mint one valid pool utxo per address.

14.b - Recommendation

Introduce a new redeemer action for the pool stake validator called create pool and parameterize the the pool stake validator by a input ref. The new redeemer action should check that the inputs contain the input ref and that the redeemer itself holds the same input ref. Then the create pool action on the pool mint validator can look up the output address stake credential, find the redeemer, and ensure the pool nft name matches the input ref specified by the pool stake validator redeemer. This way the only way to create a pool is to tie the creation of the pool to the stake address. Ensuring that an address has only one valid pool utxo per address.

14.c - Resolution

Resolved in commit `4b22f0af2d02f2e2c966edee4f5963744444caee`

15 - LF-300 Redundant `check_withdrawal_amount` in pool validator

Category	Commit	Severity	Status
Redundant	286d46429c64340c5b0341e2d05a7fc57f3680a6	Info	Resolved

15.a - Description

The `check_withdrawal_amount` value is redundant since it appears twice in the pool validator.

15.b - Recommendation

Remove the first usage of `check_withdrawal_amount` in the pool validator.

15.c - Resolution

Resolved in commit `ca7d987b74823cb287f10cabcfa6feefb44c3c8a`

16 - LF-301 Use binop chain for `sign_check` in the pool validator

Category	Commit	Severity	Status
Readability	286d46429c64340c5b0341e2d05a7fc57f3680a6	Info	Resolved

16.a - Description

The readability of the `sign_check` in the pool validator can be improved by using a binop chain.

16.b - Recommendation

Use and {...} and or {...} instead of && and ||.

16.c - Resolution

Resolved in commit `5d0213057135636053967 6a5c8629540006df556`

17 - LF-302 Remove unused liquidation order contract

Category	Commit	Severity	Status
Readability	286d46429c64340c5b0341e2d05a7fc57f3680a6	Info	Resolved

17.a - Description

The liquidation contract is redundant since collateral can be liquidated without the usage of the contract.

17.b - Recommendation

Remove the liquidation contract.

17.c - Resolution

Resolved in commit `732c37d418685990a4934f089c20edf4f543443f`

18 - LF-303 Replace usage of list.length checks of 3 or less with when statements

Category	Commit	Severity	Status
Optimize	286d46429c64340c5b0341e2d05a7fc57f3680a6	Info	Resolved

18.a - Description

The usage of list.length checks of 3 or less can be replaced with when statement, because the list.length has a greater cost since it needs to traverse the entire list.

18.b - Recommendation

Use a `when/is` expression to check list branches with three or less items.

18.c - Resolution

Resolved in commit `956f092a50badef32fa371cb137359b59131a4a1`

19 - LF-304 Redundant `consumed_utxo_check` in pool validator

Category	Commit	Severity	Status
Redundant	286d46429c64340c5b0341e2d05a7fc57f3680a6	Info	Resolved

19.a - Description

Calling list.tail on pruned_inputs and checking for Some(..) does the same action as checking the list.length > 0

19.b - Recommendation

Resolved in a different way by removing pruned_input entirely and checking for the existence of the pool_utxo in the inputs.

19.c - Resolution

Resolved in commit `2d7e0eac74bd2b38c189cda6b633cded729ac222`

20 - LF-305 Can use an expect list_at instead of list.at in `output_correct` in pool validator

Category	Commit	Severity	Status
Optimize	286d46429c64340c5b0341e2d05a7fc57f3680a6	Info	Resolved

20.a - Description

Finding the index of an output and then mapping over an `Option` is less efficient.

20.b - Recommendation

Instead of finding the index and then mapping over the `Option` to check if an output is correct, you can index directly to the item.

20.c - Resolution

Resolved in commit `2d7e0eac74bd2b38c189cda6b633cded729ac222`

21 - LF-306 Can simplify `id_from_utxo` to use serialise_data in nft.ak

Category	Commit	Severity	Status
Optimize	286d46429c64340c5b0341e2d05a7fc57f3680a6	Info	Resolved

21.a - Description

Currently the function `id_from_utxo` pulls the fields out of transaction_id and concatenates them and then hashes the result.

21.b - Recommendation

A more optimal way to produce an id is to use the `serialise_data` builtin and then hash the result.

21.c - Resolution

Resolved in commit `2a1a1fcd2c442a783d271b823f88ccaf51bd9285`

22 - LF-307 Check both for quantity and token name in liquidity validator

Category	Commit	Severity	Status
Optimize	286d46429c64340c5b0341e2d05a7fc57f3680a6	Info	Resolved

22.a - Description

Currently the liquidity validator checks for the pool_nft_token in two locations, `correct_pooltoken_mint` and `expect [pool_token_name]`. There is a simplified form in the recommendation.

22.b - Recommendation

Use this to get the same result:

```
expect [(pool_token_name, 1)] = dict.to_list(tokens(mint, pool_hash))
```

22.c - Resolution

Resolved in commit `6fe7d68d6108151e164ed11804387255c381d222`

23 - LF-308 BurnNFT action can hold duplicated items allowing for minting any collateral NFT

Category	Commit	Severity	Status
Exploit	286d46429c64340c5b0341e2d05a7fc57f3680a6	Info	Resolved

23.a - Description

This was a finding from another audit. This describes a recommendation to the issue presented in the other audit.

