ANASTASIA LABS

Security Audit Report

Date: June 21, 2024
Project: Minswap AMM V2
Version 1.0

Contents

Disclosure

Disclaimer and Scope

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

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

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

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

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

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

Assessment overview

From March 20th, 2024 to April 19th, 2024, Minswap engaged Anastasia Labs to evaluate and conduct a security assessment of its Minswap AMM V2 protocol. All code revision was performed following industry best practices.

Phases of code auditing activities include the following:

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

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

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

Assessment components

Manual revision

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

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

Executive summary

Minswap is a Decentralized Exchange (DEX). The purpose of a DEX is to enable permissionless trading of token pairs. For each swap, a fee is taken, which goes to the Liquidity Providers (LPs). Anyone can provide Liquidity as well, hence profits are decentralized. Minswap is a community-centric DEX, in that $MIN tokens are fairly distributed, without any private or VC investment. Minswap has pioneered several ideas in the Cardano ecosystem such as the FISO model (touted as the fairest ISO model in the Cardano community) or a focus on Protocol Owned Liquidity.

Features Community First: MIN tokens are distributed fairly to protocol participants and Liquidity Providers, who can participate in governance and vote democratically on protocol changes.

Innovation-Driven: We pioneered ideas such as the FISO model, the MINt token, or the Liquidity Bootstrapping Event and plan to continue doing so with further novel initiatives.

Launchpool: Minswap is permissionless, meaning anybody can list tokens without needing KYC. For promising Cardano projects that want to use some of the most potent DeFi primitives to bootstrap their liquidity, we have designed the Minswap Launch Bowl.

Stake Pool Operators Support: Minswap supported SPOs through the FISO and plans to continue doing so with a community-oriented ADA delegation policy, incentivizing SPOs in our batching solution Laminar, and by enabling automatic native token fees conversion (Babel Fees).

Code base

Repository

https://github.com/minswap/minswap-dex-v2

Commit

837dbc4cc870c409c87fc80c1e4907fb082b4921

Files audited

SHA256 Checksum	Files
961aca33ae0017d1a84a7ac6179b0d5c 2653c5361105f294c80a012f2a7f2e4b	lib/amm_dex_v2/math.ak
cce3d7b65188ebbb07ae0d9885339a4f 448da91744a6982a5b66fb646349e9c5	lib/amm_dex_v2/order_validation.ak
1a159fd056ae768fd50ec6c8ab3bad171 72d534b6417582e5bc310b3f8f880cc	lib/amm_dex_v2/pool_validation.ak
31927b82f4b2baff5f5b466b9e2a02a86 66c02543f90715d2cefb002bfae2add	validators/authen_minting_policy.ak
6c133d131463330abeb077bef8cb9a5d eb90036a644f54ce85a0041c1a7a7d74	validators/factory_validator.ak
6b37f7bac8f3b5bb153e2f25b86c03c81 4778020075505f3b74cb36024e8751f	validators/order_validator.ak
96fdf5874c1d66a61f1e7ddcac526cd9b 6e1ad02d6bcfb0fc918cd0d7d54f400	validators/pool_validator.ak

Severity Classification

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

Finding severity ratings

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

	Level	Severity	Findings
🟥	5	Critical	0
🟥	4	Major	0
🟧	3	Medium	1
🟨	2	Minor	5
🟦	1	Informational	7

Findings

ID-301 Swap multi routing

	Level	Severity	Status
🟨	3	Medium	Resolved

Description

The Swap Multi Routing feature enables users to route orders between assets across different liquidity pools, allowing for routing sequences such as `Asset(a) -> Asset(b)`, `Asset(b) -> Asset(c)`, or even more complex combinations like `Asset(a) -> Asset(b)`, `Asset(b) -> Asset(c)`, `Asset(c) -> Asset(d)`, and so forth.

However, a significant issue arises when such orders require extensive routing, as it compels the batcher to expend a substantial portion of liquidity pool funds simultaneously. This scenario poses a potential threat as it renders the liquidity pool's UTXO temporarily unusable, opening avenues for concurrency attacks.

