April 26, 2024

Sundae Labs - V3

Contents

1 - Summary

This report provides a comprehensive audit of SundaeSwap V3, a decentralized exchange protocol that realizes an automated market maker (AMM) pooled liquidity model on the Cardano Blockchain.

The investigation spanned several potential vulnerabilities, including scenarios where attackers might exploit the validator to lock up or steal funds.

The audit is conducted without warranties or guarantees of the quality or security of the code. 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

The core component of SundaeSwap V3 is the liquidity pool. Liquidity pools are script UTxOs that hold liquidity for two fixed assets. Standard operations are supported, such as swapping and providing/removing liquidity, together with more advanced operations.

To address concurrency, users does not interact directly with LPs but place orders. Orders are script UTxOs that hold all the assets and information required for the execution of the desired operation. They can be directed to a specific pool, or open to any pool that is able to process it.

Orders are processed in batches in "scoop" transactions by authorized entities called "scoopers". A scoop transaction applies a sequence of orders to a specific pool, transforming the pool state and doing all the required payments to fulfill the orders purpose.

SundaeSwap V3 protocol is booted by the creation of a single settings UTxO that governs the entire protocol. The settings UTxO determine, among other things, the list of authorized scoopers. Liquidity pools are created and validated with the minting of a pool NFT.

Orders are created with no validation, so it is up to the scoopers to select well-formed orders to be processed. There are several order types:

- Swap: to swap one token for another.
- Deposit: to provide liquidity and obtain LP tokens.
- Withdrawal: to redeem LP tokens and remove liquidity.
- Donation: to provide liquidity for free.
- Strategy: to lock funds for an operation that will be determined at processing time by a designated signer.
- Record: to create an output that can be used to do a snapshot of the pool state (than can be used later as an oracle).

1.b - Process

Our audit process involved a thorough examination of SundaeSwap V3 validators. Areas vulnerable to potential security threats were closely scrutinized, including those where attackers could exploit the validator's functions to disrupt the platform and its users. This included evaluating potential risks such as unauthorized asset addition, hidden market creation, and disruptions to interoperability with other Plutus scripts. This also included the common vulnerabilities such as double satisfaction and minting policy vulnerabilities.

The audit took place over a period of several weeks, and it involved the evaluation of the platform's mathematical model to verify that the implemented equations matched those of the AMM algorithm.

Findings and feedback from the audit were communicated regularly to the SundaeSwap team through Discord. Diagrams illustrating the necessary transaction structure for proper interaction with SundaeSwap V3 are attached as part of this report. The SundaeSwap team addressed these issues in an efficient and timely manner, enhancing the overall security of the platform.

2 - Specification

2.a - UTxOs

2.a.a - Settings UTxO

A single script UTxO that is created at launch and used for the entire protocol. Creation is validated with the minting of the "Settings NFT" that is locked into the UTxO. A multivalidator is used to contain both the spend and the minting validators.

- Address: hash of multivalidator in `settings.ak`. Parameters:
  - `protocol_boot_utxo`: reference to a UTxO that must be spent at settings creation.
- Value:
  - ADA: only min ADA.
  - Settings NFT: minting policy defined in the multivalidator.
- Datum: `SettingsDatum`

2.a.b - Pool UTxOs

One script UTxO for each liquidity pool. All pools in the protocol share the same address. Liquidity pool creation is validated with the minting of a "Pool NFT" that is locked into the UTxO. A multivalidator is used to contain both the spend and the minting validators. Moreover, the minting validator is used both for the Pool NFT and for the LP tokens.

- Address: hash of multivalidator in pool.ak. Parameters:
  - `manage_stake_script_hash`: hash of staking script that validates pool management operations.
  - `settings_policy_id`: minting policy of the Settings NFT.
- Value:
  - ADA: accumulated protocol fees (including min ADA).
  - (A, B): pair of assets contained by the pool. A may be ADA.
  - Pool NFT: minting policy defined in the multivalidator.
- Datum: `PoolDatum`

2.a.c - Order UTxOs

One script UTxO per order. All orders in the protocol share the same address. Order creation is not validated. Order spending is validated to be done by the order creator or by a transaction that involves spending a liquidity pool. The latter check is done by a staking validator that is referenced in the spend validator. This way, the check is done only once for the transaction and not one time for each spent order, optimizing mem/CPU usage.

- Address: hash of spend validator in `order.ak`. Parameters:
  - `stake_script_hash`: hash of staking script that validates for the presence of a valid liquidity pool.
- Value:
  - ADA: at least min ADA.
  - Other: assets relevant to the order + others.
- Datum: `OrderDatum`

2.a.d - Oracle UTxOs

A UTxO that can be created as the result of processing an order of type "Record", if the correct parameters for the order are used. Creation is validated with the minting of the "Oracle NFT" that is locked into the UTxO, to check that the datum contains the correct information regarding the pool state. The UTxO can be spent by an owner defined in the order. A multivalidator is used to contain both the spend and the minting validators.

- Address: hash of multivalidator in `oracle.ak`. Parameters:
  - `pool_script_hash`: hash of pool multivalidator.
- Value:
  - ADA: at least min ADA.
  - Oracle NFT: minting policy with same hash as this oracle script.
- Datum: `OracleDatum`

2.b - Transactions

2.b.a - Settings

2.b.a.a - Operation “create settings”

This transaction creates a settings UTxO, which is then referenced by several pool operations that need those protocol settings as part of their validation.

The UTxO contains a datum with the protocol settings, and its value has a NFT used to identify it, which is minted in this create transaction.

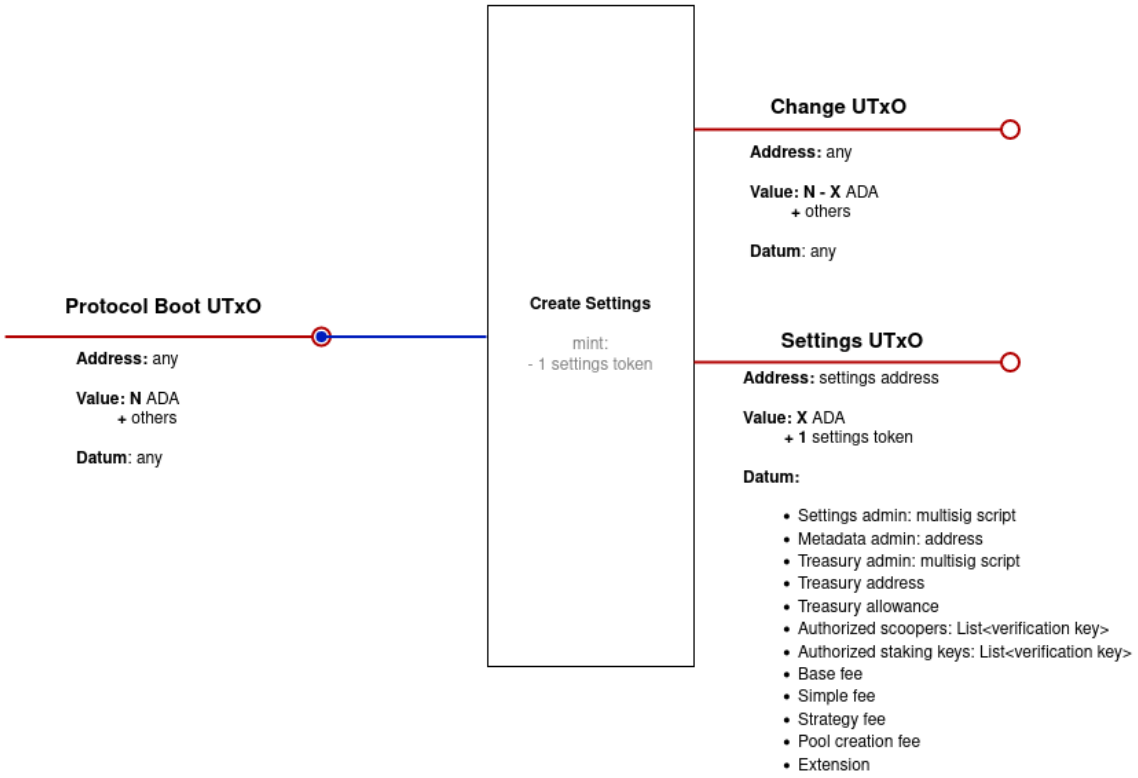

Figure 1: Create Settings diagram.

