Audit Report

April 7, 2025

Masumi - Payment Service

Contents

1 - Summary 4
1.a - Overview 4
1.b - Process 4
2 - Specification 5
2.a - UTxOs 5
2.b - Assets 5
2.c - Transactions 6
2.d - Payment State Transitions 12
3 - Audited Files 14
4 - Findings 15
5 - MAS-101 Possible to bloat Payment UTxO with trash assets 16
5.a - Description 16
5.b - Recommendation 16
5.c - Resolution 16
6 - MAS-201 Prevent inclusion of reference scripts 17
6.a - Description 17
6.b - Recommendation 17
6.c - Resolution 17
7 - MAS-301 Remove `refund_denied` field 18
7.a - Description 18
7.b - Recommendation 18
7.c - Resolution 18
8 - MAS-302 Rename `CancelDenyRefund` 19
8.a - Description 19
8.b - Recommendation 19
8.c - Resolution 19
9 - MAS-303 Add `state` field to Payment UTxO 20
9.a - Description 20
9.b - Recommendation 20
9.c - Resolution 20
10 - MAS-304 Redundant validation in `WithdrawRefund` 21
10.a - Description 21
10.b - Recommendation 21
10.c - Resolution 21
11 - MAS-305 `SetRefundRequest` has double purpose 22
11.a - Description 22
11.b - Recommendation 22
11.c - Resolution 22
12 - MAS-306 Redundant check for state in SubmitResult 23
12.a - Description 23
12.b - Recommendation 23
12.c - Resolution 23
13 - MAS-307 Payment UTxO `refund_requested` field is not needed 24
13.a - Description 24
13.b - Recommendation 24
13.c - Resolution 24
14 - MAS-308 Validate Payment UTxO initial state 25

14.a - Description 25
14.b - Recommendation 25
14.c - Resolution 25
15 - MAS-309 Unnecessary and suboptimal function `output_value_is_preserved` 26
15.a - Description 26
15.b - Recommendation 26
15.c - Resolution 26
16 - Appendix 27
16.a - Terms and Conditions of the Commercial Agreement 27
16.b - Issue Guide 29
16.c - Revisions 30
16.d - About Us 30

1 - Summary

This report provides a comprehensive audit of Masumi, a decentralized payment service that connects buyers and sellers of a service provided in the real world.

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 Masumi payment service protocol offers the possibility of doing payments to a party that provides a service using the protocol's smart contract.

The protocol has only one UTxO, the Payment UTxO, which is locked into the Vested Payment validator. This UTxO holds the payment information and the value of the payment, which could be in any asset.

The seller must submit a result before the submit result time to be able to withdraw the payment. The protocol receives a fee in the form of a percentage of the payment once the payment cycle finishes with withdrawal of funds by the seller.

The buyer can request a refund at any time before the unlock time. If there's no result submitted, the buyer can withdraw the full payment. If there's a result submitted, the buyer can request a refund, in which case the seller must authorize the refund.

If there's a dispute between the buyer and the seller, meaning a result was submitted and the buyer request a refund, the protocol has a multisignature dispute resolution mechanism that allows the administrators to resolve disputes in favor of any of the parties.

1.b - Process

Our audit process involved a thorough examination of Masumi 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 protocol's mathematical model to verify that the implemented equations matched the expected behavior.

Findings and feedback from the audit were communicated regularly to the Masumi team through Discord. Diagrams illustrating the necessary transaction structure for proper interaction with the protocol are attached as part of this report. The Masumi 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 - Buyer Payment

Main UTxO of the protocol. Holds the payment information and its value.

- Address: Vested Pay validator script hash
- Value:
  - any assets
- Datum:
  - buyer: VerificationKeyHash
  - seller: VerificationKeyHash
  - reference_id: ByteArray
  - result_hash: ByteArray
  - submit_result_time: POSIXTime
  - unlock_time: POSIXTime
  - external_dispute_unlock_time: POSIXTime
  - seller_cooldown_time: POSIXTime
  - buyer_cooldown_time: POSIXTime
  - state: State

2.a.b - Fees UTxO

Holds the fees collected by a Seller's funds unlock.

- Address: defined by parameter `fee_address` in the Vested Payment validator.
- Value:
  - same assets as the Payment UTxO

2.b - Assets

2.b.a - Registry token

Is a token linked to a payment contract and is used for discovery of agents.

- Policy ID: hash of this script.
- Name: hash of tx ID and output index of some input being spent in the minting transaction.

2.b.b - External assets

Payments can be made in any asset, these are considered as *external* assets i.e. not minted by the protocol.

2.c - Transactions

2.c.a - Buyer Transactions

2.c.a.a - Lock Funds

The Buyer makes the payment by sending assets to the Vested Payment script. The Payment UTxO datum holds the relevant information for further processing.

This transaction does not involve any Plutus scripts.

