TxPipe SHOP

Audit Report

June 6, 2025

IOG - Treasury Contracts Audit

Contents

1 - Summary

This report provides a comprehensive audit of the Treasury contracts, which provide a simple but robust method for managing funds withdrawn from the Cardano treasury.

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The Treasury contracts provide a simple but robust way to manage funds withdrawn from the Cardano treasury, ensuring that the funds cannot be delegated nor used in governance voting.

The protocol has three kind of UTxOs:
- The Treasury Reserve UTxOs (Treasury UTxOs for short), which hold the funds available to be allocated to projects over the current fiscal period.
- The Vendor UTxOs, which hold the funds allocated to specific projects.
- The Registry UTxO, which holds the script hashes of the Treasury and Vendor validators.

The protocol uses a multisignature resolution mechanism for several of the operations, based on the SundaeSwap Aicone library. These permissions are encoded in the parameters of the Treasury and Vendor validators.

At the beginning of the fiscal period, the Treasury UTxOs are created by withdrawing from the Treasury script which in turn is funded by the Cardano treasury via a governance action.

The Treasury UTxOs are then consumed to fund projects by creating Vendor UTxOs. Additionally, the Treasury UTxOs can be reorganized or disbursed to an arbitrary address with an arbitrary datum, for example, to exchange ADA for a stablecoin. At the end of the fiscal year, any remaining funds are swept back into the Cardano Treasury as a donation.

The Vendor UTxOs are used to pay projects owners. Funds form these UTxOs can be withdrawn by the project owner once the payouts are matured, that is, when the payment time has passed and the payout status is active. Payouts can be modified by both the project owner and the permissioned committee, and can also be paused by a designated permissioned committee. Unmatured funds in Vendor UTxOs may be swept back to the Treasury UTxOs at the end of the fiscal year. Additionally, if a Vendor UTxO contains a malformed datum, the associated funds can be returned to the Treasury UTxOs to prevent them from becoming permanently locked.

1.b - Process

Our audit process involved a thorough examination of Treasury 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 SundaeSwap team through Discord. Diagrams illustrating the necessary transaction structure for proper interaction with the protocol 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 - Treasury UTxOs

Treasury UTxOs hold the treasury funds available to be allocated to projects over the current fiscal period.

- Address:
  - Payment part: Treasury validator script hash. Parameters:
    - `registry_token`: policy ID of the token that identifies the registry
    - `permissions`: permissions required for different actions over the treasury script UTxOs
      - `reorganize`: to reorganize Treasury UTxOs (Section 2.c.a.c)
      - `sweep`: to sweep back funds before expiration (Section 2.c.a.f)
      - `fund`: to fund projects (Section 2.c.a.b)
      - `disburse`: to pay funds to an arbitrary destination (Section 2.c.a.d and Section 2.c.a.e)
    - `expiration`: time after which the funds can be swept back to the treasury
    - `payout_upperbound`: upper bound for any payouts created by the oversight committee
  - Staking part: empty
- Value: The assets used for funding (see Section 2.b.b).
- Datum: any

2.a.b - Vendor UTxOs

The Vendor UTxOs hold the funds allocated to a specific project. Its datum encodes the project owner (also known as the Vendor) and the chronogram of payouts that the owner is entitled to receive during the current fiscal period.

- Address:
  - Payment part: Vendor validator script hash. Parameters:
    - `registry_token`: policy ID of the token that identifies the registry
    - `permissions`: permissions required for different actions over the vendor script UTxOs
      - `pause`: to pause a payout (Section 2.c.b.b)
      - `resume`: to resume a paused payout (Section 2.c.b.b)
      - `modify`: to modify or cancel the project (Section 2.c.b.c)
    - `expiration`: time after which the funds can be swept back to the treasury
  - Staking part: empty
- Value: The assets used for funding (see Section 2.b.b).
- Datum: `VendorDatum`
  - `vendor`: multisig that identifies the project owner
  - `payouts`: list of payouts, where each payout has the following fields:
    - `maturation`: POSIX time for the unlocking
    - `value`: assets and amounts to be unlocked
    - `status`: if it is active or paused

2.a.c - Registry UTxO

The Registry UTxO holds the script hashes of the Treasury and Vendor validators for a particular instantiation. This allows both validators to access each other's script hash without introducing circular dependencies.

- Address:
  - Payment part: `oneshot` multivalidator with an "always false" spending script. Parameters:
    - `utxo_ref`: input that ensures one-shot minting

    - Staking part: any
- Value: Registry NFT, minted under same `oneshot` multivalidator policy (see).
- Datum: `ScriptHashRegistry`
    - `treasury`: script hash of the Treasury validator
    - `vendor`: script hash of the Vendor validator

2.b - Assets

2.b.a - Registry NFT

Only asset minted within the protocol. Minted by a one-shot minting policy. Locked into the Registry UTxO.

- Policy ID: hash of oneshot validator script hash.
- Token name: `REGISTRY`

2.b.b - Funding assets

External assets used for funding the projects. ADA is the primary asset expected to be used. Stablecoins such as USDMa or USDm are also considered. However, the contracts are designed to support any asset.

For simplicity, throughout this report we will use 'USD' as a shortand for any funding asset other than ADA.

2.c - Transactions

2.c.a - Treasury Reserve

The Treasury Reserve UTxO datum could be anything. We won't include that field in the Treasury UTxOs shown in the diagrams.

Also, the UTxOs could hold any assets, not just ADA or stablecoins. We'll show only ADA and USD in the diagrams for simplicity only.

There are two ways to create new Treasury UTxOs:

- `Withdraw`
- Paying to the Treasury script

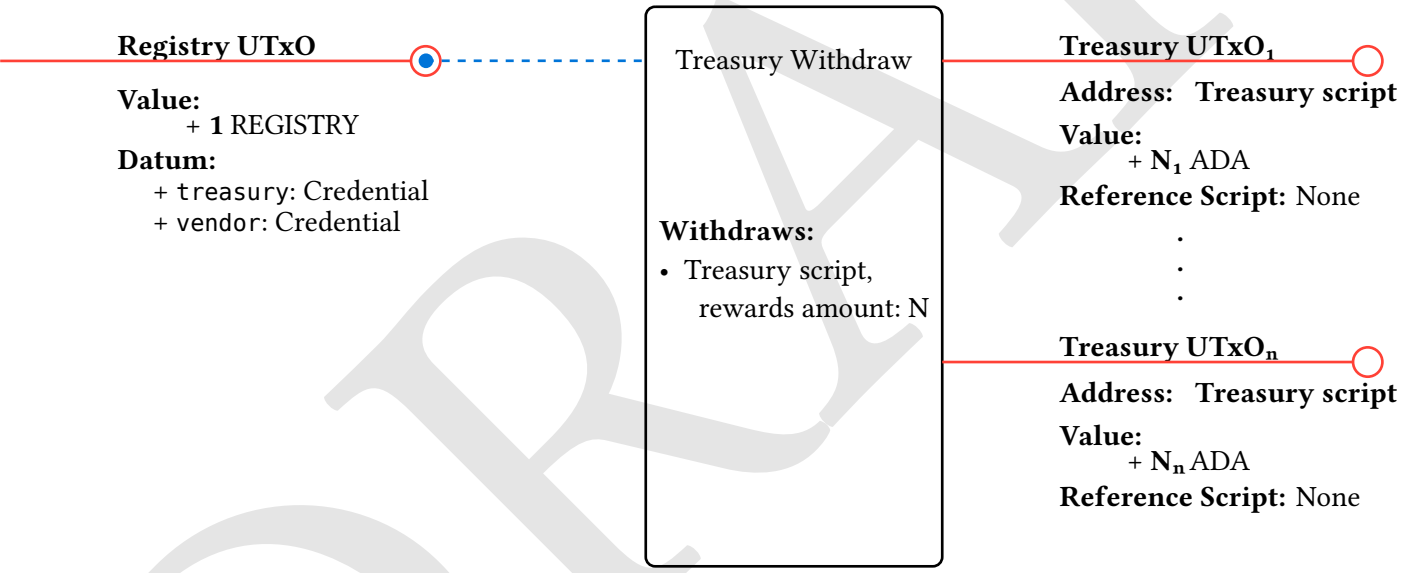
The last one does not involve any script validation. We must ensure that this is safe.

2.c.a.a - Withdraw

Anyone can trigger a withdrawal of funds from the Treasury Reserve reward account as long as all the withdrawn amount is paid to the spending portion of the treasury script only i.e. without a staking address.

Involved redeemers:

- `Data` redeemer, `Withdraw` purpose: for withdrawing the Treasury script reward account.