Code:

- settings.ak:mint()

Expected Failure Scenarios:

- The protocol boot UTxO is not being spent
- More than one settings tokens is being minted, or any other asset

2.b.a.b - Operation “update settings”

This transaction colapses two updates of different nature: ones allowed to the settings administrator and others to the treasury administrator. Each one of those can update different fields of the settings datum.

The two involved redeemers are:

- `SettingsAdminUpdate` for the settings admin
- `TreasuryAdminUpdate` for the treasury admin

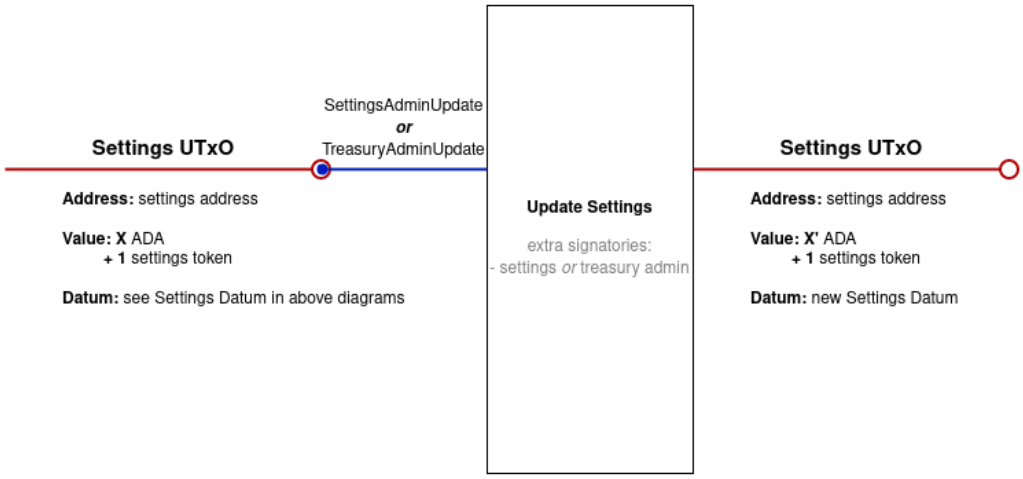

Figure 2: Update Settings diagram.

Code:

- settings.ak:spend():SettingsAdminUpdate
- settings.ak:spend():TreasuryAdminUpdate

Expected Failure Scenarios:

- There’s a minting or burning happening in the transaction
- The settings NFT is stolen from the settings UTxO
- The settings input and output have different addresses
- If the redeemer being executed is the `SettingsAdminUpdate`: other than the following fields are updated
  - settings_admin
  - metadata_admin
  - treasury_admin
  - authorized_scoopers
  - base_fee, simple_fee, strategy_fee, pool_creation_fee
  - extensions

  else if `TreasuryAdminUpdate` is being executed, other than the following fields are updated:
  - treasury_address
  - authorized_staking_keys
  - extensions?
- The tx is not signed by the given administrator: `SettingsAdminUpdate` signed by settings admin, or `TreasuryAdminUpdate` by the treasury admin

2.b.b - Pools

2.b.b.a - Operation “create pool”

This transaction creates a Pool UTxO based on the settings UTxO, which provides various protocol configurations, and funds transfered from the pool creator that act as the initial liquidity. Also, a UTxO which will hold the metadata associated with the pool is created, although the actual metadata information is uploaded in a subsequent transaction that will be performed by the metadata admin.

The minted assets are:

- pool NFT: held within the pool UTxO
- pool reference NFT: used to identify the UTxO that will hold the metadata associated to the pool
- LP tokens: paid to the pool creator. These tokens represent the amount of liquidity provided by the creator

The involved redeemers are:

- `CreatePool`
- `MintLP`

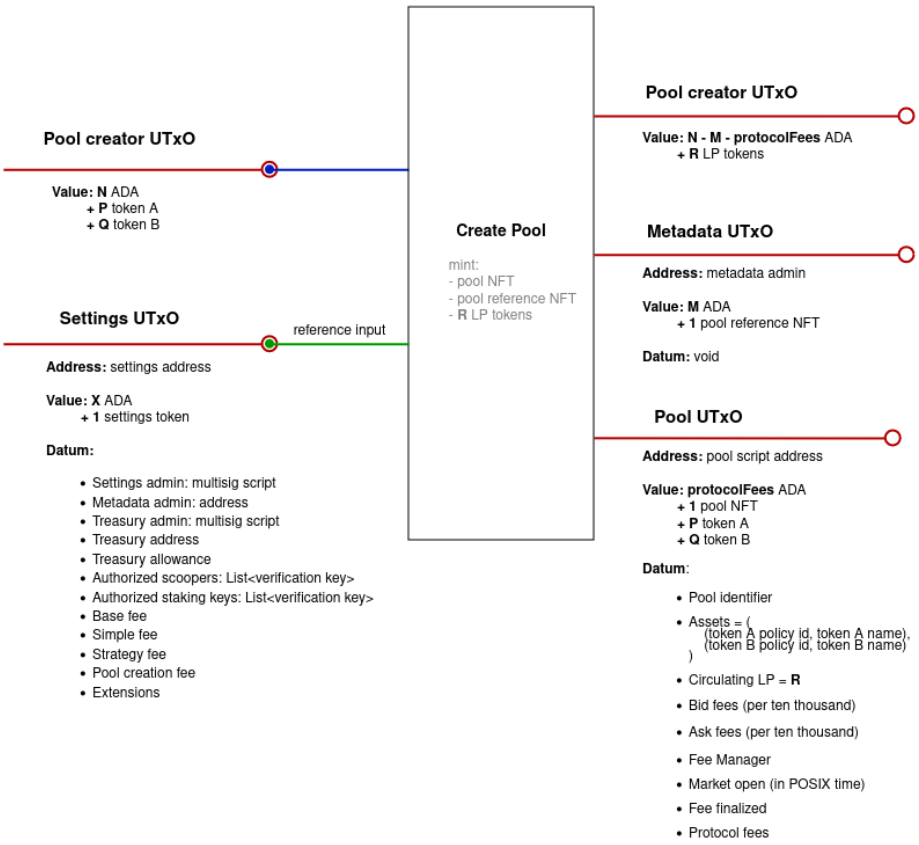

Figure 3: Create Pool diagram.

Code:

- pool.ak:mint():CreatePool

Expected Failure Scenarios:

- Pool output has the expected address: the pool script address
- Quantities of both tokens of the pair are not a positive integer
- Pool reference NFT is not paid to the metadata output
- Pool NFT or specified quantity of both tokens of the pair are not paid to the pool script address

- Pool value does have more assets than the relevant ones: the pool NFT, protocolFees ADA, and both tokens from the pair
- Pool datum is not valid. One of:
  - Pool identifier is not correct based on the rules defined for ensuring uniqueness
  - The assets property does not match with the tokens pair provided to the pool UTxO
  - Circulating LP property does not equal the minted quantity of LP tokens
  - Fees per ten thousand property is not a positive integer
  - Protocol fees is not a positive integer
- Metadata output does not have a void datum
- The settings UTxO doesn't have a token with the expected policy ID (parameter of the validator)

2.b.b.b - Operation "withdraw fees"

This transaction allows the treasury administrator to take a specific amount of ADA from the UTxO pool accumulated there in protocol fees. This withdrawn amount is then paid to the treasury address minus a portion (the allowance) that this admin can pay wherever he wants.