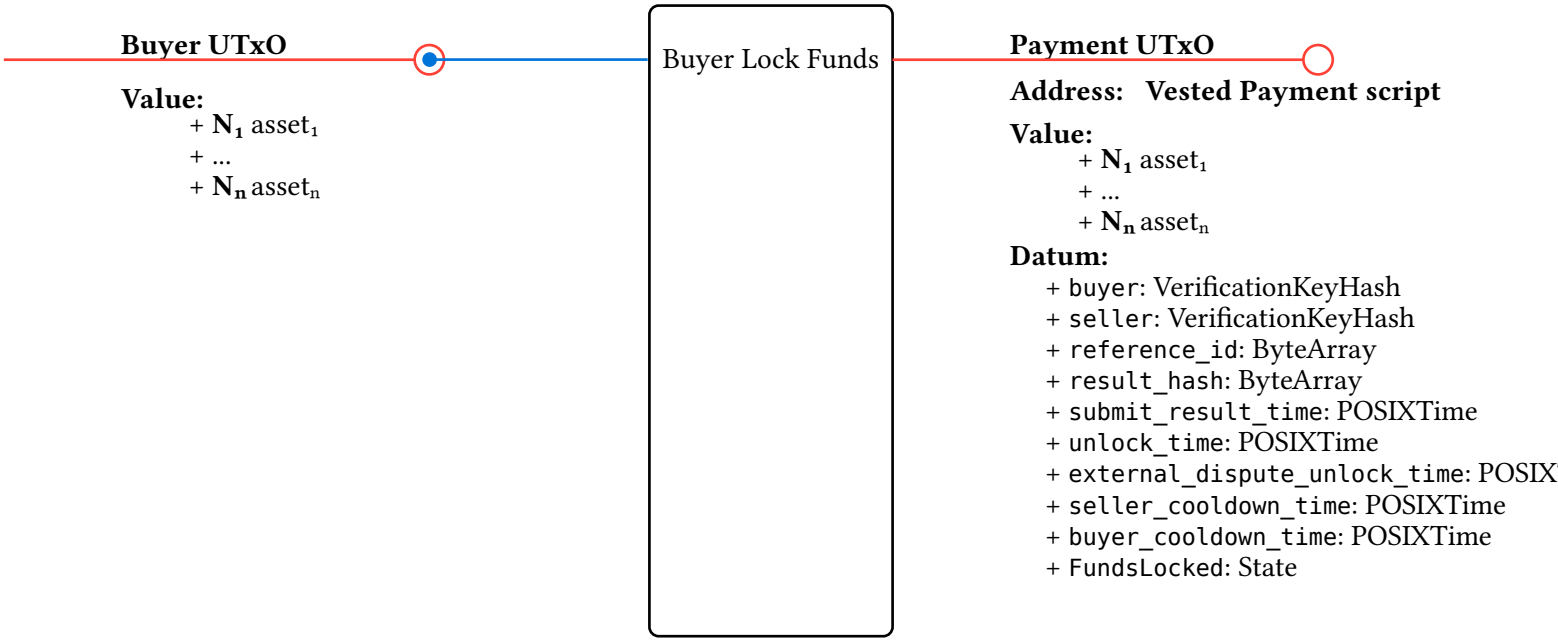


Figure 1: Buyer Lock Funds transaction

2.c.a.b - Request Refund

The Buyer can request a refund of its payment by spending the Payment UTxO that lists them as the buyer. Must be done before the unlock time.

This action sets the `state` field to `RefundRequested` or `Disputed` depending on the previous state.

Involved redeemers:

- `SetRefundRequested`, `Spend` purpose: for spending the Payment UTxO

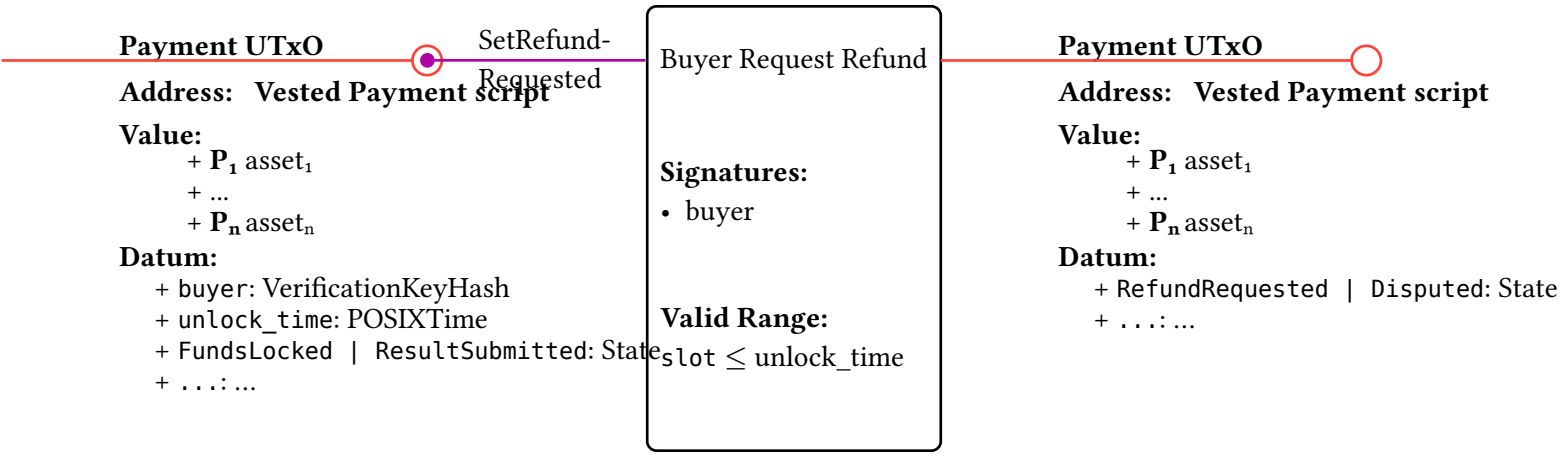


Figure 2: Buyer Request Refund transaction

2.c.a.c - Cancel Refund Request

The Buyer can cancel the refund request by spending the Payment UTxO that lists them as the buyer. Can be done at any time.