Note: $N_1 + ... + N_n = N$

Figure 1: Treasury Withdraw transaction

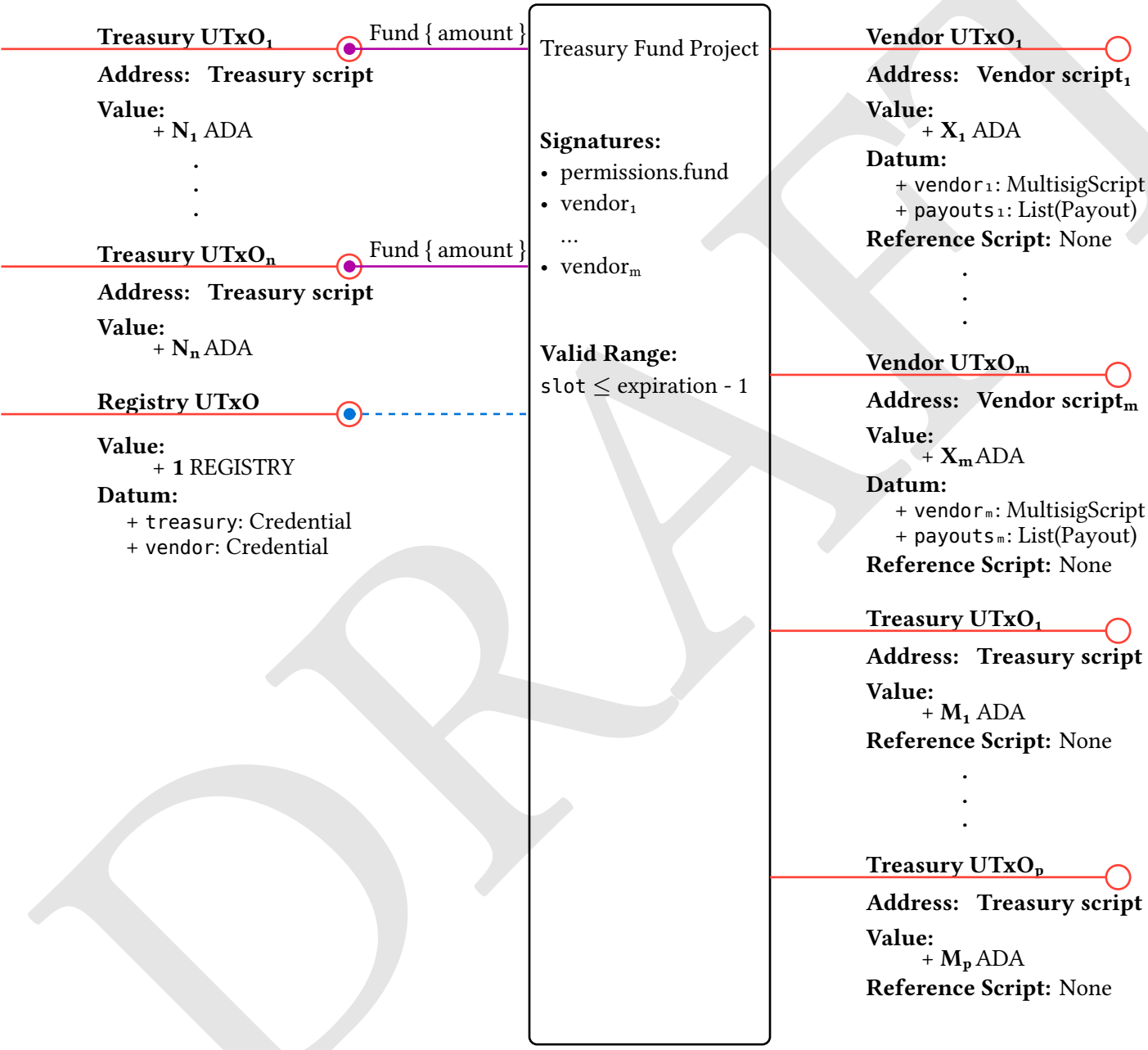
2.c.a.b - Fund project

The committee permissioned to fund projects approves a project and funds it by the submission of a `Fund` transaction.

For this version of the protocol, the funding must come from a *single* Treasury script address, potentially from multiple UTxOs sitting at that address.

Involved redeemers:

- `Fund { amount }` redeemer, `Spend` purpose: for spending each of the Treasury inputs.



Note:

amount = X_1 + ... + X_m

amount ≤ N_1 + ... + N_n

M_1 + ... + M_p = N_1 + ... + N_n - amount

Payout: { maturation: Int, value: Pairs<PolicyId, Pairs<AssetName, Int>>, status: PayoutStatus }

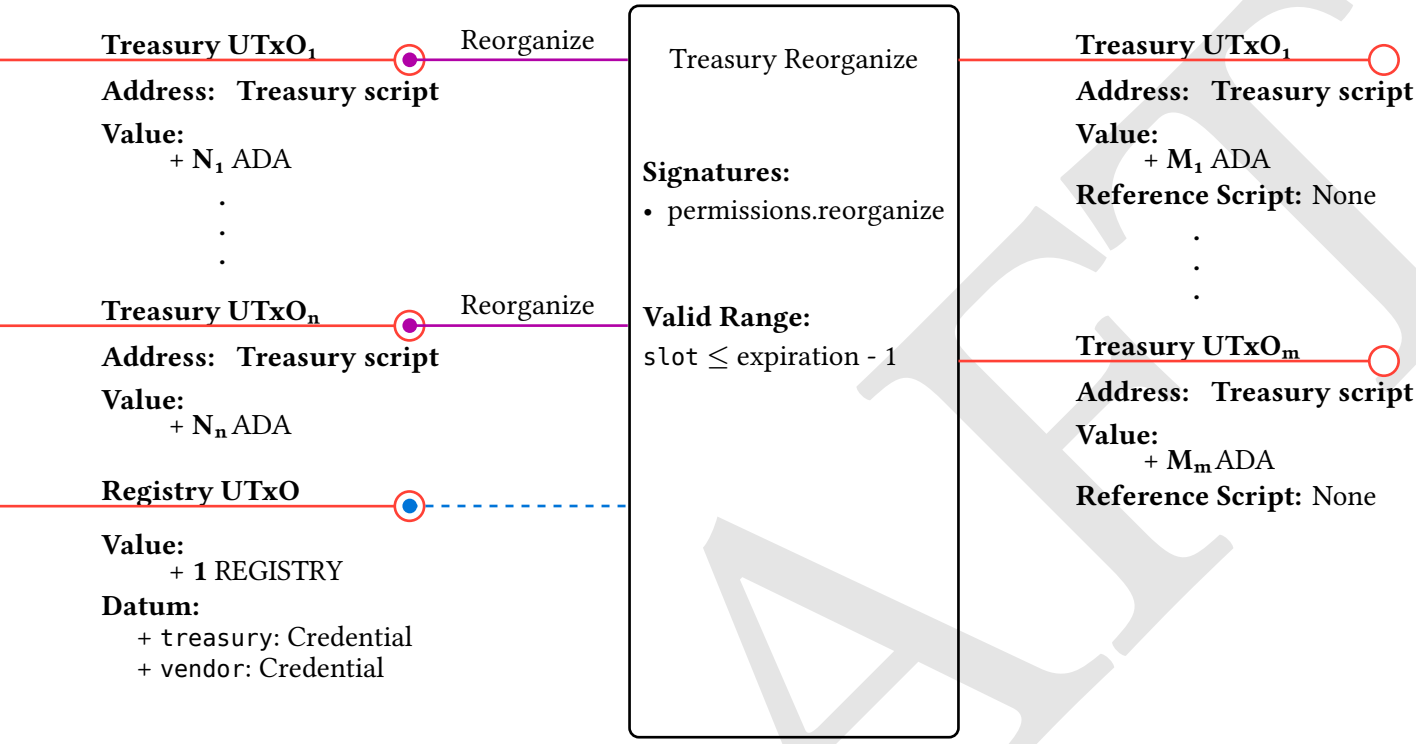
Figure 2: Fund Project transaction

2.c.a.c - Reorganize

The committee permissioned to reorganize the treasury can do so by the submission of a `Reorganize` transaction.

Involved redeemers:

- `Reorganize` redeemer, `Spend` purpose: for spending each of the Treasury inputs.