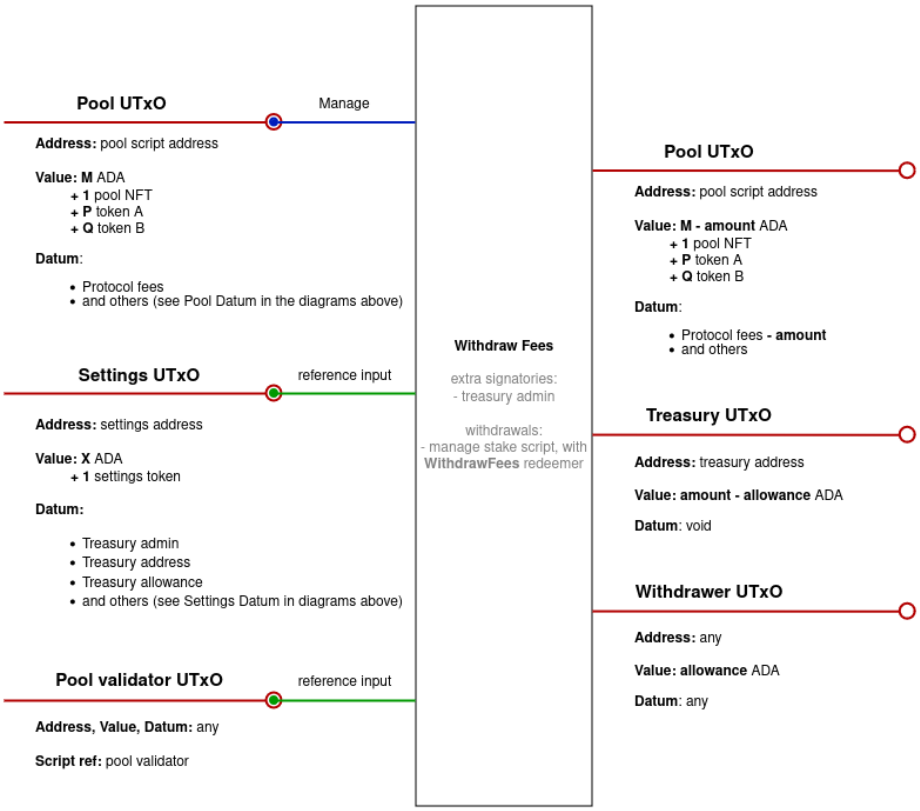

Figure 4: Withdraw Fees diagram.

Code:

- pool.ak:manage():WithdrawFees

Expected Failure Scenarios:

- Pool input address and Pool output address are distinct
- In Pool output Datum, any other field than the protocol fees one is updated
- The amount to withdraw specified within the redeemer does not match the amount of ADA taken from the Pool UTxO, or any of the other assets quantities of the UTxO change
- The transaction is not signed by the treasury administrator
- The treasury part (withdraw amount minus allowance) is not paid to the treasury address
- The amount to withdraw is greater than the available protocol fees in the Pool UTxO

2.b.b.c - Operation “update pool fees”

This transaction allows the Pool fees manager to update the bid and/or ask fee amount, information that’s stored in the Pool Datum.

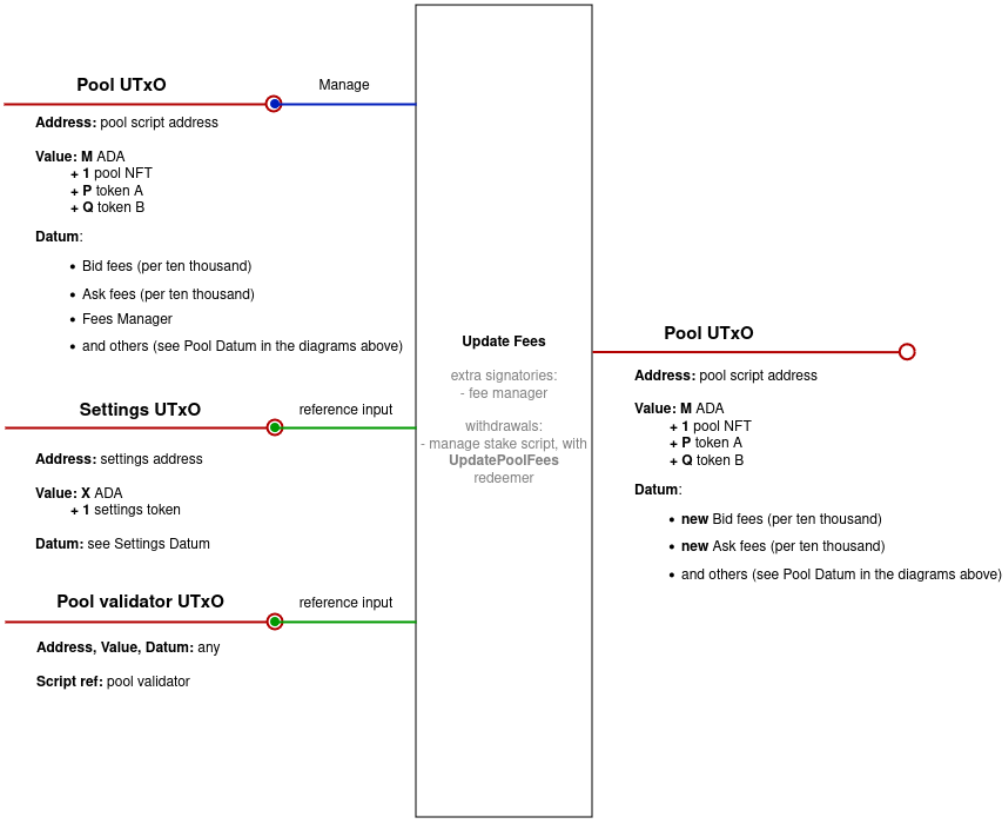

Figure 5: Update Pool Fees diagram.

Code:

- pool.ak:manage():UpdatePoolFees

Expected Failure Scenarios:

- Pool input address and Pool output address are distinct
- Pool input value and Pool output value are distinct
- In Pool output Datum, any other field than the bid and/or ask fees is updated
- If one of bid/ask fee field is updated, it is out of the valid percentage range: less than 0% or more than 100%
- The Pool fees manager is not signing the transaction

2.b.b.d - Operation “close pool”

This transaction lets the treasury administrator withdraw the remaining ADA of the pool, given that it has no liquidity left. The pool NFT must be burnt.

Code:

- pool.ak:mint():BurnPool
- pool.ak:manage():WithdrawFees by this branch.

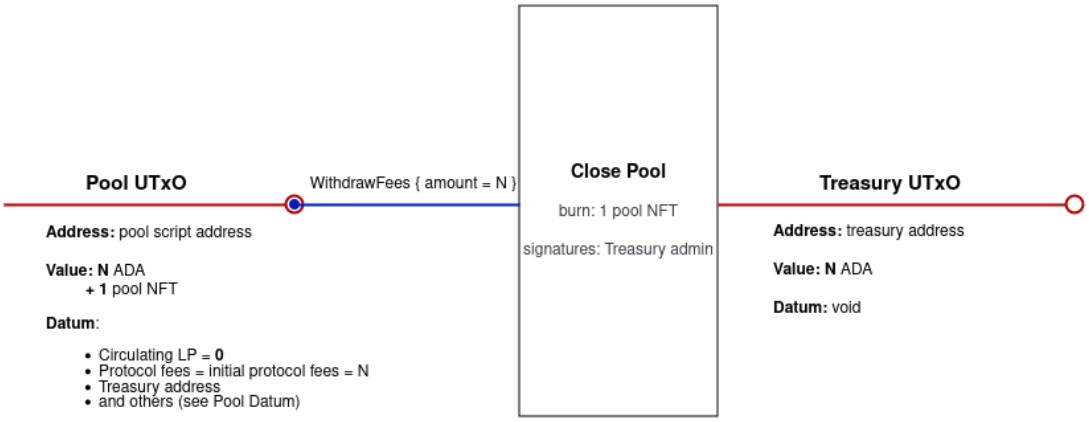

Figure 6: Close Pool diagram.

Expected Failure Scenarios:

- Pool NFT is not burned.
- Transaction is not signed by the treasury administrator.
- Pool remaining ADA are not paid to the treasury address.
- There's LP in circulation.
- Pool has other asset than the pool NFT and ADA, or has more ADA than initial protocol fees ADA.