Recommendation

To mitigate the risk of concurrency attacks and ensure the stability of the liquidity pool, it is advisable to impose a reasonable cap on the amount of routing permitted within a single transaction.

Resolution

Resolved in commit 6239e7e6fd7e8db98e8334d29e798a3a0e348f10.

ID-201 Unbounded time range

	Level	Severity	Status
🟨	2	Minor	Resolved

Description

The `expired_time` variable is utilized to validate whether an order exceeds the expiration time set at the order datum. Subsequently, this `expired_time` is compared against `end_valid_time_range`, which is derived from the upper-bound validity range.

Listing 1: validators/pool_validator.ak

```
let end_valid_time_range =
  utils.must_get_finite_end_validity(validity_range)
```

The issue arises when only the upper bound of the validity ranges provided by the transaction is considered.

This becomes problematic if the upper validity bound is set to an excessively high number. In such cases, the transaction can be rendered invalid because the `end_valid_time_range` will be greater than the `expired_time`.

Listing 2: validators/pool_validator.ak

```
when expiry_setting_opt is {
  None -> True
  Some((expired_time, _)) -> end_valid_time_range <= expired_time
}
```

Recommendation

It is recommended to employ a more accurate time approximation method. This can be achieved by obtaining both the upper and lower bounds from the transaction and ensuring that the time range between them does not exceed a reasonable duration, such as a 10-minute time range.

- Calculate time approximation

```
let curren_time_approximation =
  upper_bound - lower_bound / 2 + lower_bound
```

- Check time range is within a reasonable limit

```
let is_valid = upper_bound - lower_bound <= 10 * 60 * 1000
```

This can prevent the exploitation of large upper bounds to invalidate transactions.

Resolution

Resolved in commit 6239e7e6fd7e8db98e8334d29e798a3a0e348f10.

ID-202 Centralized Batcher

	Level	Severity	Status
🟨	2	Minor	Acknowledged

Description

`order_inputs` are sorted lexicographically by the ledger, and consequently, to process orders accurately, the batcher must sort orders based on creation time. However, allowing the batcher to handle this sorting introduces a vulnerability wherein the batcher could engage in front running by prioritizing certain orders, unfairly benefiting itself.

The below code snippet demonstrates the passing of `input_indexes` via the redeemer, a parameter set by the batcher itself, granting it full control over the sorting outcome.

Listing 3: validators/pool_validator.ak

```
let sorted_order_inputs =
  bytearray.foldr(
    input_indexes,
    [],
    fn(idx, ips) { list.push(ips, utils.list_at_index(order_inputs, idx)) },
  )
```

Recommendation

Should any indication of front running by the batcher be observed, immediate action must be taken. This includes revoking the address license associated with the batcher. Additionally, consider implementing measures to decentralize the sorting process.

Resolution

Acknowledged

ID-203 Centralized Admin

	Level	Severity	Status
🟨	2	Minor	Acknowledged

Description

The current setup grants the admin full control over global settings, potentially leading to misuse. The admin holds the power to remove/change any of the listed entities, thus presenting a risk of centralization.

```
// This setting grants permissions to authorized actors who can interact
   with Liquidity Pool features.
pub type GlobalSetting {
  // List of authorized batchers who can process orders.
  batchers: List<Address>,
  // The actor who can update the Pool's base fee and fee sharing.
  pool_fee_updater: Address,
  // The actor who can withdraw the Pool's fee sharing.
  fee_sharing_taker: Address,
  // The actor who can change the Pool's stake key.
  pool_stake_key_updater: Address,
  // The actor who can update the Pool's dynamic fee.
  pool_dynamic_fee_updater: Address,
  // The actor who can update the addresses mentioned above.
  // This admin can be transferred to another wallet and should be stored in
     the most secure location.
  admin: Address,
}
```

The `GlobalSetting` value is read from a reference input UTxO, that can only be updated by providing a witness associated with the admin address. The admin address can be defined as one of the following:

- **Public key address**: updating transaction must be signed with the associated private key.
- **Validator script**: in this case an UTxO must be spent from the address (the validation script must approve the spending).
- **Staking script**: the transacion is also accepted, if any staking operation is approved by the script.