Note: $N_1 + ... + N_n \leq M_1 + ... + M_m$

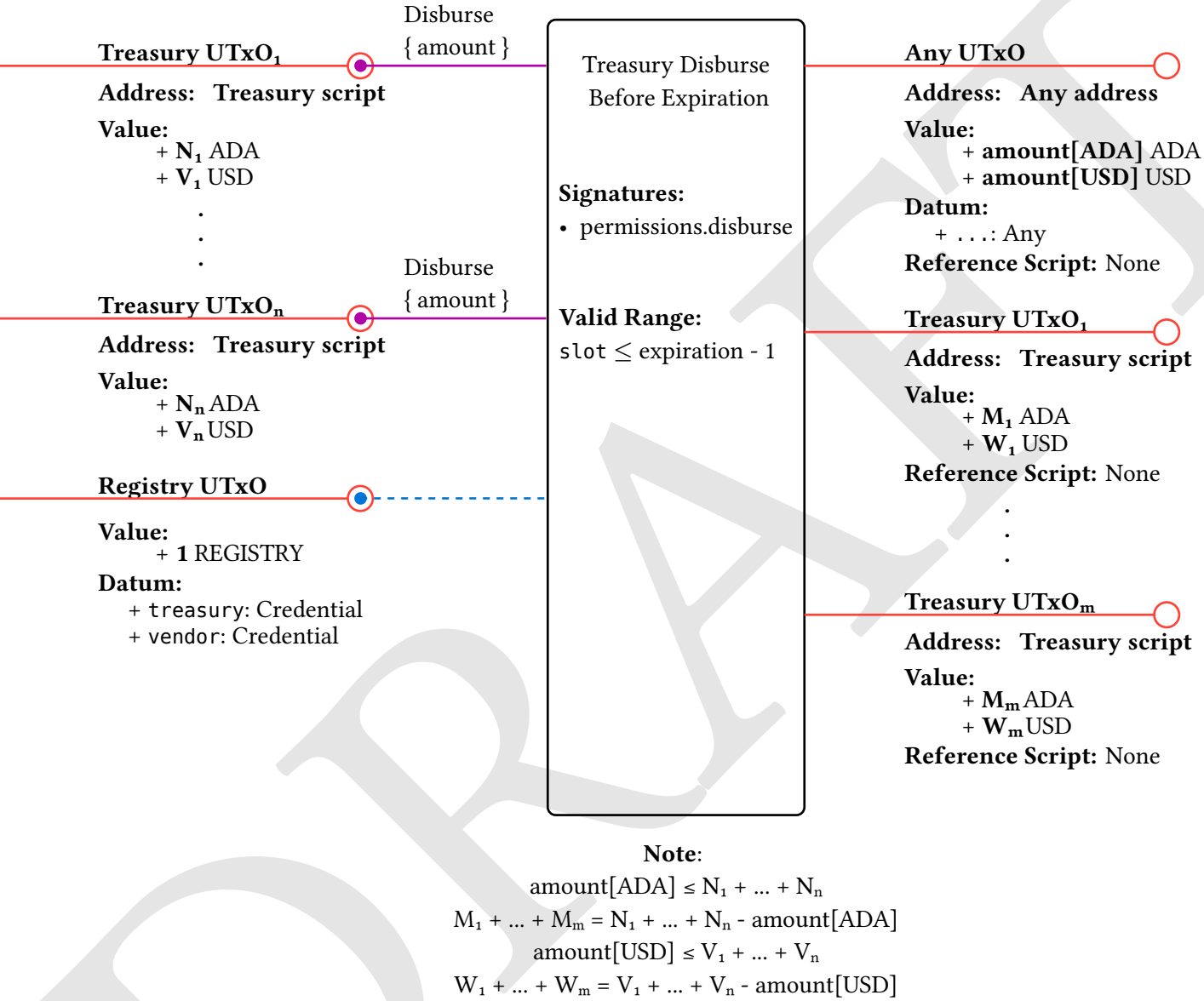
Figure 3: Reorganize Treasury transaction

2.c.a.d - Disburse Before Expiration

The permissioned committee can disburse funds an arbitrary amount of funds to an arbitrary address with an arbitrary datum. This might be used to support conversion to and from stablecoins, or fiat payments.

Involved redeemers:

- `Disburse { amount }` redeemer, `Spend` purpose: for spending each of the Treasury inputs.



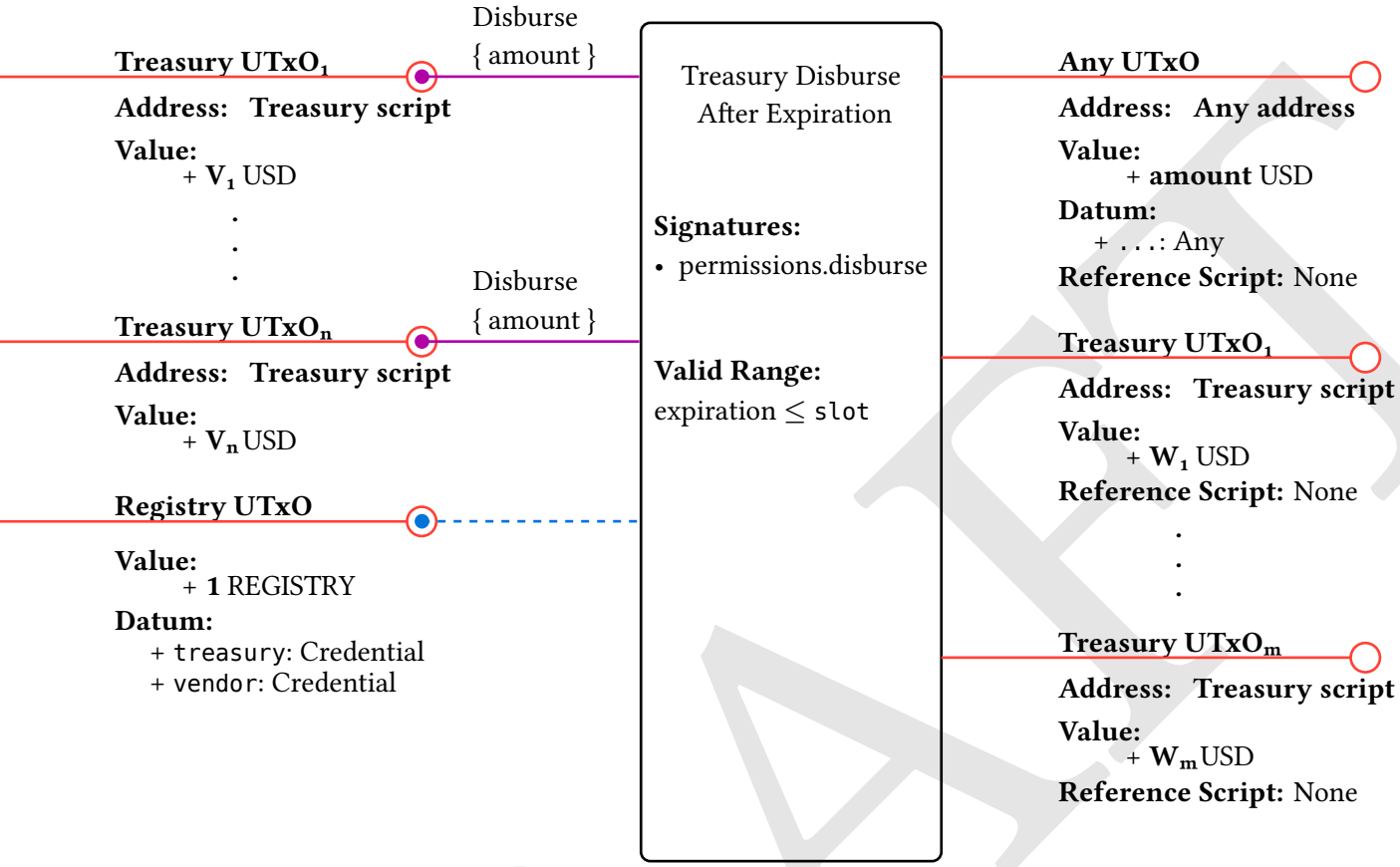
Note:

$$\text{amount[ADA]} \leq N_1 + \ldots + N_n$$
$$M_1 + \ldots + M_m = N_1 + \ldots + N_n - \text{amount[ADA]}$$
$$\text{amount[USD]} \leq V_1 + \ldots + V_n$$
$$W_1 + \ldots + W_m = V_1 + \ldots + V_n - \text{amount[USD]}$$

Figure 4: Treasury Disburse Before Expiration transaction

2.c.a.e - Disburse After Expiration

When the expiration time has passed, only native assets can be disbursed. This is to prevent USDM, for example, getting permanently locked.