This action sets the `state` field to `FundsLocked` or `ResultSubmitted` depending on the previous state.

Involved redeemers:

- `UnSetRefundRequested`, `Spend` purpose: for spending the Payment UTxO

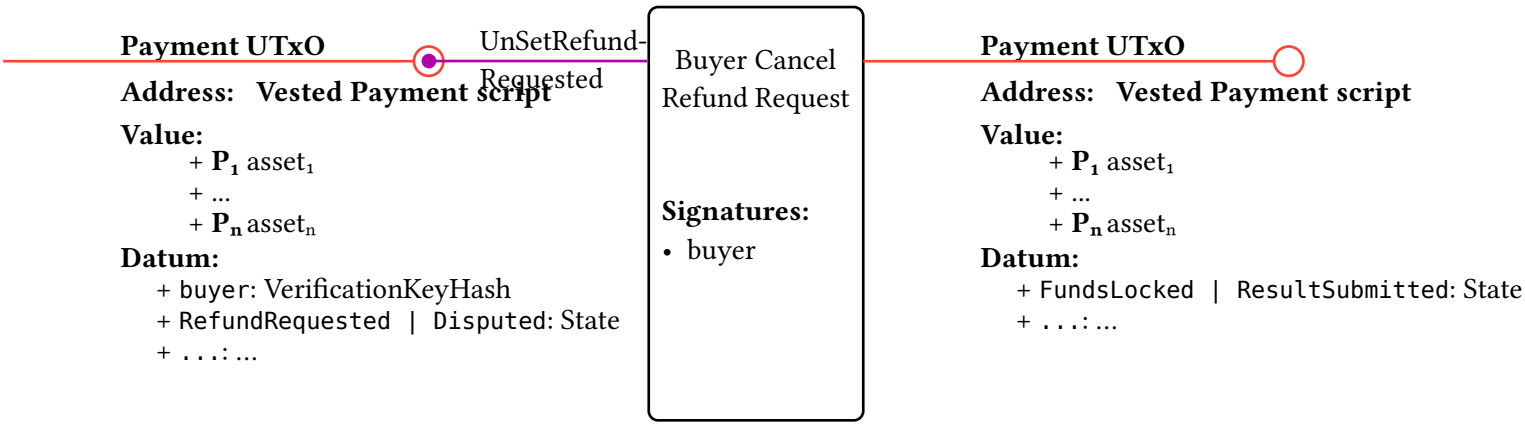


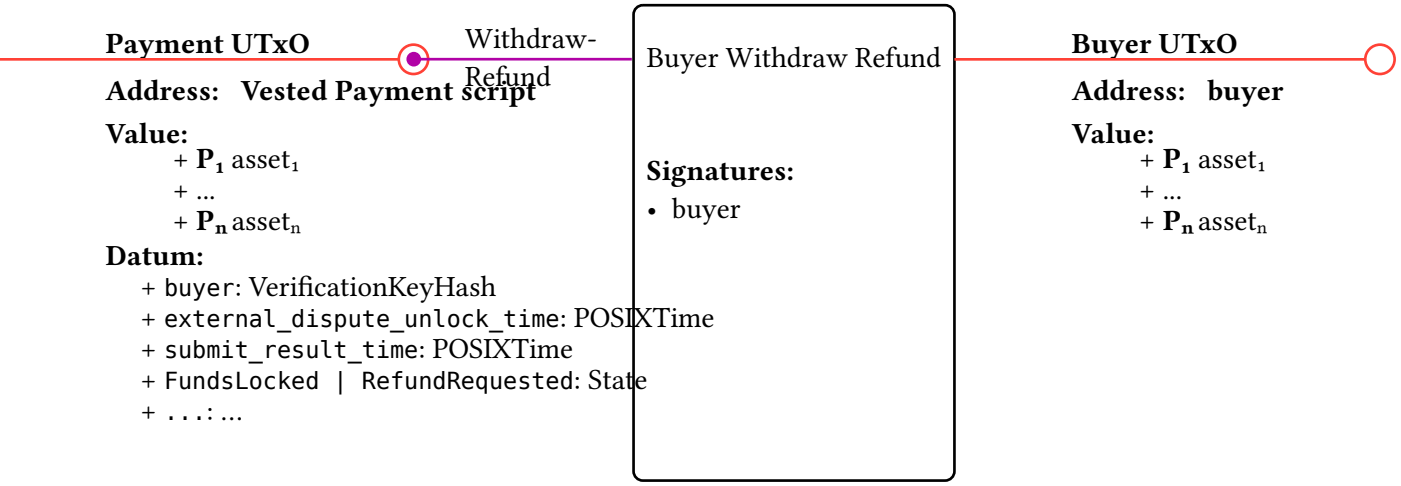
Figure 3: Buyer Cancel Refund Request transaction

2.c.a.d - Withdraw Refund

The Buyer can withdraw the requested refund by spending the Payment UTxO in the cases where the refund request was automatically approved (no denial by the Seller) or when no result was submitted after the submit result time.