2.b.c - Orders

2.b.c.a - Operation "create order"

This transaction transfers funds from the user to the Order script address, whose associated validator will then be executed to unlock them. This transaction does not require a validator execution and stores a datum that contains information about the type of Order the user made.

The assets sent are different depending on the type of Order but they all contain at least the maximum protocol fee ADA, the assets relevant to the specific type of Order plus any other assets (optional).

More precisely, the assets relevant to the specific type of Order are:

- Swap: the offered asset given in exchange for the pair's other asset.
- Deposit: both assets from the pair.
- Withdrawal: LP tokens.
- Donation: both assets from the pair.
- Strategy: none.
- Record: record NFT.

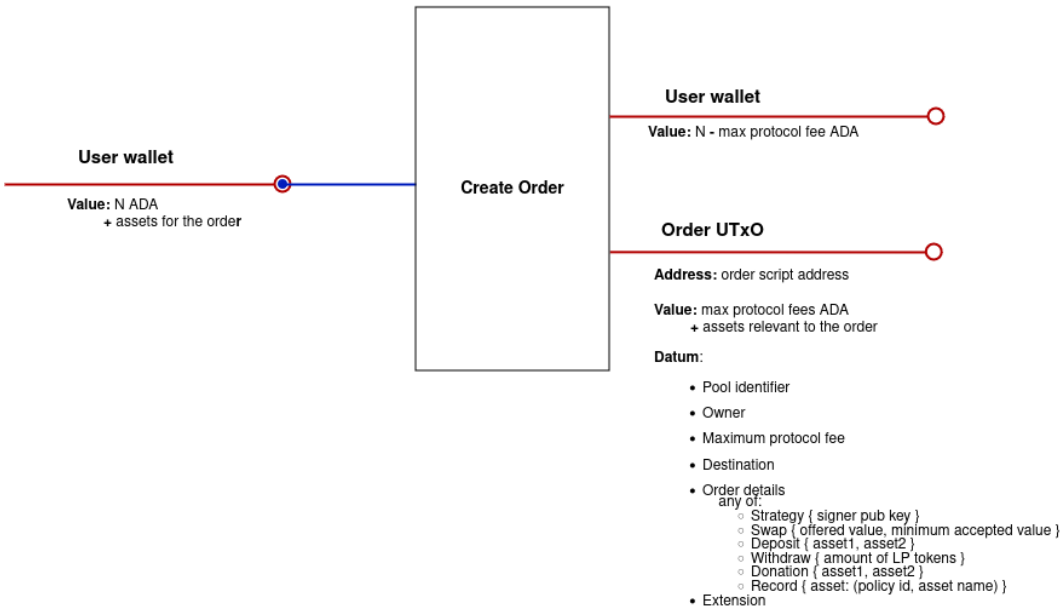

Figure 7: Create Order diagram.

2.b.c.b - Operation "cancel order"

This transaction spends an Order UTxO with a Cancel redeemer that allows the transfer of those funds wherever the order Owner wants, which must sign the transaction.

The most common use cases are recovering the funds and doing an order update by consuming the order UTxO and producing a new one.

It is worth noting that the order Owner is a multi-sig script, which allows a straightforward signature requirement as just the presence of a specific public key signature as well as a complex Cardano native/ simple script-like validation.

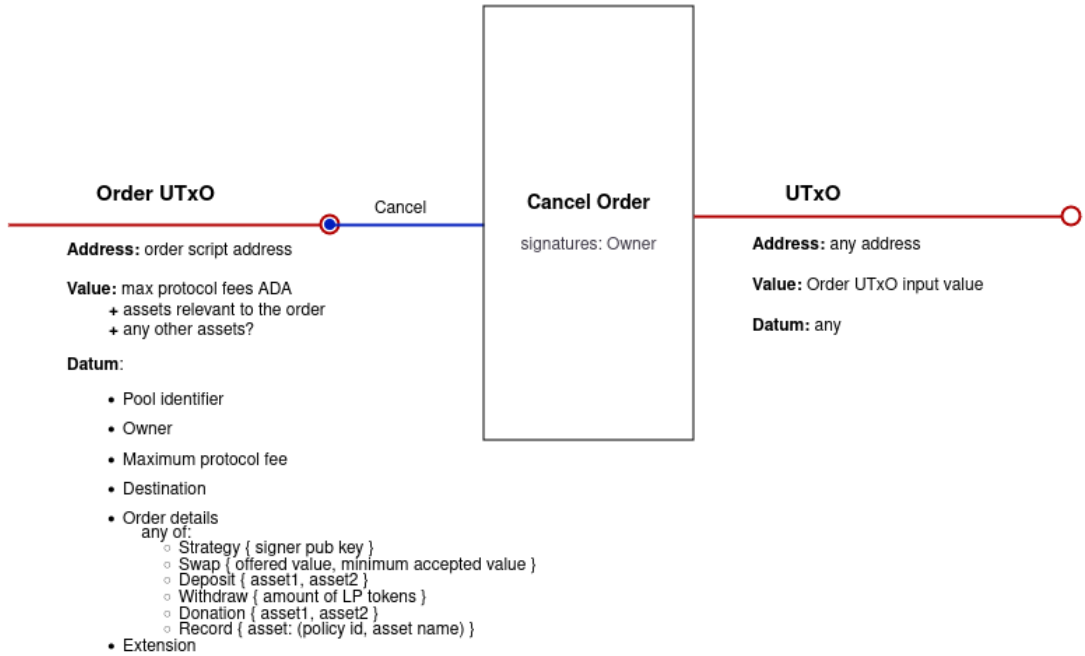

Figure 8: Cancel Order diagram.

Code:

- order.ak:spend():Cancel

Expected Failure Scenarios:
- Owner is **not** signing the transaction

2.b.c.c - Operation “scoop”

This transaction processes a batch of orders against a particular pool, performed by an authorized scooper.

For each order input there is a related destination output that will contain the assets resulting from the processing of such order (plus a remainder in some cases) and any other assets that were in the order and are not related with the pool. The only exception to this rule are Donation orders that have no reminder i.e. the liquidity ratio of the pool is preserved with the exact amounts provided in the donation.

Both the Pool and Order validators are executed. They are attached to the transaction within reference inputs.