Note:
amount ≤ V_1 + ... + V_n
W_1 + ... + W_m = V_1 + ... + V_n - amount

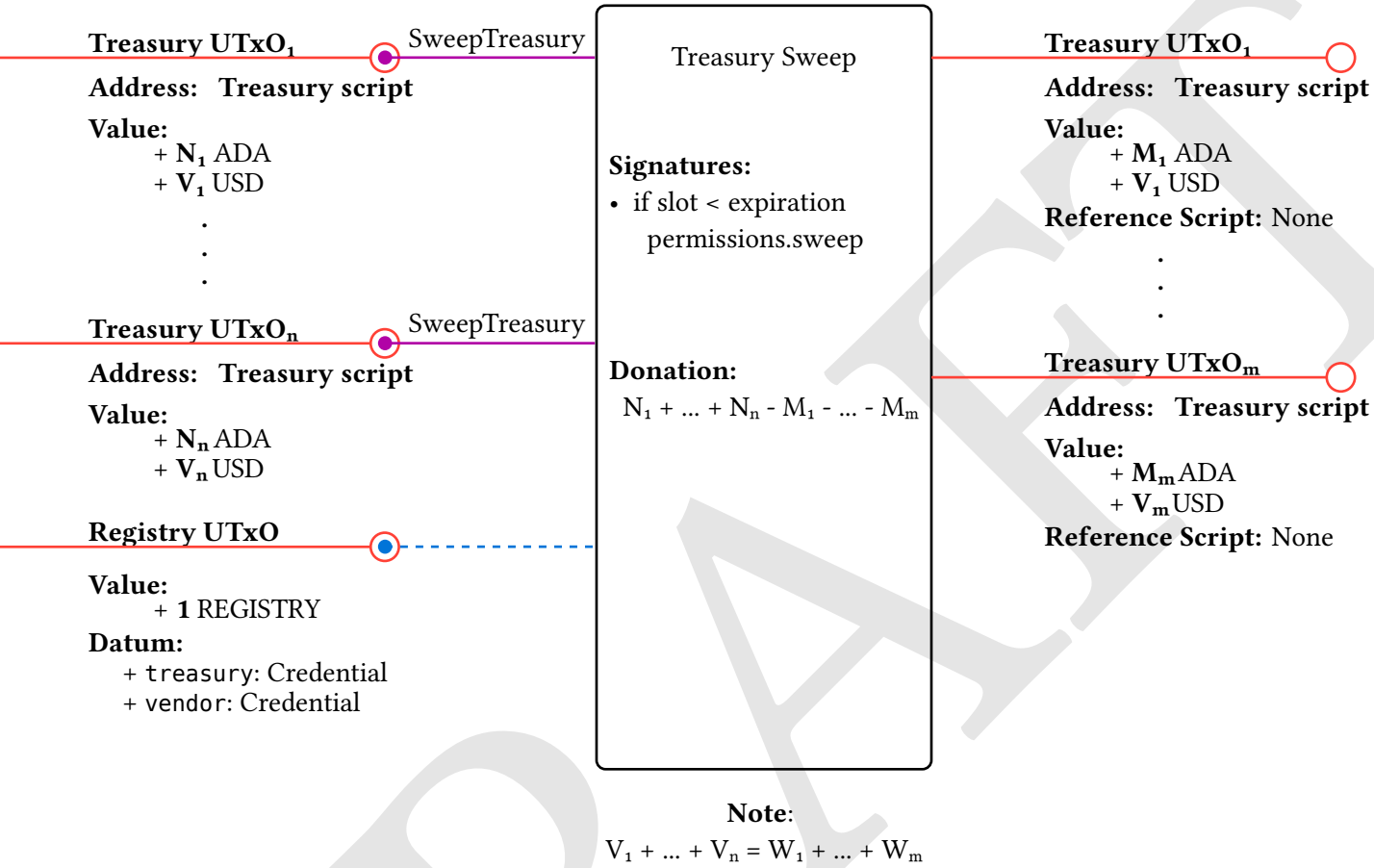
Figure 5: Treasury Disburse After Expiration transaction

2.c.a.f - Sweep

The funds can be swept back to the Cardano treasury by anyone after the expiration time has passed, or with permissions from the permissioned committee early.

Involved redeemers:

- `Sweep` redeemer, `Spend` purpose: for spending each of the Treasury inputs.



Note:
$V_1 + ... + V_n = W_1 + ... + W_m$

Figure 6: Treasury Sweep transaction

2.c.b - Vendor

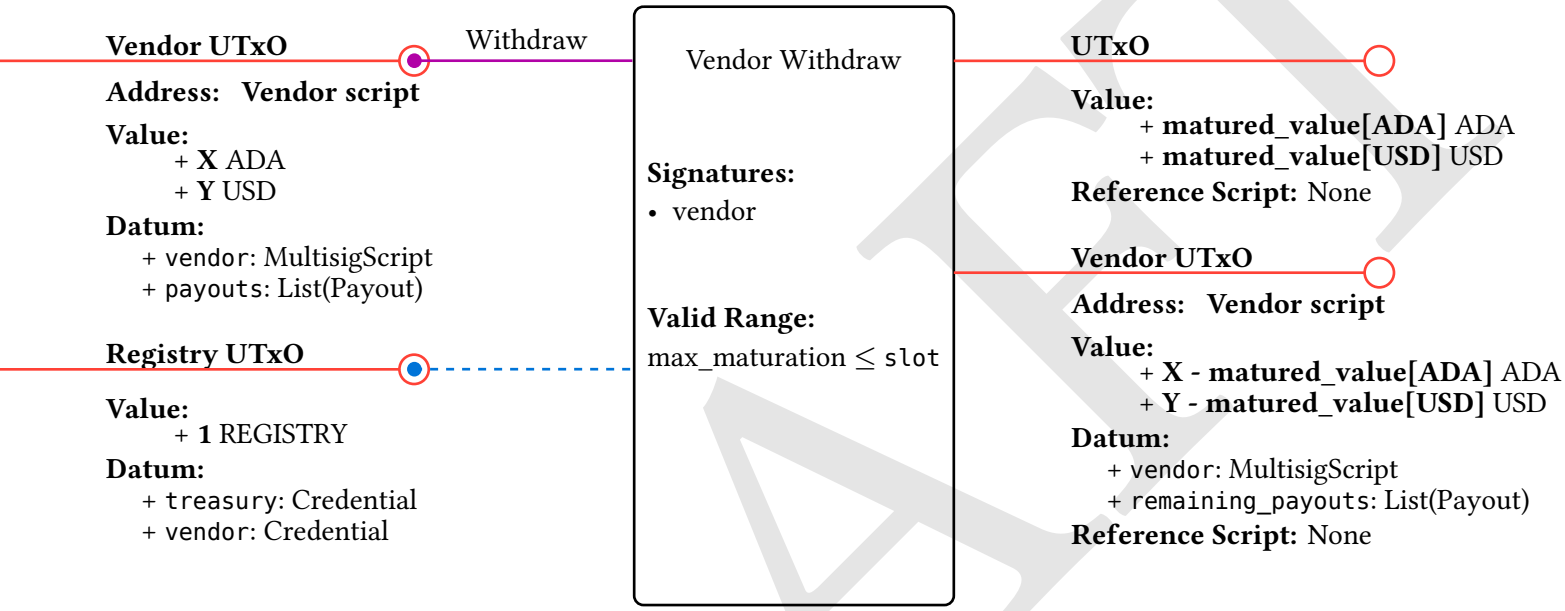
The Vendor UTxO value could hold any assets, not just ADA or stablecoins. We'll show only ADA and USD in the diagrams for simplicity only.

2.c.b.a - Withdraw

The vendor can withdraw funds from its owned UTxOs if those funds are *matured*, which means that their associated maturity time has passed, and the payouts status is active.

Involved redeemers:

- `Withdraw` redeemer, `Spend` purpose: for spending the Vendor UTxO.



Note:
payouts = expected_payouts + remaining_payouts
expected_payouts: active payouts where the maturation time is before the current slot
matured_value = sum of all matured payouts values
max_maturation = max(expected_payouts[0].maturation, ..., expected_payouts[m-1].maturation)

Figure 7: Vendor Withdraw transaction

2.c.b.b - Adjudicate

The permissioned committee can pause or resume a specific payout. This means modifying one or more of the payouts statuses encoded in the Vendor UTxO datum. When at least one payout flips from active to paused, the `pause` permission must be satisfied. When at least one payout flips from paused to active, the `resume` permission must be satisfied.

Involved redeemers:

- `Adjudicate { statuses }` redeemer, `Spend` purpose: for spending the Vendor UTxO.