23.b - Recommendation

First change the NFTRedeemer type to

```
NFTRedeemer {
  mints: List<Mints<CollateralNFTInner>>,
  burns: List<Burns<CollateralNFTInner>>,
}
```

For mints you would fold over the mints field and default value is the inputs. Mints would need to check the exact value output to the output index. Burn would fold over the burn field and the default value would be the transactions mint value specifically the mints under the own_address policy. This enforces that the burns are popped off from value one by one, preventing using the same burn validation for multiple redeemers. By checking exact value in the output index you prevent repeated output indexes from being valid.

23.c - Resolution

Resolved in commit `0f7cd71a2f6eace839506f7b65a8e1c868a03a5a`

24 - LF-309 `spend_check` is redundant since input is checked for existence

Category	Commit	Severity	Status
Optimize	286d46429c64340c5b0341e2d05a7fc57f3680a6	Info	Resolved

24.a - Description

In collateral NFT validator, checking for a redeemer is redundant since the input is checked for existence.

24.b - Recommendation

Remove the redundant check.

24.c - Resolution

Resolved in commit `9240e3e1c5f9f2e0b0e3dfc61887dbae90f4a9e6`

25 - LF-311 Can replace usages of expect Some on list.head with expect [list_item, ..]

Category	Commit	Severity	Status
Optimize	286d46429c64340c5b0341e2d05a7fc57f3680a6	Info	Acknowledged

25.a - Description

It is less efficient to call `list.head` with `expect Some(_)` than just directly calling `expect` on the list pattern directly.

25.b - Recommendation

```
expect Some(list_value) = list.head(list_items)
```

can be replaced with

```
expect [list_value, ..] = list_items
```

25.c - Resolution

Resolved in commit `N/A`

26 - LF-312 `utils.do_oracle_calculation` does not need to return an option

Category	Commit	Severity	Status
Optimize	286d46429c64340c5b0341e2d05a7fc57f3680a6	Info	Resolved

26.a - Description

All current usages of this function expect or preemptively prevent the none case.

26.b - Recommendation

Return an `Int` directly instead of `Option<Int>`.

26.c - Resolution

Resolved in commit `b39fafc1497f989d7209369c5a0e53de9386c6f8`

27 - Appendix

27.a - Disclaimer

This report is governed by the terms in the agreement between TxPipe (**TXPIPE**) and Lenfi (**CLIENT**). This report cannot be shared, referred to, altered, or relied upon by any third party without TXPIPE's written consent. This report does not endorse or disapprove any specific project, team, code, technology, asset or similar. It provides no warranty or guarantee about the quality or nature of the technology analyzed.

TXPIPE DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, related to this report, its content, and the related services and products. This report is provided as-is. TxPipe does not take responsibility for any product or service advertised or offered by Client or any third party. **TXPIPE IS NOT RESPONSIBLE FOR MONITORING ANY TRANSACTION BETWEEN YOU AND CLIENT AND/OR ANY THIRD-PARTY PROVIDERS OF PRODUCTS OR SERVICES.**

This report should not be used for making investment or involvement decisions with any project, services or assets. This report provides general information and is not a form of financial, investment, tax, legal, regulatory, or other advice.

TxPipe created this report as an informational review of the due diligence performed on the Client's smart contract. This report provides no guarantee on the security or operation of the smart contract on deployment or post-deployment. **TXPIPE HAS NO DUTY TO MONITOR CLIENT'S OPERATION OF THE PROJECT AND UPDATE THE REPORT ACCORDINGLY.**

The information in this report may not cover all vulnerabilities. This report represents an extensive assessment process intended to help increase the quality of the Client's code. However, blockchain technology and cryptographic assets present a high level of ongoing risk, including unknown risks and flaws.

TxPipe recommends multiple independent audits, a public bug bounty program, and continuous security auditing and monitoring. Errors in the manual review process are possible, and TxPipe advises seeking multiple independent opinions on critical claims. **TXPIPE BELIEVES EACH COMPANY AND INDIVIDUAL IS RESPONSIBLE FOR THEIR OWN DUE DILIGENCE AND CONTINUOUS SECURITY.**

27.b - Issue Guide

27.b.a - Severity

Severity	Description
Critical	Critical issues highlight exploits, bugs, loss of funds, or other vulnerabilities that prevent the dApp from working as intended. These issues have no workaround.
Major	Major issues highlight exploits, bugs, or other vulnerabilities that cause unexpected transaction failures or may be used to trick general users of the dApp. dApps with Major issues may still be functional.
Minor	Minor issues highlight edge cases where a user can purposefully use the dApp in a non-incentivized way and often lead to a disadvantage for the user.
Info	Info are not issues. These are just pieces of information that are beneficial to the dApp creator. These are not necessarily acted on or have a resolution, they are logged for the completeness of the audit.

27.b.b - Status

Status	Description
Resolved	Issues that have been **fixed** by the **project** team.
Acknowledged	Issues that have been **acknowledged** or **partially fixed** by the **project** team. Projects can decide to not **fix** issues for whatever reason.
Identified	Issues that have been **identified** by the **audit** team. These are waiting for a response from the **project** team.

27.c - Revisions

This report was created using a git based workflow. All changes are tracked in a github repo and the report is produced using typst. The report source is available here. All versions with downloadable PDFs can be found on the releases page.

27.d - About Us

TxPipe is a blockchain technology company responsible for many projects that are now a critical part of the Cardano ecosystem. Our team built Oura, Scrolls, Pallas, Demeter, and we're the original home of Aiken. We're passionate about making tools that make it easier to build on Cardano. We believe that blockchain adoption can be accelerated by improving developer experience. We develop blockchain tools, leveraging the open-source community and its methodologies.

27.d.a - Links

- Website
- Email
- Twitter