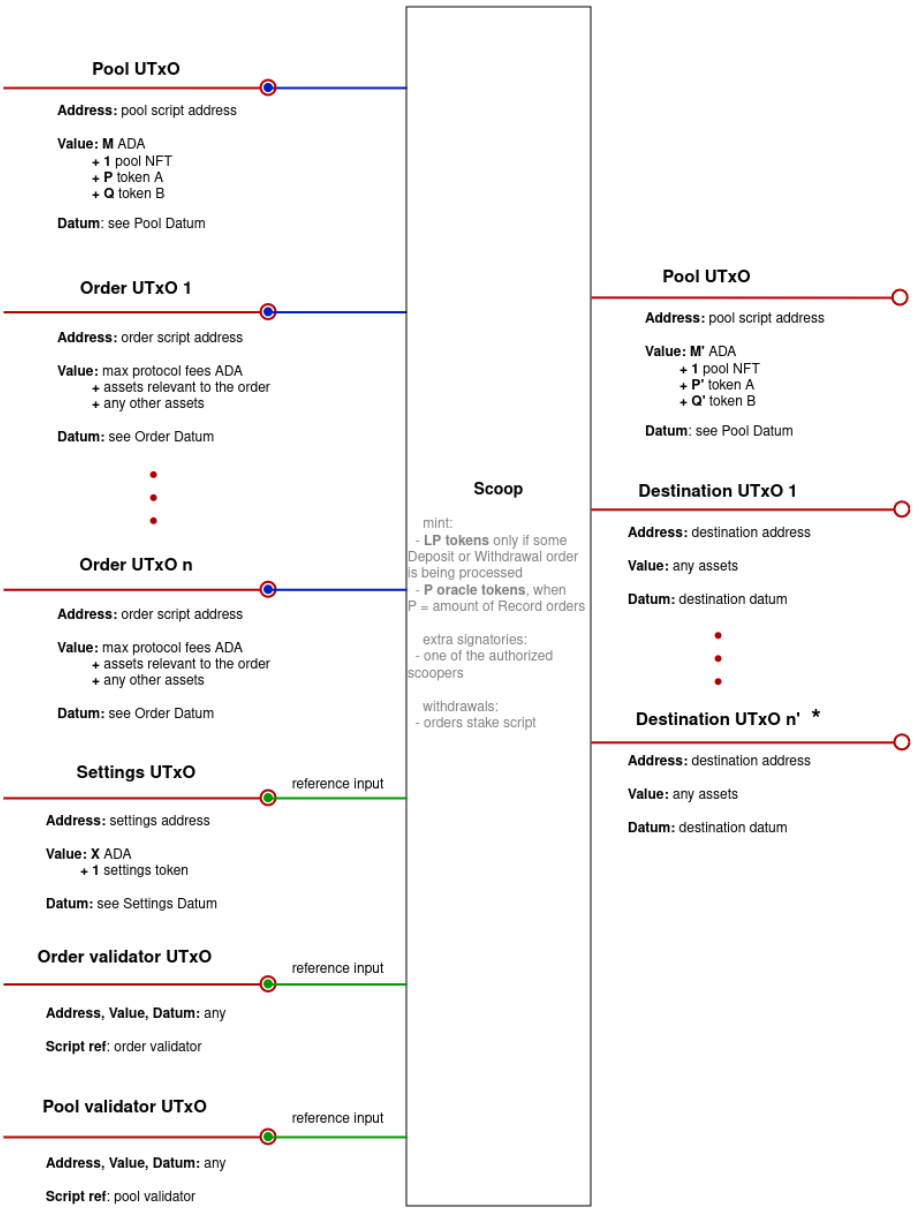

Figure 9: Scoop diagram.

Code:

- pool.ak:spend():PoolScoop

- order.ak:spend():Scoop
- stake.ak:stake():WithdrawFrom
- pool.ak:mint():MintLP
- If there are oracle orders, oracle.ak:mint():Mint

Expected Failure Scenarios:

- Pool output address is not equal than Pool input address
- In the pool datum, other field/s than `circulating_lp` is/are modified
- Pool NFT is stolen from the Pool UTxO or burned
- Pool pair amounts does not match the expected quantities given the specific processed orders
- Pool output has a token that wasn't in the Pool input
- Fees are not correctly paid
- For each destination output. One of:
  - Is not paid to the destination address specified in the corresponding Order input
  - The destination output doesn't have the datum specified in the corresponding Order
  - The paid value is not consistent with the action defined in the corresponding Order
- For each Order input. One of:
  - If the Order has pool identifier, it does not match with the identifier of the Pool being consumed
  - If the Order is of the Strategy type and doesn't have a defined strategy execution, or if it has a strategy execution defined, is not signed by the expected party
  - The assets contained in the Order does not contain the needed assets to perform the requested action over the Pool
- The market is not open yet i.e. the tx validation range is not contained in the interval [market open POSIX time, +∞)
- An incorrect amount of LP tokens are minted/burned if any, or the `circulating_lp` property of the Pool datum is not updated accordingly
- There's no signature of an authorized scooper
- If there's an oracle order:
  - there's a 1-to-1 correspondence with oracle script outputs
  - each oracle script output has only one oracle token
  - oracle datum has the correct validity range and recorded pool values i.e. the token A and B reserves, and circulation LP recorded in oracle datum matches with the Pool output state.

2.b.d - Oracles

2.b.d.a - Operation "create oracle"

Oracle creation is embedded in scoop operation by processing a previously created Record order. So, unlike the other operations described, this one is "contained" within the scoop operation.

From the scoop operation POV: for each Record order input there's an oracle script output uniquely identified by an oracle token (NFT) minted in this same transaction. The oracle datum contains a snapshot of the state of the pool output i.e. quantities of both tokens of the pair and of circulating LP.

2.b.d.b - Operation "close oracle"

This transaction allows to close an oracle on behalf of its owner by enforcing the burning of the oracle token, which is a must since people will be relying on the oracle token to authenticate the actual pool values.

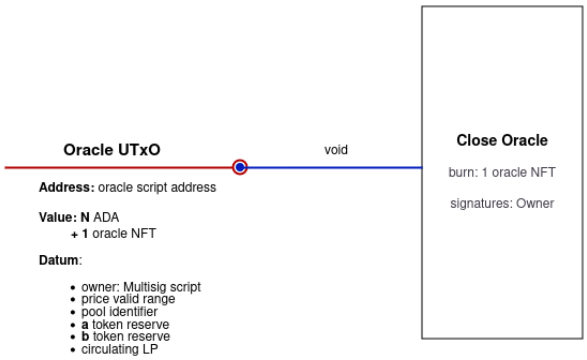

Figure 10: Close Oracle diagram.

Code:

- oracle.ak:spend()
- oracle.ak:mint():Burn

Expected Failure Scenarios:

- Owner doesn't sign the transaction.
- Oracle token is preset in some output i.e. is not being burned.

2.c - Audited Files

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 `edc118880d3baffcb7d5bd277faec2e7dc54c59b`.

Filename
validators/oracle.ak
validators/order.ak
validators/pool.ak
validators/pool_stake.ak
validators/settings.ak
validators/stake.ak
lib/calculation/deposit.ak
lib/calculation/donation.ak
lib/calculation/process.ak
lib/calculation/record.ak
lib/calculation/shared.ak
lib/calculation/strategy.ak
lib/calculation/swap.ak
lib/calculation/withdrawal.ak
lib/shared.ak
lib/types/oracle.ak
lib/types/order.ak
lib/types/pool.ak
lib/types/settings.ak
https://github.com/SundaeSwap-finance/aicone/blob/main/lib/sundae/multisig.ak

3 - Launch details

3.a - Parameters

Name	Value
`protocol_boot_utxo`	`382b27b28c70343161f9abebdab78264e0fd7271baf3bb88ca04b52e5f0067ef#01`

3.b - Script hashes

Validator	Hash (Blake2b-224)
Settings (spend & mint)	`6d9d7acac59a4469ec52bb207106167c5cbfa689008ffa6ee92acc50`
Pool (spend & mint)	`e0302560ced2fdcbfcb2602697df970cd0d6a38f94b32703f51c312b`
Pool staking	`4399813dad91bb78a5eb17c26ff50852bc75d3fa7b6e9ae87232ccc1`
Manage withdrawal	`e0fccbbfb75923bff6dac5f23805dcf6cecfaae8aa3a6d3e474ee670`
Order	`fa6a58bbe2d0ff05534431c8e2f0ef2cbdc1602a8456e4b13c8f3077`
Order withdrawal	`99e5aacf401fed0eb0e2993d72d423947f42342e8f848353d03efe61`
Oracle (spend & mint)	`f50153654bd0e167563cd0bbbff1b73c40157e53408a0ef517e67a5d`

4 - Findings

ID	Title	Severity	Status
SSW-001	Create pool doesn't validate the pool output address	Critical	Resolved
SSW-002	Pool output address is not correctly checked in scoop operation	Critical	Resolved
SSW-101	Settings datum size is limited forever by the initially locked ADA	Major	Resolved
SSW-102	Order `Scoop` redeemer enforces one and only one withdrawal	Major	Resolved
SSW-201	Create pool doesn't validate if ADA is not in the pair	Minor	Resolved
SSW-202	Metadata output datum not checked in pool create	Minor	Resolved
SSW-203	Create pool doesn't validate `fees_per_10_thousand` in pool output datum	Minor	Resolved
SSW-204	No way to modify the list of authorized staking keys in the protocol settings	Minor	Resolved
SSW-205	Pool fees update lacks validation of fees percentages	Minor	Resolved
SSW-206	Pool NFT cannot be burned	Minor	Resolved
SSW-301	Redundant parameters in process_order: outputs = output + rest_outputs	Info	Resolved
SSW-302	Redundant check for pool output stake credential in pool scoop validator	Info	Resolved