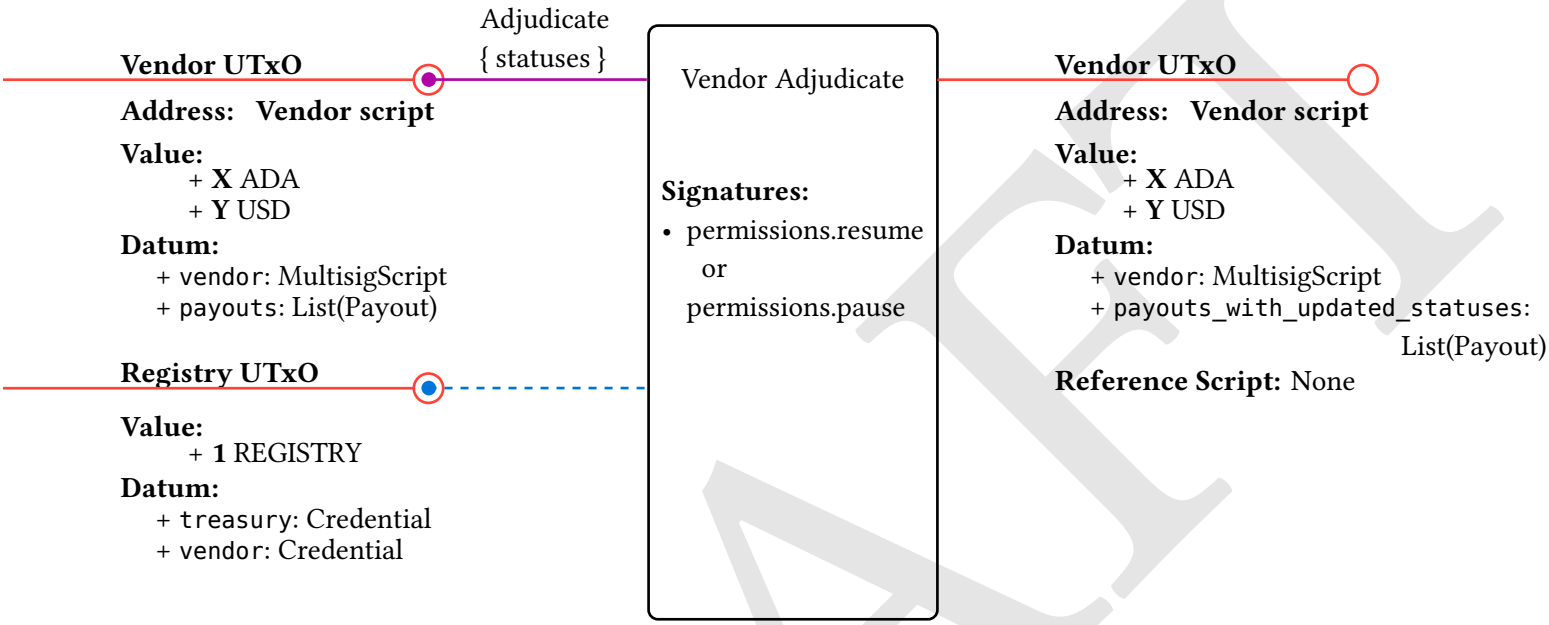


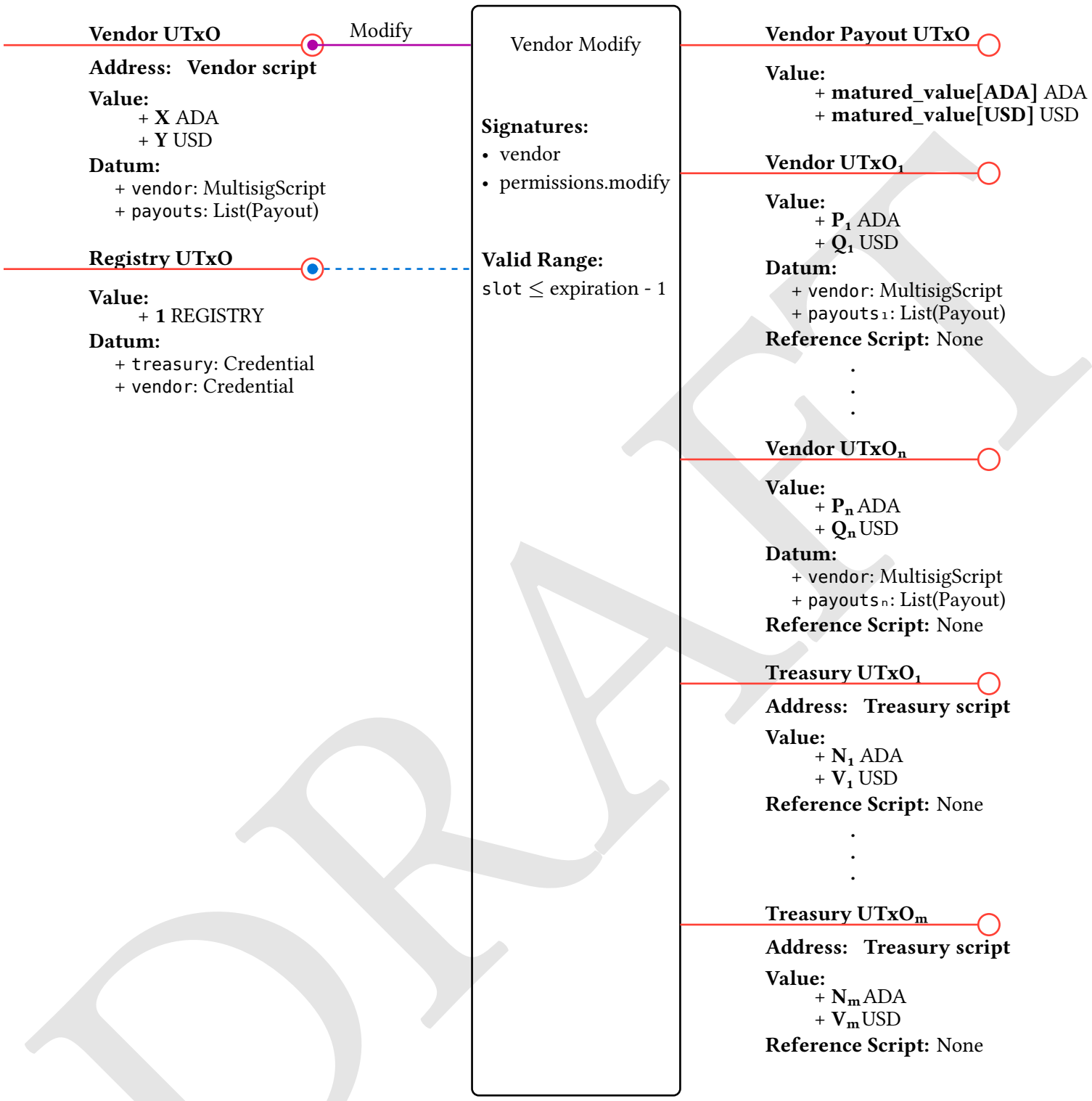
Figure 8: Vendor Adjudicate transaction

2.c.b.c - Modify

Both the vendor and the permissioned committee agree on modifying or canceling the project. The provided Vendor input must keep matured funds, whilst the rest of the funds are split between new treasury and vendor UTxOs.

Involved redeemers:

- `Modify` redeemer, `Spend` purpose: for spending the Vendor UTxO.



Note:

Vendor input value = matured_value + unmatured_value

unmatured_value = $P_1 + \ldots + P_n + Q_1 + \ldots + Q_n + V_1 + \ldots + V_n + N_1 + \ldots + N_n$

$\forall i \in \{1..n\}$ $payout_i$: current slot < $payout_i$.maturation $\vee$ $payout_i$.status = Pause