Involved redeemers:

- `WithdrawRefund`, `Spend` purpose: for spending the Payment UTxO



Note: external_dispute_unlock_time <= slot or submit_result_time <= slot

Figure 4: Buyer Withdraw Refund transaction

2.c.b - Seller Transactions

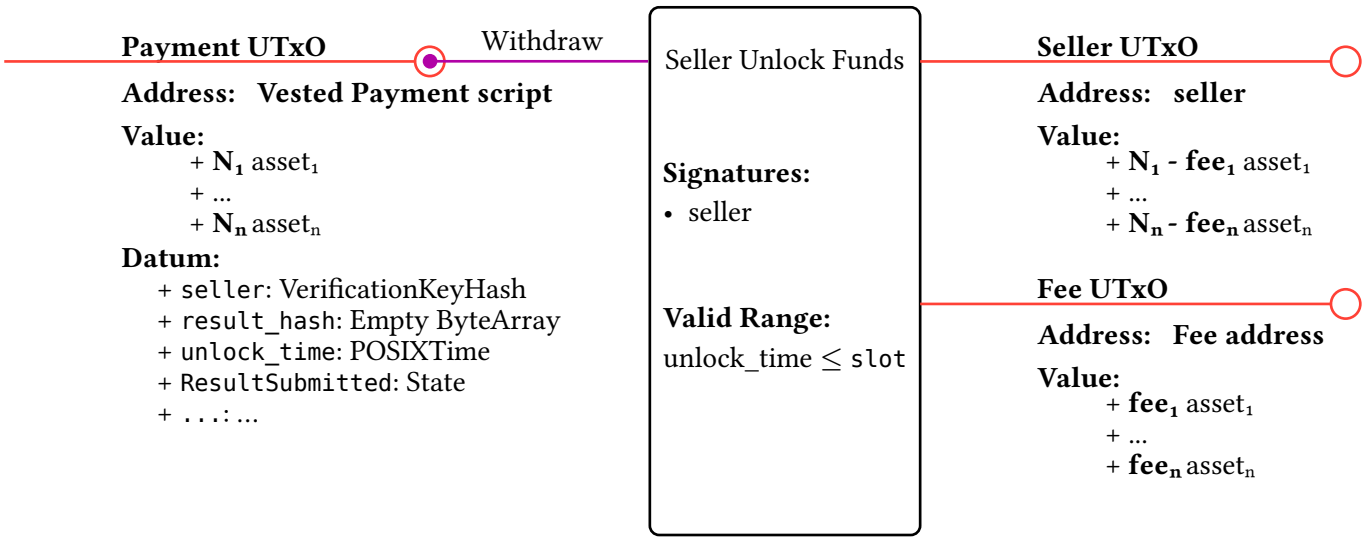
2.c.b.a - Unlock Funds

The Seller can claim the funds by spending the Payment UTxO that lists them as the designated seller. This can be done at any time after the unlock time.

When unlocking funds, a fee is taken from the Payment UTxO and transferred to a designated fee address. The remaining balance goes to the Seller. Both the fee percentage (specified in permille) and the fee recipient address are parameters of the Vested Payment validator.

Involved redeemers:

- `Withdraw`, `Spend` purpose: for spending the Payment UTxO



Note: $fee_i = N_i * fee\_permille / 1000$, where $i \in \{1,...,n\}$

Figure 5: Seller Unlock Funds transaction

2.c.b.b - Submit Result

The Seller can submit the result of the payment by updating the Payment UTxO datum with the `result_hash` field.

Involved redeemers:

- `SubmitResult`, `Spend` purpose: for spending the Payment UTxO

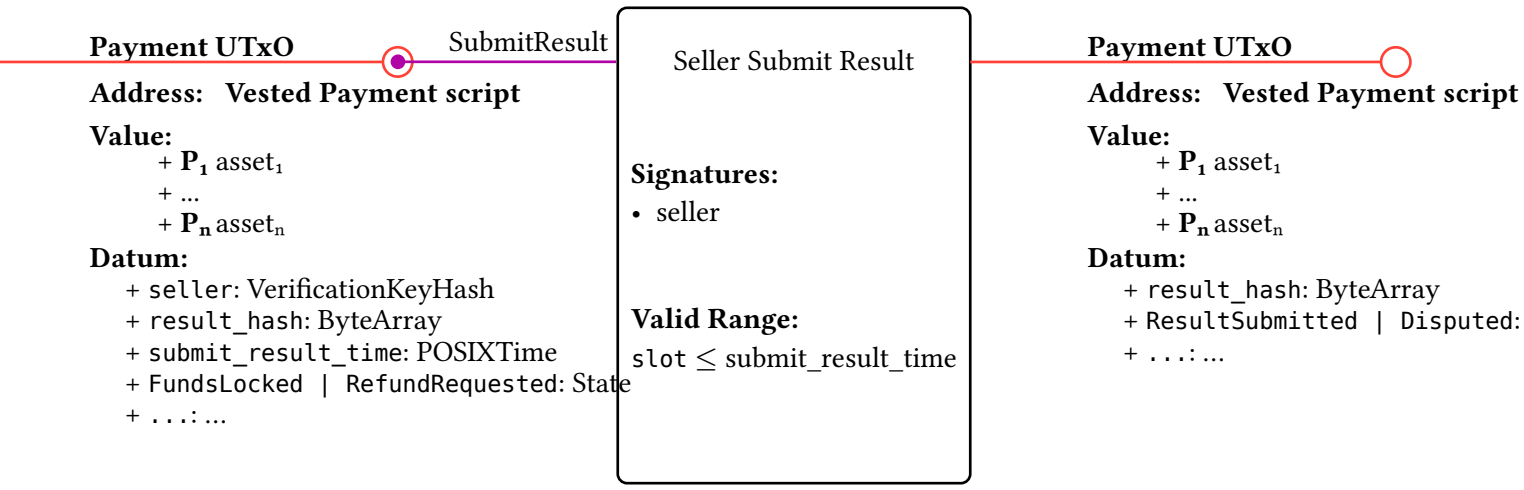


Figure 6: Seller Submit Result transaction

2.c.b.c - Authorize Refund

The Seller can authorize refund. The result hash must be set to empty in the Payment output datum, but the input result hash could be anything thus allowing to overwrite it.

Involved redeemers:

- `AuthorizeRefund`, `Spend` purpose: for spending the Payment UTxO

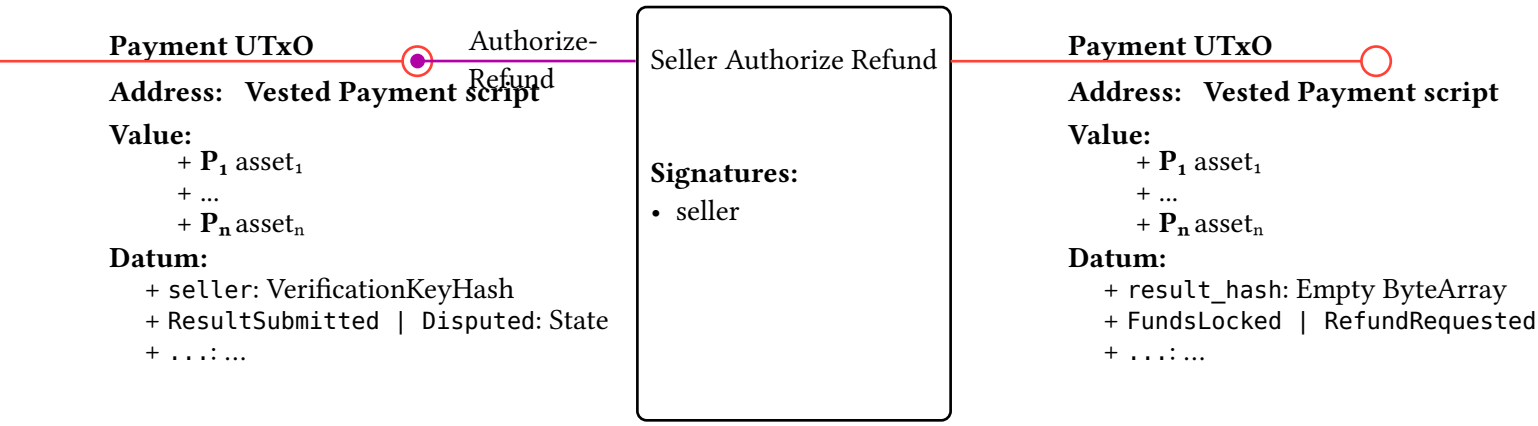


Figure 7: Seller Authorize Refund transaction

2.c.c - Admin Transactions

2.c.c.a - Resolve Dispute

An Admin can resolve the dispute by spending the Payment UTxO. The transaction must be signed by `required_admins_multi_sig` of the `N` admins (both defined in the Vested Payment validator parameter).

Involved redeemers:

- `WithdrawDisputed`, `Spend` purpose: for spending the Payment UTxO

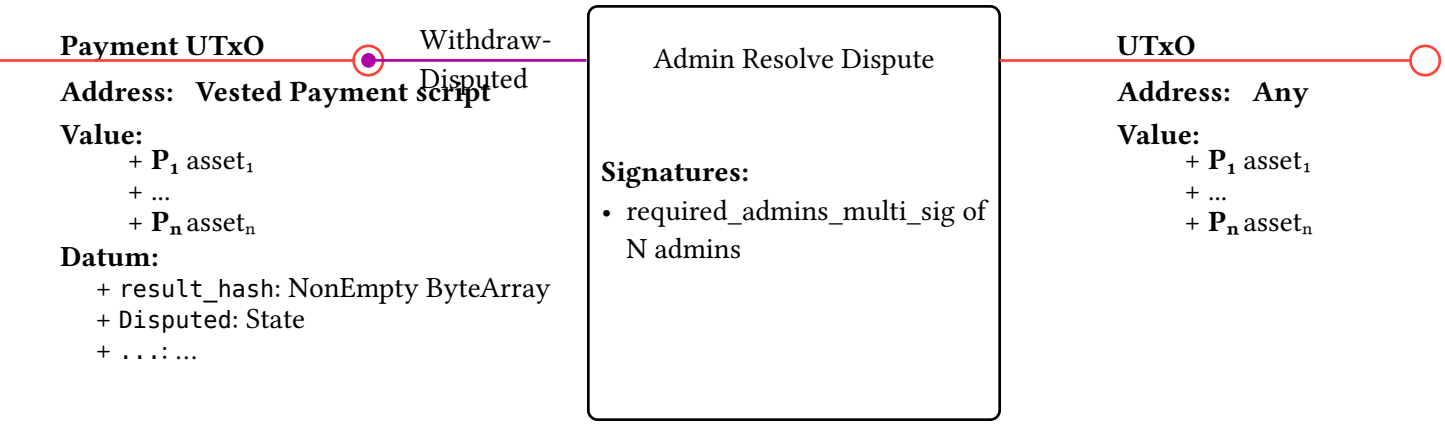


Figure 8: Admin Resolve Dispute transaction

2.d - Payment State Transitions

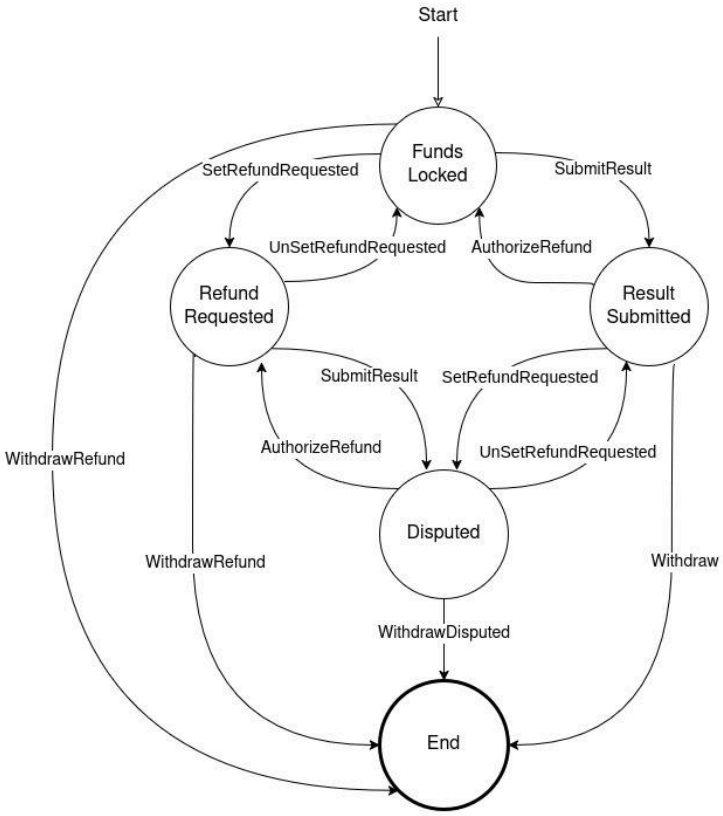


Figure 9: Payment State Transitions Diagram

2.d.a - States

Each state is defined by the following datum fields values.

2.d.a.a - Funds Locked

Initial state after the buyer locks their funds in the contract.

- state: FundsLocked
- result_hash: None

2.d.a.b - Refund Requested

State after the buyer has requested a refund.

- state: RefundRequested
- result_hash: None

2.d.a.c - Result Submitted

State after the seller has submitted the result.

- state: ResultSubmitted
- result_hash: Some(hash)

2.d.a.d - Disputed

State when both a refund is requested, and result is submitted which implies that the refund is denied.

- state: Disputed
- result_hash: Some(hash)

2.d.a.e - End

Final state after funds have been withdrawn by any party.

- UTxO is consumed
- No more state transitions possible

3 - Audited Files

Below is a list of all audited files 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 `91ae39c85f4e5529a04b676e8cf9451f8466e55d`.

Filename
./smart-contracts/registry/validators/mint.ak
./smart-contracts/payment/validators/vested_pay.ak

4 - Findings

ID	Title	Severity	Status
MAS-101	Possible to bloat Payment UTxO with trash assets	Major	Resolved
MAS-201	Prevent inclusion of reference scripts	Minor	Resolved
MAS-301	Remove `refund_denied` field	Info	Resolved
MAS-302	Rename `CancelDenyRefund`	Info	Resolved
MAS-303	Add `state` field to Payment UTxO	Info	Resolved
MAS-304	Redundant validation in `WithdrawRefund`	Info	Resolved
MAS-305	`SetRefundRequest` has double purpose	Info	Resolved
MAS-306	Redundant check for state in SubmitResult	Info	Resolved
MAS-307	Payment UTxO `refund_requested` field is not needed	Info	Resolved
MAS-308	Validate Payment UTxO initial state	Info	Acknowledged
MAS-309	Unnecessary and suboptimal function `output_value_is_preserved`	Info	Resolved

5 - MAS-101 Possible to bloat Payment UTxO with trash assets

Category	Commit	Severity	Status
Vulnerability	44822377c6518e979b4e7e708ecf9c17d4badc46	Major	Resolved

5.a - Description

The validator allows operations that consume and recreate the Payment UTxO to add arbitrary assets to the output. This is because the validation only checks that the output value is greater than or equal to the input value, without enforcing exact equality. As a result, malicious users could bloat the UTxO with unwanted tokens.