SSW-303	Optimizable power of two (`do_2_exp`)	Info	Resolved
SSW-304	Redundant `datum` parameter in `process_order`	Info	Resolved
SSW-305	Total fee computed recursively can be calculated in single expression	Info	Resolved
SSW-306	Optimizable check for initial LP minting in create pool	Info	Resolved
SSW-307	Optimizable check for LP minting in scoop	Info	Resolved
SSW-308	No checks on settings UTxO when it is created	Info	Resolved
SSW-309	Optimizable manipulation of values in `do_donation`	Info	Resolved
SSW-310	Formula simplifications in `do_deposit`	Info	Resolved
SSW-311	Asymmetry of deposit operation	Info	Acknowledged
SSW-312	Optimizable manipulation of output value in `has_expected_pool_value`	Info	Resolved
SSW-313	`UpdatePoolFees` doesn't requires the settings UTxO as reference input	Info	Acknowledged
SSW-314	`PoolState` not used anymore	Info	Acknowledged

5 - SSW-001 Create pool doesn't validate the pool output address

Category	Commit	Severity	Status
Vulnerability	4a5f4f494665f7a110e89d5aa5425fd5cae2311a	Critical	Resolved

5.a - Description

There is no check on the pool output address where pool datum and value are paid to. Without this check, a pool NFT can be minted and paid to any address, even a particular wallet. This token can be used later to scoop orders that are not directed to a specific pool and steal their funds.

5.b - Recommendation

Check that the pool ouput is paid to the pool script hash. This is, that the payment part of the output address equals the own policy ID.

5.c - Resolution

Resolved in commit `d43f212d2a94507bbc7964757093b615c69a8d05` (PR #53).

6 - SSW-002 Pool output address is not correctly checked in scoop operation

Category	Commit	Severity	Status
Vulnerability	00d71b1ff06eac15284c191834926be2d6fe17ed	Critical	Resolved

6.a - Description

The payment part of the output address is not being checked to be correct under the “PoolScoop” redeemer. Without this check, it is possible for a scooper to pay the pool funds and datum to any payment key or script hash, effectively dismanlting the pool and stealing the funds.

The required check was there at some point but it was lost while or after solving SSW-302.

6.b - Recommendation

Check that the pool ouput is paid to the pool script hash. The check can be done a single time at the top-level of the validator so it applies to both redeemers “PoolScoop” and “WithdrawFees”.

6.c - Resolution

Resolved in commit `d43f212d2a94507bbc7964757093b615c69a8d05` (PR #53).

7 - SSW-101 Settings datum size is limited forever by the initially locked ADA

Category	Commit	Severity	Status
Bug	4a5f4f494665f7a110e89d5aa5425fd5cae2311a	Major	Resolved

7.a - Description

Once the protocol settings UTxO is created, it is not possible to change the value locked into it, and in particular the min ADA required by Cardano for storing the UTxO in the blockchain. If an update in the settings requires storing a bigger datum, for instance by adding elements to some of the stored lists, it may be possible that the min ADA required is more than the locked one, making the update impossible.

7.b - Recommendation

The spending validator for the settings UTxO should allow the possiblity of changing the locked value at least for adding more ADA.

7.c - Resolution

Resolved in commit `c65928e0cb00a27a5ac9672d9f9ea0f81a8cc38b` (PR #54).

8 - SSW-102 Order `Scoop` redeemer enforces one and only one withdrawal

Category	Commit	Severity	Status
Bug	4b9fd66acfc2752623d766c95a776263106bdbcd	Major	Resolved

8.a - Description

This `expect` clause of the Order validator enforces that in a Scoop there's one and only one withdrawal.

An issue that may arise because of this is a failure in a scroop transaction when there's a Strategy Order where `StrategyAuthorization` is `Script`. This script must be in the withdrawals, and is not equal than the withdrawal script that the Order validator is expecting.

Another thing to take into account is that the withdrawals are in lexicographical order, so we should be careful when assuming that a certain script is in some specific index of the withdrawals as a list.

8.b - Recommendation

Allow the possiblity to have more than one withdrawal script in a transaction that involves the `Scoop` redeemer of the Order validator.

8.c - Resolution

Resolved in commit `b6fbf3dfa98fa7a0dda65e2d20814dfbf50db365` (PR #74).

9 - SSW-201 Create pool doesn't validate if ADA is not in the pair

Category	Commit	Severity	Status
Bug	4a5f4f494665f7a110e89d5aa5425fd5cae2311a	Minor	Resolved

9.a - Description

Pool output value is checked to have at most 3 different assets (line 415). However, if ADA is not in the pair of assets (A, B), the output value will have four assets: A, B, ADA and the pool NFT. Therefore, the validation fails and it is not possible to create the pool.

9.b - Recommendation

A quick solution is to fix the check so it compares to 3 or 4 depending if ADA is in the pair or not.

Alternatively, we propose to change the approach for validating the output value by building the expected output value and comparing it with value equality. We think this approach is more straightforward and less error prone as it ensures that there is only one possible outcome for the value.

9.c - Resolution

Resolved in commit `ad7183c85af150451dc32a7c3ac091d125f65574` (PR #66).

10 - SSW-202 Metadata output datum not checked in pool create

Category	Commit	Severity	Status
Bug	4a5f4f494665f7a110e89d5aa5425fd5cae2311a	Minor	Resolved

10.a - Description

In pool create, no check is done on the datum that is paid to the metadata output. If the payment is done with no datum and the metadata address set in the settings corresponds to a script, the UTxO will be locked forever and it will not be possible to set the metadata.

10.b - Recommendation

Ensure, at least, that the metadata output has a datum. If the metadata address corresponds a script, this is a necessary condition for the UTxO to be spent.

10.c - Resolution

Resolved in commit `b731c1f5e16cf5be0d39dabae0246eba728ea3ad` (PR #55).

11 - SSW-203 Create pool doesn't validate `fees_per_10_thousand` in pool output datum

Category	Commit	Severity	Status
Robustness	4a5f4f494665f7a110e89d5aa5425fd5cae2311a	Minor	Resolved

11.a - Description

In pool create, no checks are done on the `fees_per_10_thousand` field in the pool output datum. This field is a pair of integers that represent percentages with two decimals. If the integers are not in the range [0, 10000] they will not represent valid percentage values.

11.b - Recommendation

Add the missing checks to ensure that the integers are in the correct range.

11.c - Resolution

Resolved in commit `c290154883e21373ebcc9dcf575d1311388b9429` (PR #56).

12 - SSW-204 No way to modify the list of authorized staking keys in the protocol settings

Category	Commit	Severity	Status
Bug	4a5f4f494665f7a110e89d5aa5425fd5cae2311a	Minor	Resolved

12.a - Description

Once the protocol settings UTxO is created, it is not possible to modify the `authorized_staking_keys` field in the UTxO datum. The spending validator is checking that this field is not changed both in the cases of settings admin and treasury admin updates.

12.b - Recommendation

Depending on business requirements, devise some way that the `authorized_staking_keys` field could be updated.

12.c - Resolution

Resolved in commit `6104be8df1ec9dc81a9e38e84be7d15ea9d6510b` (PR #57).

13 - SSW-205 Pool fees update lacks validation of fees percentages

Category	Commit	Severity	Status
Robustness	4b9fd66acfc2752623d766c95a776263106bdbcd	Minor	Resolved

13.a - Description

In the `CreatePool` redeemer there's a check for `bid_fees_per_10_thousand` and `ask_fees_per_10_thousand` for keeping them within the range [0, 10.000].

In the `UpdatePoolFees` redeemer, those percentages could be updated while this check is not performed. This implies that the fee manager could update them to whatever he wants, defeating the purpose of the check performed in `CreatePool`.

13.b - Recommendation

Repeat said check in `UpdatePoolFees` redeemer.

13.c - Resolution

Resolved in commit `a890bdee4f776d58ef5a022f18121bb77777acb2` (PR #75).