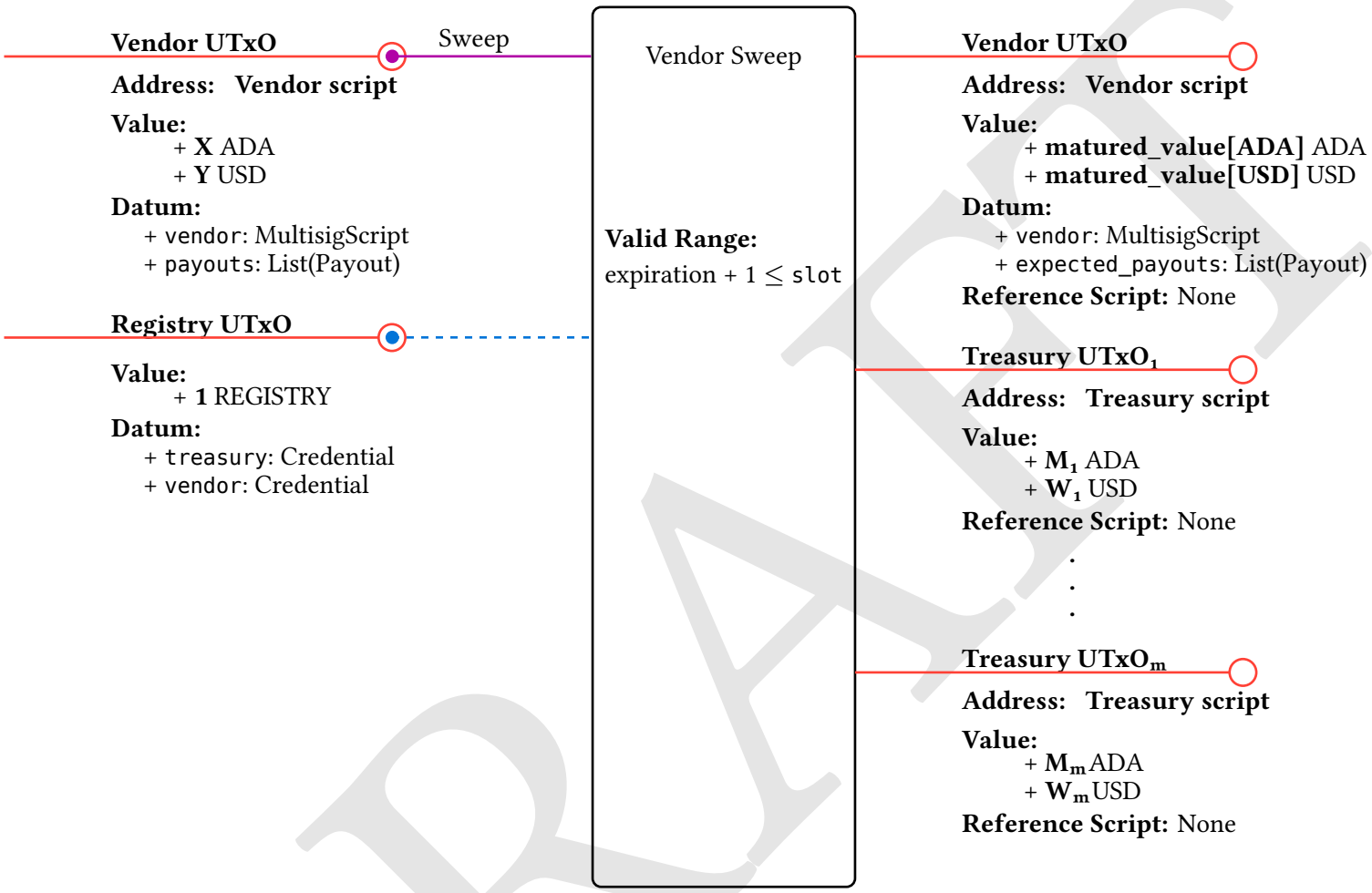
Figure 9: Vendor Modify transaction

2.c.b.d - Sweep

After the expiration time has passed, all paused, unmatured funds can be swept back to the treasury contract.

Involved redeemers:

- `SweepVendor` redeemer, `Spend` purpose: for spending the Vendor UTxO.



Note:

swept_value = vendor input value - matured_value

swept_value[ADA] = M_1 + ... + M_m

swept_value[USD] = W_1 + ... + W_m

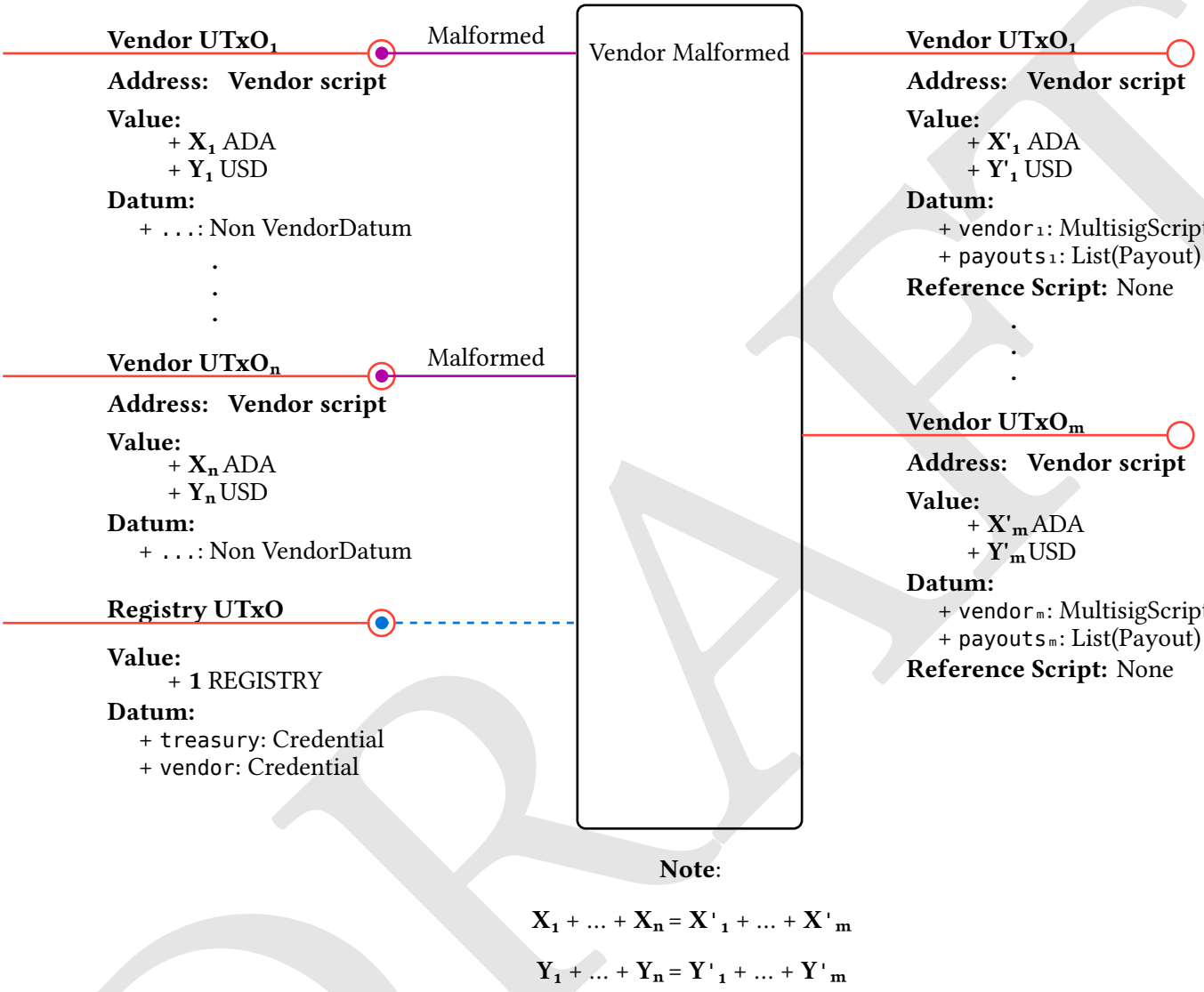
Figure 10: Vendor Sweep transaction

2.c.b.e - Malformed

In case the Vendor UTxO datum is malformed i.e. not a `VendorDatum`, the funds can be sent back to the treasury contract. This is to prevent the funds from being locked in a malformed UTxO.

Involved redeemers:

- `Malformed` redeemer, `Spend` purpose: for spending the Vendor UTxO.



Note:

$$X_1 + \ldots + X_n = X'_1 + \ldots + X'_m$$

$$Y_1 + \ldots + Y_n = Y'_1 + \ldots + Y'_m$$

Figure 11: Vendor Malformed transaction

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 `0fa83e95bb3703fd026c1d4ac6a8a0b4e1887d3d`.

Filename
validators/oneshot.ak
validators/treasury.ak
validators/vendor.ak
lib/types.ak
lib/utilities.ak
lib/logic/treasury/disburse.ak
lib/logic/treasury/fund.ak
lib/logic/treasury/reorganize.ak
lib/logic/treasury/sweep.ak
lib/logic/treasury/withdraw.ak
lib/logic/vendor/adjudicate.ak
lib/logic/vendor/malformed.ak
lib/logic/vendor/modify.ak
lib/logic/vendor/sweep.ak
lib/logic/vendor/withdraw.ak