5.b - Recommendation

Enforce exact equality of the input and output values for all assets but ADA. Use a check that only allows increasing the amount of locked ADA in the Payment UTxO.

5.c - Resolution

Resolved in commit `bd4e6b94193671616572ab1c66c7f8a58b1d1ac0`. Resolution lead to a new finding MAS-309.

6 - MAS-201 Prevent inclusion of reference scripts

Category	Commit	Severity	Status
Improve-ment	44822377c6518e979b4e7e708ecf9c17d4badc46	Minor	Resolved

6.a - Description

With the addition of the `minFeeRefScriptsCoinsPerByte` protocol parameter in the Voltaire era, including a reference script in any input (whether it's a reference or not) will impact the transaction fees, regardless of whether the script is executed. Given that the reference script field is not validated in any output of the protocol, there's an attack vector where a malicious party includes a huge reference script in every output of a transaction, costing more fees to the next party interacting with those UTxOs.

6.b - Recommendation

Ensure that any UTxO belonging to the protocol does not include a reference script.

6.c - Resolution

Resolved in commit `7bc49d9`.

7 - MAS-301 Remove `refund_denied` field

Category	Commit	Severity	Status
Redundancy	44822377c6518e979b4e7e708ecf9c17d4badc46	Info	Resolved

7.a - Description

The `refund_denied` field is redundant with the `result_hash` field. It is always set to `False` if `result_hash` is `None` and `True` when `result_hash` is set.

7.b - Recommendation

Remove the `refund_denied` field.

7.c - Resolution

Resolved in commit `ef6889b`.

8 - MAS-302 Rename `CancelDenyRefund`

Category	Commit	Severity	Status
	44822377c6518e979b4e7e708ecf9c17d4badc46	Info	Resolved

8.a - Description

The `CancelDenyRefund` redeemer name is misleading. Not only cancels a refund denial by setting the `refund_denied` field to `False`, but also clears the `result_hash` field.

8.b - Recommendation

A possibility could be `CancelSubmitResult`.

8.c - Resolution

Resolved in commit `7bc49d9`.

9 - MAS-303 Add `state` field to Payment UTxO

Category	Commit	Severity	Status
	44822377c6518e979b4e7e708ecf9c17d4badc46	Info	Resolved

9.a - Description

The Payment UTxO state must be deduced from the datum fields:

- `result_hash`
- `refund_requested`
- `refund_denied`

It would be useful to have a single field to quickly identify the state of the payment.

9.b - Recommendation

Add a `state` field to the Payment UTxO datum.

9.c - Resolution

Resolved in commit `ef6889b`.

10 - MAS-304 Redundant validation in `WithdrawRefund`

Category	Commit	Severity	Status
Redundancy	44822377c6518e979b4e7e708ecf9c17d4badc46	Info	Resolved

10.a - Description

The last `or` check of the `WithdrawRefund` redeemer is redundant. The second part of the `or` implies the first part, so it is safe to remove the first part.

10.b - Recommendation

Remove the first part of the last `or` check.

10.c - Resolution

Resolved in commit `046a0c6`.

11 - MAS-305 `SetRefundRequest` has double purpose

Category	Commit	Severity	Status
Ambiguity	eaff234c7933ae24c00eceb8160c02b3925488db	Info	Resolved

11.a - Description

The `SetRefundRequest` redeemer has double purpose:
- Request a refund by setting the `refund_requested` field to `True`
- Adding funds to the Payment UTxO

11.b - Recommendation

Split the `SetRefundRequest` redeemer into two: one for requesting a refund and another one for adding funds.

11.c - Resolution

Resolved in commit `8551c1a`.

12 - MAS-306 Redundant check for state in SubmitResult

Category	Commit	Severity	Status
Redundancy	cf3bcbcd6b2b0779bd157cfb91f1184e8dfeb7a3	Info	Resolved

12.a - Description

SubmitResult redeemer can be used in any of the four possible states, so it is not necessary to have a check for it. In other words, the following check is not necessary because it always evaluates to true:

```
expect or {
    state == FundsLocked,
    state == RefundRequested,
    //allow to set the field again, to update to a new result
    state == ResultSubmitted,
    state == Disputed,
  }
```

12.b - Recommendation

Remove the redundant check. If the purpose of the check is to make an explicit statement that any state is possible, use inline comments to indicate this.

12.c - Resolution

Resolved in commit `fabc3e7f3795c1583f8557646cfb94313dbbc6e3`.

13 - MAS-307 Payment UTxO `refund_requested` field is not needed

Category	Commit	Severity	Status
Redundancy	f4cb7ebd848f708980a67e0bd60e165b450bb1ac	Info	Resolved

13.a - Description

Now that there's a `state` field in the Payment UTxO datum, the `refund_requested` field is redundant. Sufficies with ensuring that the transitions of the `state` field are valid.

13.b - Recommendation

Remove the `refund_requested` field.

13.c - Resolution

Resolved in commit `2d7afde`.