14 - SSW-206 Pool NFT cannot be burned

Category	Commit	Severity	Status
Bug	4b9fd66acfc2752623d766c95a776263106bdbcd	Minor	Resolved

14.a - Description

The Pool minting policy has two redeemers: `CreatePool` and `MintLP`.

- `CreatePool` checks that only one pool NFT is minted.
- `MintLP` ensures that the pool NFT is not minted nor burned.

Then, there's no possiblity of burning the pool NFT, which is required in the `WithdrawFees` redeeemer of the Pool validator whenever the Pool has no more liquidity.

14.b - Recommendation

Add a new redeemer in the Pool minting policy that allows to burn the pool NFT under the expected conditions i.e. whenever the Pool has no more liquidity left.

14.c - Resolution

Resolved in commit `e84cfdfe9b15ab2f85d960d6d840ac0305788d1a` (PR #73).

15 - SSW-301 Redundant parameters in process_order: outputs = output + rest_outputs

Category	Commit	Severity	Status
Redundancy	bcde39aa87567eaee81ccd7fbaf045543c233daa	Info	Resolved

15.a - Description

The `process_order` function takes an `outputs` list but also its head `output` and tail `rest_outputs`. Probably for optimization, as the caller `process_orders` already destructured the list, to avoid repeating it. However, there is no need for the caller to destructure.

15.b - Recommendation

Remove parameters `output` and tail `rest_outputs` from `process_orders`. Destructure inside `process_orders`, and remove destructuring from `process_orders`.

15.c - Resolution

Resolved in commit `5d78f9e2ed10c7206711fc5a58ed0595dbf51c50` (PR #28).

16 - SSW-302 Redundant check for pool output stake credential in pool scoop validator

Category	Commit	Severity	Status
Redundancy	bcde39aa87567eaee81ccd7fbaf045543c233daa	Info	Resolved

16.a - Description

Stake credential is checked in lines 230-231 but entire address was already checked in line 125.

16.b - Recommendation

Remove redundant check.

16.c - Resolution

Resolved in commit `7acd97e69f82e328587abac00ace3d226bddd933` (PR #26).

17 - SSW-303 Optimizable power of two (`do_2_exp`)

Category	Commit	Severity	Status
Optimization	bcde39aa87567eaee81ccd7fbaf045543c233daa	Info	Resolved

17.a - Description

Function `do_2_exp` is used in the pool scoop operation to compute the power of 2 over the set {0, 1, ..., n-1} where n is the number of scooped orders. Current definition is a simple linear recursion, resulting in a relevant impact in mem/cpu consumption.

17.b - Recommendation

Instead of current definition use the optimized `math.pow2` function from Aiken standard library, or the even more optimized version proposed by TxPipe here.

Our tests with the provided benchmark main.ts show that the maximum number of orders can go from 32 to 36.

17.c - Resolution

Resolved in commit `e92bff96934483bf4fe03762e5e2cdef9706eaae` (PR #27).

18 - SSW-304 Redundant `datum` parameter in `process_order`

Category	Commit	Severity	Status
Redundancy	bcde39aa87567eaee81ccd7fbaf045543c233daa	Info	Resolved

18.a - Description

All the information used internally by `process_order` is contained in fields `details` and `destination` that are already parameters of `process_order`. Also: in do_deposit, do_withdrawal and do_donation directly pass details and destination.

18.b - Recommendation

Remove `datum` parameter from `process_order`. In `do_deposit`, `do_withdrawal` and `do_donation`, directly pass `details` and `destination`, instead of whole datum.

18.c - Resolution

Resolved in commit `c143cd30ccebfb8c83940d1fac34475d12a64d80` (PR #29).

19 - SSW-305 Total fee computed recursively can be calculated in single expression

Category	Commit	Severity	Status
Optimization	bcde39aa87567eaee81ccd7fbaf045543c233daa	Info	Resolved

19.a - Description

Total fee is computed in `process_orders` using an accumulator parameter `total_fee` that is returned at the end of the recursion. Individual fee for each order is computed and returned by `process_order` to be accumulated.

However, we observe that the result is equivalent to:

```
total_fee = amortized_base_fee * order_count + simple_fee * simple_count
      + strategy_fee * strategy_count
```

19.b - Recommendation

Remove `total_fee` as parameter and return value from `process_orders`. In `pool.ak`, directly define `total_protocol_fee` using the given formula.

Also, there is no need for `process_order` to return the fee anymore.

19.c - Resolution

Resolved in commit `e686590f18dce0ef50074296cdc502f2adb9fea0` (PR #30).

20 - SSW-306 Optimizable check for initial LP minting in create pool

Category	Commit	Severity	Status
Optimization	4a5f4f494665f7a110e89d5aa5425fd5cae2311a	Info	Resolved

20.a - Description

In pool create, Aiken's `math.sqrt` is used to check for the correct initial minting of LP tokens. This function implements the recursive Babylonian method. However, the expected value for the sqrt is already known, as it is available in the minting field and in the datum. Having the expected value, it is much more efficient to check that it is correct by squaring it and comparing to the radicand:

```
/// Checks if an integer has a given integer square root x.
/// The check has constant time complexity (O(1)).
pub fn is_sqrt(self: Int, x: Int) -> Bool {
  x * x <= self && ( x + 1 ) * ( x + 1 ) > self
}
```

See this PR for more information.

20.b - Recommendation

First, define `initial_lq` by taking it from the minting field or from the datum (`circulating_lp`). Then, check that it is correct with `is_sqrt(coin_a_amt_sans_protocol_fees * coin_b_amt, initial_lq)`.

20.c - Resolution

Resolved in commit `12a55dab0bdddd8d8eba6c1d98ef75c3b5f95747` (PR #58).

21 - SSW-307 Optimizable check for LP minting in scoop

Category	Commit	Severity	Status
Optimization	fd3a48511eea723fe58d32e79993c86c26df0a94	Info	Resolved

21.a - Description

We understand that the idea in the current version of MintLP redeemer validation is to ensure that the UTxO that contains the pool NFT i.e. the pool UTxO must be in the inputs because in its spending validator the minting of LP tokens is controlled.

The issue with this check is that it is O(n): the worst case being when the pool UTxO is in the tail of the inputs and there are +20 orders. Given that this redeemer runs during a scoop validation, it is interesting to optimize it as much as possible.

21.b - Recommendation

Instead of checking the presence of the pool NFT in the inputs, we can check its presence in the outputs given that we can safely asume that the pool UTxO is the first output. This is O(1). But this is not sufficient: we must also check that the pool NFT is not being minted in this same transaction. These two checks ensure that the pool NFT is in the inputs.

21.c - Resolution

Resolved in commit `db3d33e3a22a28a7c1e7abfcb798e00e68427ff6` (PR #59).

22 - SSW-308 No checks on settings UTxO when it is created

Category	Commit	Severity	Status
Robustness	4a5f4f494665f7a110e89d5aa5425fd5cae2311a	Info	Resolved

22.a - Description

The settings UTxO creation is validated through the minting policy of the settings NFT token. This policy only checks for the minting itself, ensuring that the token name is correct and that the indicated UTxO is consumed (this way enforcing an NFT). There is no validation of where is this NFT being paid to, or anything related to the creation of the settings UTxO.

However, the policy could also be checking for the creation of the settings UTxO, including checks for the correct address, value and datum. This pattern is known as the "base case" for the "inductive reasoning" that can guarantee the consistency of the contract state through its entire lifetime.

Reference: Edsko de Vries, Finley McIlwaine. *Verifying initial conditions in Plutus*.

22.b - Recommendation

In the settings NFT minting policy, add checks to ensure that the settings UTxO is correctly created. This is, check the payment to the correct address, datum and value.

22.c - Resolution

Resolved in commit `f007b795e9e49a38e0b3f00355d4f97ce1e27c3a` (PR #60).