4 - Findings

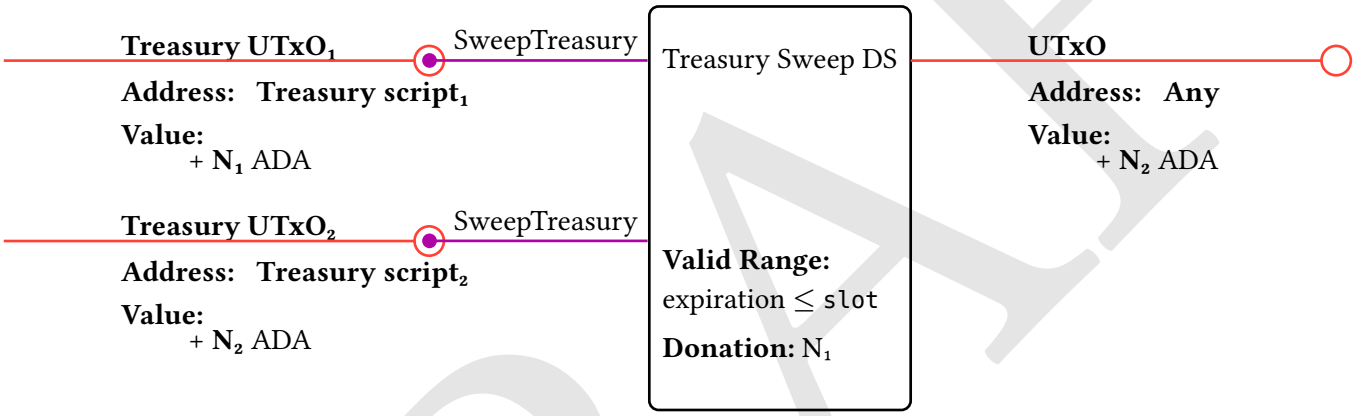
ID	Title	Severity	Status
TRS-001	Treasury Sweep and Vendor Malformed Double Satisfaction	Critical	Resolved
TRS-002	Treasury script withdrawal Double Satisfaction	Critical	Resolved
TRS-003	False publish purpose renders scripts unusable	Critical	Resolved
TRS-101	Treasury Fund: Vendor UTxOs can be created with insufficient funds	Major	Resolved
TRS-102	Matured payouts can be passed as not matured by modifying the validity range	Major	Resolved
TRS-103	Vendor Adjudicate: vulnerability allows bypassing payout status checks	Major	Resolved
TRS-104	Treasury Fund DS attack vector	Major	Resolved
TRS-105	Funds from malformed vendor UTxOs can be stolen	Major	Resolved
TRS-201	Vendor Sweep: incompatible with Treasury `Reorganize`	Minor	Resolved
TRS-202	Vendor Sweep: vendor output stake credential not checked	Minor	Resolved
TRS-203	Treasury Sweep DDOS	Minor	Resolved
TRS-204	Prevent inclusion of reference scripts	Minor	Resolved

5 - TRS-001 Treasury Sweep and Vendor Malformed Double Satisfaction

Category	Commit	Severity	Status
Vulnerability	85f525b56a8c73579a139c9a3dcb384a2db5d1c6	Critical	Resolved

5.a - Description

It is possible to steal funds from Treasury UTxOs when sweeping after expiration. This can be done by satisfying the `SweepTreasury` redeemer using a donation from a single UTxO that is sufficient to validate other Treasury UTxOs for other Treasury script instances. The attack is illustrated in the following transaction example:



Note: $N_1 \geq N_2$

In the case of `Malformed`, the attack vector is similar: two different Vendor scripts validations related to the same Treasury script can be satisfied by the same Treasury outputs, allowing to steal funds from some malformed vendor inputs.

5.b - Recommendation

Forbid inputs that do not originate from Treasury and Vendor scripts registered in the Registry in both the Vendor and Treasury validators.

5.c - Resolution

Resolved in commit `af164f6`.

6 - TRS-002 Treasury script withdrawal Double Satisfaction

Category	Commit	Severity	Status
Vulnerability	85f525b56a8c73579a139c9a3dcb384a2db5d1c6	Critical	Resolved

6.a - Description

By producing Treasury outputs with the Treasury script withdrawal, we can satisfy Treasury outputs related checks on some Vendor redeemers. For example:

- `Malformed`: Vendor malformed UTxOs funds go to treasury script, say X ADA, and Treasury script Withdrawal of X ADA, then only X ADA enforced to Treasury outputs, so X ADA could be stolen. Valid at any point in time.
- `SweepVendor`: Sweep X ADA of a Vendor input, and Withdrawal of amount X, SweepVendor of X ADA, then only X ADA enforced to Treasury outputs, so X ADA could be stolen. Valid only after expiration.
- `Modify`: similar to the above ones, but requires more signatures and is valid before expiration only.

6.b - Recommendation

Forbid script inputs when doing the Treasury script withdrawal.

6.c - Resolution

Resolved in commit `f2a3a6a`.

7 - TRS-003 False publish purpose renders scripts unusable

Category	Commit	Severity	Status
Bug	85f525b56a8c73579a139c9a3dcb384a2db5d1c6	Critical	Resolved

7.a - Description

The current implementation of the publish purpose is set to False. This is done to prevent actions like delegating to a pool or deregistering the script stake credential. But it also prevents two actions that are critical to the usage of the protocol.

- Credential Registration: When funds are withdrawn from the treasury, they are added to the reward address of the script. From there, they must be withdrawn, executing the withdraw purpose. Before this is done, the script must be registered. This is done using one of multiple certificates. The legacy `stake_registration` certificate, doesn't require any validation to run, but that certificate will be deprecated in the near future. The rest of the registration certificates, `reg_cert`, `stake_reg_deleg_cert`, `vote_reg_deleg_cert` and `stake_vote_reg_deleg_cert` require the validation of the script to run with the publish purpose.

This means that while it is currently possible to register and withdraw from the treasury script, this will be invalidated when the legacy certificate is dropped.

- Drep Delegation: Another action that is forbidden by the current implementation is delegating to a dRep. This is an issue because, according to Article IV, Section 5 of the cardano constitution, funds that are withdrawn from the treasury and held by administrators must be delegated to the Always Abstain dRep.

We acknowledge Matthias "KtorZ" Benkort for helping with this finding.

7.b - Recommendation

Update the publish validation so it allows registration as well as delegation only to the "always abstain" dRep.

7.c - Resolution

Resolved in commit `11534a2`.

8 - TRS-101 Treasury Fund: Vendor UTxOs can be created with insufficient funds

Category	Commit	Severity	Status
Bug	85f525b56a8c73579a139c9a3dcb384a2db5d1c6	Major	Resolved

8.a - Description

It is possible to create Vendor UTxOs with insufficient funds to cover the amounts specified in their datum. The Treasury `Fund` redeemer checks that the amount taken from Treasury UTxOs goes to Vendor UTxOs by checking:

$$\sum \text{payouts values} = \text{amount}$$

and

$$\sum \text{vendor outputs values} \geq \text{amount}.$$

However, it does not ensure that for each Vendor UTxO, it holds that

$$\sum \text{output datum payouts} \leq \text{output value}.$$

This is problematic because there's no way to add missing funds with any of the Vendor redeemers.

8.b - Recommendation

For each created Vendor UTxO ensure that

$$\sum \text{output datum payouts} \leq \text{output value}.$$

8.c - Resolution

Resolved in commit `9c13ade`.

9 - TRS-102 Matured payouts can be passed as not matured by modifying the validity range

Category	Commit	Severity	Status
Bug	85f525b56a8c73579a139c9a3dcb384a2db5d1c6	Major	Resolved

9.a - Description

During multiple vendor operations, payouts are separated into matured and unmatured. Matured payouts that are active are considered to be funds that already belong to the vendor, so they must stay active and in the Vendor contract until the vendor performs a Withdraw operation.

To check which payouts are already matured, the following check is used:

```
is_entirely_after(validity_range, ip.maturation)
```

But the validity range can be manipulated so the lower bound is smaller than the maturation time, regardless of the current time and the upper bound.

Depending on the operation, this can have various consequences:

- `Adjudicate`: In this operation it is possible for matured, active payouts to be switched to paused, with the only required permissions being `config.permissions.pause`.
- `Modify`: In this operation matured funds could be sent back to the treasury or distributed into a new payout structure. Although, this operation requires the vendor signature, so they would need to also approve of this redistribution of matured funds.
- `Withdraw`: In this operation the vendor could claim less funds that what belong to them.

9.b - Recommendation

Add a limit to the validity interval length, to prevent the lower bound to be moved too far into the past.

9.c - Resolution

Resolved in commit `c757af0`.

A limit of 36 hours was imposed in the adjudicate operation. For modify and withdraw operations, seeing that both require the vendor signature, and the matured funds belong to them, they were left with no modifications.

10 - TRS-103 Vendor Adjudicate: vulnerability allows bypassing payout status checks

Category	Commit	Severity	Status
Vulnerability	d8fa5e85e5d5d2ca9dd608ad61589a4d6be59fad	Major	Resolved

10.a - Description

In the Vendor Adjudicate logic, statuses provided by the redeemer can be fewer than the payouts in the datum, bypassing validation checks for some payouts. This is due to the behavior of `list.zip` which silently drops elements from the longer list when lists have unequal lengths.

The vulnerability exists in the following code:

```
let (pause_permission_needed, resume_permission_needed) =
  input_vendor_datum.payouts
    |> list.zip(output_vendor_datum.payouts)
    |> list.zip(statuses) // vulnerable line
    |> list.foldl(...)
```

An attacker could provide a `statuses` list shorter than the number of payouts, which would result in only a subset of payouts being checked. This allows malicious actors to modify payout datums fields that should remain equal.

10.b - Recommendation

Add a validation check to ensure the `statuses` list has the same length as the list of payouts.

10.c - Resolution

Resolved in commit `c757af0`.