14 - MAS-308 Validate Payment UTxO initial state

Category	Commit	Severity	Status
Robustness	44822377c6518e979b4e7e708ecf9c17d4badc46	Info	Acknowledged

14.a - Description

It is possible to create a Payment UTxO with an invalid state i.e. bad datum fields values.

14.b - Recommendation

Validate the Payment UTxO initial state by adding a minting purpose to the vested payment validator. The minting operation would check the datum fields values and only allow valid values, and pay the minted token to the Payment UTxO. The presence of this token would indicate that the initial datum fields values of the UTxO are correct.

14.c - Resolution

The **project team** decided not to resolve this finding to avoid an increase in transaction costs. Users will be responsible for identifying well-formed Payment UTxOs. As the **audit team** we endorse the decision, as this finding is only informational.

15 - MAS-309 Unnecessary and suboptimal function `output_value_is_preserved`

Category	Commit	Severity	Status
Coding style	cf3bcbcd6b2b0779bd157cfb91f1184e8dfeb7a3	Info	Resolved

15.a - Description

Function `output_value_is_preserved` checks that two given values `v1` and `v2` are equal besides the lovelace amount where `v2` can be greater than `v1`. This is exactly what the `match` function from the standard library does, when the third parameter is `<=`. Moreover, `match` is efficiently implemented while `output_value_is_preserved` has several performance issues (like repeated calls to `flatten`).

15.b - Recommendation

Remove `output_value_is_preserved` and use `match` instead.

15.c - Resolution

Resolved in commit `a9acad19d15a738e0342779371ee86dbb814621a`.

16 - Appendix

16.a - Terms and Conditions of the Commercial Agreement

16.a.a - Confidentiality

Both parties agree, within a framework of trust, to discretion and confidentiality in handling the business. This report cannot be shared, referred to, altered, or relied upon by any third party without Txpipe LLC, 651 N Broad St, Suite 201, Middletown registered at the county of New Castle, written consent.

The violation of the aforementioned, as stated supra, shall empower TxPipe to pursue all of its rights and claims in accordance with the provisions outlined in Title 6, Subtitle 2, Chapter 20 of the Delaware Code titled "Trade Secrets,", and to also invoke any other applicable law that protects or upholds these rights.

Therefore, in the event of any harm inflicted upon the company's reputation or resulting from the misappropriation of trade secrets, the company hereby reserves the right to initiate legal action against the contractor for the actual losses incurred due to misappropriation, as well as for any unjust enrichment resulting from misappropriation that has not been accounted for in the calculation of actual losses.

16.a.b - Service Extension and Details

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/code analyzed.

This agreement does not authorize the client Masumi to make use of the logo, name, or any other unauthorized reference to Txpipe LLC, except upon express authorization from the company.

TxPipe LLC shall not be liable for any use or damages suffered by the client or third-party agents, nor for any damages caused by them to third parties. The sole purpose of this commercial agreement is the delivery of what has been agreed upon. The company shall be exempt from any matters not expressly covered within the contract, with the client bearing sole responsibility for any uses or damages that may arise.

Any claims against the company under the aforementioned terms shall be dismissed, and the client may be held accountable for damages to reputation or costs resulting from non-compliance with the aforementioned provisions. **This report provides general information and is not intended to constitute financial, investment, tax, legal, regulatory, or any other form of advice.**

Any conflict or controversy arising under this commercial agreement or subsequent agreements shall be resolved in good faith between the parties. If such negotiations do not result in a conventional agreement, the parties agree to submit disputes to the courts of Delaware and to the laws of that jurisdiction under the powers conferred by the Delaware Code, TITLE 6, SUBTITLE I, ARTICLE 1, Part 3 § 1-301. and Title 6, SUBTITLE II, chapter 27 §2708.

16.a.c - Disclaimer

The audit constitutes a comprehensive examination and assessment as of the date of report submission. The company expressly disclaims any certification or endorsement regarding the subsequent performance, effectiveness, or efficiency of the contracted entity, post-report delivery, whether resulting from modification, alteration, malfeasance, or negligence by any third party external to the company.

The company explicitly disclaims any responsibility for reviewing or certifying transactions occurring between the client and third parties, including the purchase or sale of products and services.

This report is strictly provided for ***informational purposes*** and reflects solely the due diligence conducted on the following files and their corresponding hashes using sha256 algorithm:

Filename: `./smart-contracts/registry/validators/mint.ak` **Hash**: `5134d96e552ab050c7315f2d90083e47d82f973a88dece13a0a5570fb31f45d1`
Filename: `./smart-contracts/payment/validators/vested_pay.ak` **Hash**: `ba0f213240ccdd01b540056852780600f052bf050301f2adfe359bcde42b933a`

TxPipe advocates for the implementation of multiple independent audits, a publicly accessible bug bounty program, and continuous security auditing and monitoring. Despite the diligent manual review processes, the potential for errors exists. TxPipe strongly advises seeking multiple independent opinions on critical matters. It is the firm belief of TxPipe that every entity and individual is responsible for conducting their own due diligence and maintaining ongoing security measures.

16.b - Issue Guide

16.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.

16.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.

16.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.

16.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.

16.d.a - Links

- Website
- Email
- Twitter