The latter two options are not mutually exclusive, since a UPLC function can behave both as a validator and a staking script depending on its parameters.

Recommendation

We recommend implementing an upgradable multisig contract where a consensus of m out of n signatures is required to approve changes to the global settings. This multisig setup not only ensures greater security, but it can also allows for the addition or removal of members, mitigating the risk associated with the loss of a single private key.

Resolution

Mitigated by the MinSwap team implementing and using a multisig wallet as per the recommendation.

ID-204 Staking control

	Level	Severity	Status
🟨	2	Minor	Resolved

Description

During pool creation, the stake credential of the newly created pool UTxO are not verified by the `factory_validator` or the `authen_minting_policy` validators. This oversight allows the pool creator to set an arbitrary staking credential, potentially granting undue influence over staking (delegation/voting) activities.

The following code snippet illustrates the absence of staking credential verification:

Listing 4: validators/factory_validator.ak

```
expect [pool_output] =
  list.filter(
    outputs,
    fn(output) {
      let Output { address: out_addr, value: out_value, .. } = output
      let Address { payment_credential: out_addr_payment_credential, .. } =
        out_addr
      when out_addr_payment_credential is {
        ScriptCredential(hash) -> and {
            pool_hash == hash,
            value.quantity_of(
              out_value,
              authen_policy_id,
              utils.pool_auth_asset_name,
            ) == 1,
          }
        _ -> False
      }
    },
  )
```

However, after pool creation, this issue can be mitigated by a special user called `pool_stake_key_updater`, designated by the `GlobalSetting` datum, allowing for the updating of the staking credential for the given pool.

Recommendation

To address this, it is recommended to use the pool address as a parameter in `factory_validator` instead of `pool_hash: ValidatorHash`, ensuring certainty of the staking credential when creating the pool UTxO

Additionally, it's important to inform liquidity providers that when they lock their ADA, staking control is given to the pools, enabling them to exercise delegation/voting rights.

Resolution

Resolved in commit 7362272349f0eab1360e9732f1a6b44207484bbb.

ID-205 Min Ada

	Level	Severity	Status
🟨	2	Minor	Resolved

Description

The Cardano ledger imposes constraints to prevent excessive growth, enforcing all UTXO entries to contain a minimum ada value.

This value is derived from a formula based on the Ledger protocol parameter known as `coinsPerUTxOByte`.

However, the issue arises when this minimum ada value is hardcoded, preventing contracts from adapting to potential Ledger upgrades of the `coinsPerUTxOByte` parameter.

The following code snippet illustrates the hardcoded min ada value.

Listing 5: validators/factory_validator.ak

```
let expected_pool_out_value =
  value.zero()
    |> value.add(ada_policy_id, ada_asset_name, 3000000)
    |> value.add(asset_a_policy_id, asset_a_asset_name, amount_a)
    |> value.add(asset_b_policy_id, asset_b_asset_name, amount_b)
    |> value.add(authen_policy_id, lp_asset_name, remaining_liquidity)
    |> value.add(authen_policy_id, utils.pool_auth_asset_name, 1)
```

Recommendation

We recommend setting the minimum ada value, currently '3000000', to a bounded and upgradable parameter managed by the pool admin. This value could be defined within a range, enabling the pool contract to accommodate changes to the Ledger's 'coinsPerUTxOByte' parameter.

Resolution

Resolved in commit 6239e7e6fd7e8db98e8334d29e798a3a0e348f10.

ID-101 Batcher index

	Level	Severity	Status
🟦	1	Informational	Resolved

Description

The `batcher_address` expression incurs unnecessary computation, due to the coercion of the `batcher_index` from type `ByteArray` to `Int` using the `builtin.index_bytearray` function. This coercion is performed to satisfy the requirements of the `list_at_index` function.