23 - SSW-309 Optimizable manipulation of values in `do_donation`

Category	Commit	Severity	Status
Optimization	2487900eea2ea1d87f6e8a04707dcf039becd265	Info	Resolved

23.a - Description

The processing of donation orders could be optimized in its manipulation of values. In particular, when computing the `remainder` variable the `value.merge` and `value.negate` functions are used (see https://github.com/SundaeSwap-finance/sundae-contracts/blob/2487900eea2ea1d87f6e8a04707dcf039becd265/lib/calculation/donation.ak#L39-L44), which are not particularly efficient.

23.b - Recommendation

Instead of using `value.merge` and `value.negate` it's possible to use `value.add` (a lot more efficient) to achieve the same effect.

For example:

```
let remainder =
  input_value
    |> value.add(ada_policy_id, ada_asset_name, -actual_protocol_fee)
    |> value.add(assets.1st.1st, assets.1st.2nd, -assets.1st.3rd)
    |> value.add(assets.2nd.1st, assets.2nd.2nd, -assets.2nd.3rd)
```

gives us better mem and cpu numbers in the 30 shuffled orders processing test.

The original implementation gives

```
PASS [mem: 12571807, cpu: 4883611979] process_30_shuffled_orders_test
```

while the version with just `value.add`'s

```
PASS [mem: 10580319, cpu: 4281305949] process_30_shuffled_orders_test
```

(numbers obtained with aiken version v1.0.21-alpha+4b04517)

23.c - Resolution

Resolved in commit `30f4d17cacc3fa9d8bc7a6d85ecae4eb4772e8a4` (PR #61).

24 - SSW-310 Formula simplifications in `do_deposit`

Category	Commit	Severity	Status
Simplification	2487900eea2ea1d87f6e8a04707dcf039becd265	Info	Resolved

24.a - Description

To compute the final amounts to be deposited, change is calculated in two different ways depending on wich of the assets is the one that has a change. However, it is possible to skip the change definition and have simpler formulas for the final amounts.

24.b - Recommendation

TODO: property-based checks for equivalence between new formulas and old ones.

For the first case, where there is change in asset B, the deposited B amount can be directly computed as:

```
pool_state.quantity_b.3rd * user_gives_a / pool_state.quantity_a.3rd
```

To preserve the exact same rounding behavior as the original code, ceiling division should be used:

```
(pool_state.quantity_b.3rd * user_gives_a - 1) / pool_state.quantity_a.3rd + 1
```

For the second case, where there is change in asset A, the deposited A amount can be directly defined as `b_in_units_of_a`.

Proof: (Rounding details are left out of the proof.)

First, `change` definition can be simplified as follows:

```
change
=  // definition of change
quantity_b * (b_in_units_of_a - user_gives_a) / quantity_a
=  // definition of b_in_units_of_a
quantity_b * (user_gives_b * quantity_a / quantity_b - user_gives_a) / quantity_a
= // distributive
quantity_b * user_gives_b * quantity_a / quantity_b / quantity_a
- quantity_b * user_gives_a / quantity_a
= // cancel out quantity_a and quantity_b
user_gives_b - quantity_b * user_gives_a / quantity_a
```

Then, deposited B amount is:

```
user_gives_b - change
= // simplified version of change
user_gives_b -  (user_gives_b - quantity_b * user_gives_a / quantity_a)
= // math
quantity_b * user_gives_a / quantity_a
```

On the other hand, deposited A amount is

```
user_gives_a - change
= // definition of change
user_gives_a - (user_gives_a - b_in_units_of_a)
= // math
b_in_units_of_a
```

24.c - Resolution

Resolved in commit `db5185ca01e3ffdb4c643a651f4b439ddd3d0ae4` (PR #76).

25 - SSW-311 Asymmetry of deposit operation

Category	Commit	Severity	Status
Theoretical	2487900eea2ea1d87f6e8a04707dcf039becd265	Info	Acknowledged

25.a - Description

In abstract, the AMM model is defined over an unordered pair of assets {A, B}, so all operations are symmetric in term of the roles of A and B. In Sundae's implementation, the deposit operation is asymmetric at least for some corner cases, as illustrated in the tests provided in this branch.

This issue is related to the way that function `do_deposit` is implemented and the rounding issues that arise when using integer arithmetics. The implementation can be easily modified to achieve symmetry.

Our understanding is that this finding is not exploitable beyond rounding errors. However, symmetry may be a desirable property as it puts the implementation much closer to the theoretical model.

25.b - Recommendation

If the symmetry property is desired, modify `do_deposit` in a way that it is guaranteed by the code.

25.c - Resolution

Project team decided not to resolve this finding. Current business model and implementation, including the formulas for the deposit operation, are inherited from SundaeSwap V2 protocol, and were extensively tested in previous audits. As the **audit team** we endorse the decision, as this finding is only informational.

26 - SSW-312 Optimizable manipulation of output value in `has_expected_pool_value`

Category	Commit	Severity	Status
Optimization	2487900eea2ea1d87f6e8a04707dcf039becd265	Info	Resolved

26.a - Description

As stated in a code comment here, each `value.quantity_of`, `value.lovelace_of`, and also `has_exact_token_count` calls traverse the output value. This could be optimized by doing just one traversal of the entire value.

26.b - Recommendation

Instead of using `value.quantity_of`, `value.lovelace_of`, and `has_exact_token_count`, we can traverse the output value just once by converting it into a list and then using fold or a recursive function to perform all the needed checks throughout.

26.c - Resolution

Resolved in commit `bfe8e8fd9f1177b6b202c6e871a8ed6e65d217e9` (PR #52).

27 - SSW-313 `UpdatePoolFees` doesn't requires the settings UTxO as reference input

Category	Commit	Severity	Status
Redundancy	da66d15afa9897e6bdb531f9415ddb6c66f19ce4	Info	Acknowledged

27.a - Description

The settings UTxO is required as reference input by contract in both spend Pool redeemers `PoolScoop` and `Manage`. This in particular means that in both `WithdrawFees` and `UpdatePoolFees` redeemers of the manage stake script is required as well. Furthermore, is explicitly looked up in there. Even though it's needed by `WithdrawFees` logic, for `UpdatePoolFees` logic it is not.

27.b - Recommendation

Move the find_settings_datum call from the Pool spend inside the `PoolScoop` branch, and in the manage stake script move such function call inside the `WithdrawFees` branch.

27.c - Resolution

Project team decided not to resolve this finding in the scope of the audited version. As the **audit team** we endorse the decision, since the downside is just the little extra off-chain work of adding the settings UTxO as reference input when building the update pool fees transaction.

28 - SSW-314 `PoolState` not used anymore

Category	Commit	Severity	Status
Redundancy	da66d15afa9897e6bdb531f9415ddb6c66f19ce4	Info	Acknowledged

28.a - Description

Given the merged PR #78 `refactor to use continuations`, the `PoolState` type previously used is not needed anymore by the Pool validator.

28.b - Recommendation

Remove `PoolState` type definition.

28.c - Resolution

Project team decided not to resolve this finding in the scope of the audited version. As the **audit team** we endorse the decision, as this doesn't impact the contracts since unused structures are not included in the compilation result.

29 - Minor issues

In this section we list some issues we found that do not qualify as findings such as typos, coding style, naming, etc. We used the Github issues system to report them.

- typo: continuout -> continuing #37
- settings validator: use function to get spent output #38
- unnecessary tuple definitions #39
- compare_asset_class: camelCase to snake_case #41
- testing code: replace “escrow” with “order” in definitions #44
- pool validator: unused import TransactionId #45
- settings type definition: unused import Output #46

30 - Contributed PRs

In this section we list some code contributions we did, usually as a result of studying and/or confirming possible findings.

- New Aiken test for process_orders with 30 shuffled donation orders #48
- has_expected_pool_value traverses output value just once #52
- Optimization for count_orders #64

31 - Appendix

31.a - Disclaimer

This report is governed by the terms in the agreement between TxPipe (**TXPIPE**) and Sundae Labs (**CLIENT**). This report cannot be shared, referred to, altered, or relied upon by any third party without TXPIP’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.**

31.b - Issue Guide

31.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.

31.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.

31.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.

31.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.

31.d.a - Links

- Website
- Email
- Twitter