Listing 6: lib/amm_dex_v2/types.ak

```
pub type PoolBatchingRedeemer {
  batcher_index: ByteArray,
  orders_fee: List<Int>,
  input_indexes: List<Int>,
  pool_input_indexes_opt: Option<List<Int>>,
  vol_fees: List<Option<Int>>,
}
```

Listing 7: validators/pool_validator.ak

```
let batcher_address =
  utils.list_at_index(batchers, builtin.index_bytearray(batcher_index, 0))
```

Recommendation

It is advised to update the type of `batcher_index` to `Int`. This adjustment will remove the need for coercion and improve efficiency.

Resolution

Resolved in commit 6239e7e6fd7e8db98e8334d29e798a3a0e348f10.

ID-102 Order input name

	Level	Severity	Status
🟦	1	Informational	Resolved

Description

The expressions `user_inputs` and `sorted_user_inputs` may cause confusion when referencing to order inputs within the codebase.

Recommendation

To improve clarity, our recommendation to rename `user_inputs` to `order_inputs` and `sorted_user_inputs` to `sorted_order_inputs`

Resolution

Resolved in commit 6239e7e6fd7e8db98e8334d29e798a3a0e348f10.

ID-103 Redundant calculation

	Level	Severity	Status
🟦	1	Informational	Resolved

Description

When the user deposits assets to a pool in a different ratio to what the pool has, the function `calculate_deposit_amount` is used to calculate the released amount of liquidity tokens. The calculation uses the following steps:

1. Calculate $swap_x$ based on the formula in the documentation.
2. Calculate $receive_y$ with the knowledge of $swap_x$.
3. Calculate ΔL with the equation $\Delta L = \frac{\Delta y + receive_y}{y_0 - receive_y} * L$.

Since the formula that allows the analytical calculation of $swap_x$ is based on the equation:

$$\frac{\Delta x - swap_x}{x_0 + swap_x} = \frac{\Delta y + receive_y}{y_0 - receive_y}$$

calculating $receive_y$ is redundant.

Recommendation

To reduce the cost of the validator function, the calculation could be replace with the following steps:

1. Calculate $swap_x$ based on the formula in the documentation.
2. Calculate ΔL with the equation $\Delta L = \frac{\Delta x - swap_x}{x_0 + swap_x} * L$.

Resolution

Resolved in commit 6239e7e6fd7e8db98e8334d29e798a3a0e348f10.

ID-104 Calculate_deposit_swap_amount unit tests

	Level	Severity	Status
🟦	1	Informational	Acknowledged

Description

The function `calculate_deposit_swap_amount` lacks unit testing or property-based testing to validate the following equation:

$$(1-f)*(y_0+\Delta y)*swap_x^2+(2-f)*(y_0+\Delta y)*x_0*swap_x+(x_0^2*\Delta y-x_0*y_0*\Delta x)=0$$

Recommendation

We strongly recommend implementing unit tests or property-based tests to validate the correctness of `calculate_deposit_swap_amount` function

Example Test Case

Below is an example implementation of a unit test. However, it's essential to note that this is just one case, and further testing should cover a broader range of scenarios.

Additionally, consideration should be given to potential errors introduced by decimal precision in the inputs.

```
test swap_amount_test() {
  let amount_a = 10
  let amount_b = 6
  let reserve_a = 100
  let reserve_b = 100
  let trading_fee_a_numerator = 500
  let (n, d) =
    calculate_deposit_swap_amount(
      amount_in: amount_a,
      amount_out: amount_b,
      reserve_in: reserve_a,
      reserve_out: reserve_b,
      trading_fee_numerator: trading_fee_a_numerator,
    )
  expect Some(one_minus_f) =
    rational.new(10000 - trading_fee_a_numerator, 10000)
  expect Some(two_minus_f) =
    rational.new(20000 - trading_fee_a_numerator, 10000)
  expect Some(swap) = rational.new(n, d)
  let a =
    rational.mul(swap, swap)
      |> rational.mul(rational.from_int(amount_b + reserve_b))
      |> rational.mul(one_minus_f)
  let b =
    rational.from_int(reserve_a * ( amount_b + reserve_b ))
      |> rational.mul(swap)
      |> rational.mul(two_minus_f)

  let c =
    rational.from_int(
      calculate_pow(reserve_a) * amount_b - reserve_a * reserve_b * amount_a
    )

  let z =
    rational.add(a, b)
      |> rational.add(c)
      |> rational.truncate()
  z == 0
}
```

Resolution

Acknowledged

ID-105 Calculate_amount_in property

	Level	Severity	Status
🟦	1	Informational	Acknowledged

Description

The function `calculate_amount_in` lack unit test to validate the following property

$$x_0 * y_0 = k_0 \leq k_1 = x_1 * y_1$$

This property should hold for all swaps as a guarantee for liquidity providers that the assets held in a pool cannot decrease over time with swap operations.

Since the release of the fuzz Aiken library, this property can be expressed as a test:

```
pub type SwapInput {
  reserve_in: Int,
  reserve_out: Int,
  amount_out: Int,
  trading_fee_numerator: Int,
}

fn swap_input() -> Fuzzer<SwapInput> {
  let reserve_in <- and_then(int_at_least(1))
  let amount_out <- and_then(int_at_least(1))
  let reserve_out <- and_then(int_at_least(amount_out + 1))
  let trading_fee_numerator <- map(int_at_least(0))
  SwapInput { reserve_in, reserve_out, amount_out, trading_fee_numerator }
}

test prop_k_does_not_decrease_on_swap(swap_input via swap_input()) {
  let amount_in =
    calculate_amount_in(
      reserve_in: swap_input.reserve_in,
      reserve_out: swap_input.reserve_out,
      amount_out: swap_input.amount_out,
      trading_fee_numerator: swap_input.trading_fee_numerator,
    )

  let x0 = swap_input.reserve_in
  let y0 = swap_input.reserve_out
  let x1 = x0 + amount_in
  let y1 = y0 - swap_input.amount_out

  let k0 = x0 * y0
  let k1 = x1 * y1
  k0 <= k1
}
```

Recommendation

We suggest adding property based testing to validate the aforementioned property.

Resolution

Acknowledged

ID-106 Output datum type

	Level	Severity	Status
🟦	1	Informational	Acknowledged

Description

The function below validates the output datum type follows the datum set at the order input.

```
fn is_valid_datum(raw_datum: Datum, extra_order_datum: ExtraOrderDatum) ->
    Bool {
  let expect_extra_order_datum =
    when raw_datum is {
      NoDatum -> EODNoDatum
      DatumHash(dh) -> EODDatumHash(dh)
      InlineDatum(dat) ->
        EODInlineDatum(hash.blake2b_256(builtin.serialise_data(dat)))
    }
  extra_order_datum == expect_extra_order_datum
}
```

However, a critical issue arises when users set the `refund_receiver_datum` and/or `success_receiver_datum` to inline datum while the receiver is of a Plutus V1 address. This could potentially result in funds being permanently locked

This occurs because the batcher must set the output order datum following `refund_receiver_datum` and `success_receiver_datum` from `OrderDatum`.

```
pub type OrderDatum {
  // The address's payment credential that can cancel the order, can by
      PubKey or Script
  canceller: OrderAuthorizationMethod,
  // The address of the output after being killed by Batcher or cancelled by
       bots (order is expired)
  refund_receiver: Address,
  // The datum hash of the output after being killed by Batcher or cancelled
       by bots (order is expired)
  refund_receiver_datum: ExtraOrderDatum,
  // The address which receives the funds after order is processed
  success_receiver: Address,
  // The datum hash of the output after order is processed.
  success_receiver_datum: ExtraOrderDatum,
  // The Liquidity Pool's LP Asset that the order will be applied to
  lp_asset: Asset,
  // The information about Order Type
  step: OrderStep,
  // The maximum fee users have to pay to Batcher to execute batching
      transaction
  // The actual fee Batcher will take might be less than the maximum fee
  max_batcher_fee: Int,
  // expiry setting option contain
  // - Order Expired time: If the order is not executed after Expired Time,
     anyone can help the owner cancel it
  // - Max tip for cancelling expired order
  expiry_setting_opt: Option<(Int, Int)>,
}
```

Recommendation

It is recommended to either verify or warn the user if the receiver address is of Plutus V1 type. Then, ensure that the ExtraOrderDatum is of type datum hash to prevent potential fund lockup.

Resolution

The team has acknowledged this situation, and Minswap Labs has the responsibility to provide and communicate sufficient information to users to prevent potential fund lock-ups.

ID-107 Missing batcher fee check

	Level	Severity	Status
🟦	1	Informational	Resolved

Description

The donation order validator function `validate_donation` checks that the batcher fee is included in the order, if one of the assets (`asset_a`) is Ada. In other cases (where neither of the assets are Ada) this check is omitted.

Recommendation

While this missing check can lead to the batcher not receiving the batcher fee, it is unlikely that a batcher would include this order in a transaction, since the incentive to do so is missing. However, we recommend checking it so that the inclusion of the batcher fee is consistently verified for every case.

Resolution

Resolved in commit 2fa7b1653a9d6dd9e0b9a3500f88529b2f6e511a.

Post Audit Findings

The following two findings were discovered as a part of the bug bounty conducted by Minswap team. Fixes for these vulnerabilities were reviewed by us.

ID-01 Use of Spend validator as Stake validator

	Level	Severity	Status
🟨	2	Minor	Resolved

Description

The `GlobalConfig` stores a collection of Addresses that are authorized to execute certain actions. To validate that each action is carried out by it's respective authorized actor, the function 'authorize_license_holder' is called. The function gets the `PaymentCredential` of the given address and from there below cases follow:

1. If the `PaymentCredential` is a `VerificationKeyCredential`, the corresponding pub_key_hash must be present in the extra signatories field of the transaction.

2. If the `PaymentCredential` is a `ScriptCredential`, it has two ways to validate. The script hash must either be in the withdrawals dictionary or be used as the payment_credential of any input of the transaction.

This gives a lot of flexibility to the actor system in the `GlobalConfig`. But there's one particular situation where an actor other than intended can authorize an action. This happens if the intended actor is a "spend" script and the redeemer of such script is not typed (declared as Data). Let's call this the "vulnerable script".

The attack can be done by creating a transaction with a withdraw action involving the vulnerable script. This may look weird, because the spend validator will be expecting three parameters (datum, redeemer and script context), while a withdraw validator would only expect two, given that there's no datum involved. But, when trying to withdraw from a script, the plutus core evaluation machine will apply the redeemer to the first parameter of the script, then, it will apply the script context to the second parameter, and then evaluate the resulting code. But the resulting code is a lambda function. Seeing that there's no more parameters to apply, the evaluation will stop, and it will be considered a success.

So, if an attacker builds the redeemer in such a way that it matched the expected type of the Datum, and the redeemer parameter is not typed (so the script context also has the expected type for the argument it is being applied to), the execution will succeed. And the action will be validated.

This attack could result in a multitude of issues, depending on which actor is vulnerated. The most problematic one of course being the admin.

Recommendation

It is recommended to modify the datum type of `GlobalConfig`. Instead of storing just Addresses, use a type similar to `OrderAuthorizationMethod`, indicating specifically which actor authentication method is to be used (Signature, Spend script, Withdraw script). This way, if an actor needs to be authorized using a Spend validator, the 'authorize_license_holder' function would only look for the inputs, and if the attack is performed the withdraw action would be ignored.

Resolution

Resolved in commit `d299bd13b5b29afb16771fa184225f11739b6693`.

ID-02 Token dust attack on Factory UTxO

	Level	Severity	Status
🟨	3	Medium	Resolved

Description

The `validate_factory` Spend validator just checked for the presence of Factory NFT in the outputs being returned to it. This allowed addition of dust tokens to Factory UTxOs.

```
factory_payment_credential == payment_cred,
value.quantity_of(
  out_value,
  authen_policy_id,
  utils.factory_auth_asset_name,
) == 1,
```

Depending on costing parameters, this could either make it more expensive to insert new pool or make transaction's execution units greater than the script execution budget, which would prevent addition of new pools.

Recommendation

It is recommended to check that Factory Output UTxOs only have Factory NFT and ADA.

Resolution

Resolved in commit d299bd13b5b29afb16771fa184225f11739b6693.