11 - TRS-104 Treasury Fund DS attack vector

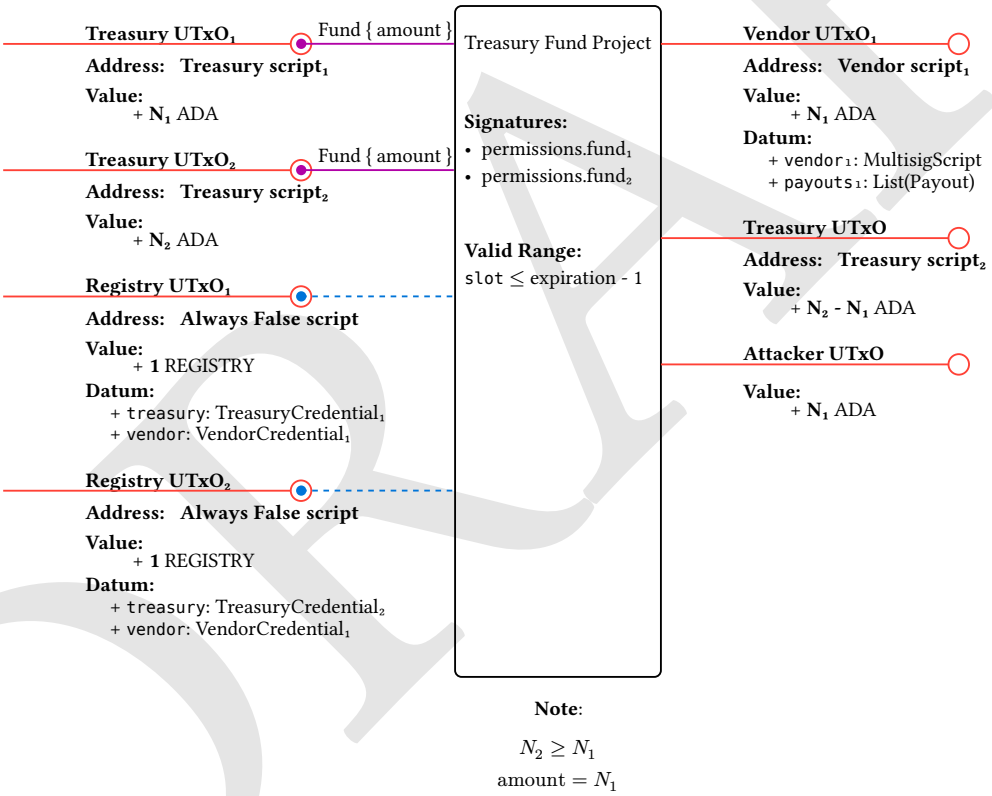
Category	Commit	Severity	Status
Vulnerability	d8fa5e85e5d5d2ca9dd608ad61589a4d6be59fad	Major	Resolved

11.a - Description

During the Treasury Fund operation, funds are moved from the treasury script to the vendor script.

It is possible to craft a double satisfaction attack by including inputs from two different treasury scripts. These scripts will each have their own registry token, but both registry UTxOs should have the same vendor credential for the attack to work. Then, both treasury scripts will validate against the same vendor UTxOs, leaving funds free to be stolen.

Example transaction:



11.b - Recommendation

We recommend checking that there are no inputs from scripts other than the treasury and vendor scripts,

11.c - Resolution

Resolved in commit `af164f6`.

12 - TRS-105 Funds from malformed vendor UTxOs can be stolen

Category	Commit	Severity	Status
Vulnerability	d8fa5e85e5d5d2ca9dd608ad61589a4d6be59fad	Major	Resolved

12.a - Description

Malformed vendor UTxOs are UTxOs with an invalid format datum. To prevent funds from being locked, these UTxOs can be consumed, only if all funds are sent to the treasury.

But the validation is not looking for treasury inputs. So it is possible to combine a Vendor Malformed operation with some treasury operation like reorganize or sweep, creating a double satisfaction attack on the treasury output and stealing some funds.

12.b - Recommendation

We recommend checking that there are no treasury inputs

12.c - Resolution

Resolved in commit `867d5f5`.

13 - TRS-201 Vendor Sweep: incompatible with Treasury `Reorganize`

Category	Commit	Severity	Status
Bug	85f525b56a8c73579a139c9a3dcb384a2db5d1c6	Minor	Resolved

13.a - Description

The `SweepVendor` redeemer allows spending Treasury UTxOs, in theory with `Reorganize` redeemer, but this is incompatible with the protocol's design:

- `SweepVendor` can only be used after expiration
- `Reorganize` can only be used before expiration

13.b - Recommendation

Either remove the ability to spend Treasury UTxOs in `SweepVendor` transactions, or add a new redeemer for this use case.

13.c - Resolution

Resolved in commit `a4d83ec`.

14 - TRS-202 Vendor Sweep: vendor output stake credential not checked

Category	Commit	Severity	Status
Robustness	85f525b56a8c73579a139c9a3dcb384a2db5d1c6	Minor	Resolved

14.a - Description

The `SweepVendor` redeemer does not check that the vendor output stake credential is None if there are still not withdrawn funds.

14.b - Recommendation

Check that the stake credential is `None`.

14.c - Resolution

Resolved in commit `3659533`.

15 - TRS-203 Treasury Sweep DDOS

Category	Commit	Severity	Status
Robustness	d8fa5e85e5d5d2ca9dd608ad61589a4d6be59fad	Minor	Resolved

15.a - Description

When the expiry date of a treasury is reached, anyone can use the Treasury Sweep operation to donate the remaining ADAs to the cardano treasury. The validation doesn't force users to donate all the ADAs, because there might be some native tokens left in the outputs, so the minADA needs to be covered.

But this leniency allows any quantity to be donated, even 1 lovelace. With this in mind, an attacker could keep building transactions donating the minimum amount, and locking the rest of the ADAs and, potentially, USDM and other tokens.

15.b - Recommendation

We recommend updating the validation so most of the ADAs need to be donated. One possible solution would be to have a hardcoded maximum amount of lovelace that can be in the treasury outputs that can definitely cover any minADA requirements. Something like 10 or 20 ADAs would be enough. We also recommend limiting the amount of treasury outputs to only one, to prevent other spam and ddos attacks.

15.c - Resolution

Resolved in commit `eddd236`.

16 - TRS-204 Prevent inclusion of reference scripts

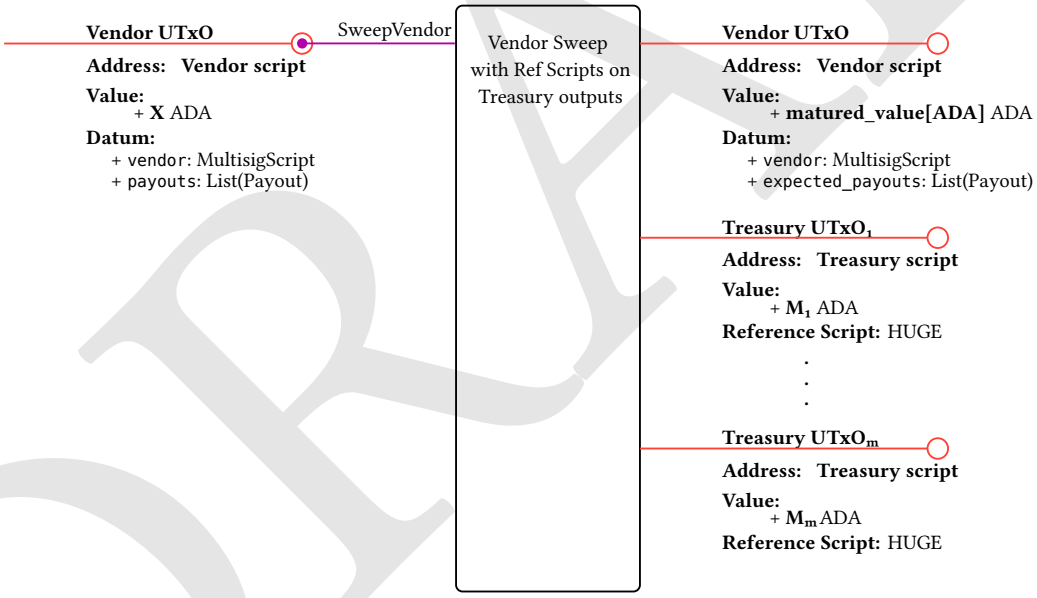
Category	Commit	Severity	Status
Robustness	d8fa5e85e5d5d2ca9dd608ad61589a4d6be59fad	Minor	Resolved

16.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.

In this protocol, a attacker could exploit this on `SweepVendor` and `Malformed` (both unpermissioned Vendor redeemers) by including a reference script in the Treasury outputs of those transactions, costing more fees to the next party interacting with them.

A concrete example of this attack could be



Each M_i is the minimum ADA value to comply min ADA for i-th Treasury output. Allows dividing to the max number of Treasury UTxOs.

Then those Treasury UTxOs would be costly to consume with `SweepTreasury`.

16.b - Recommendation

At least check that no reference scripts are attached to Treasury outputs on `SweepVendor` and `Malformed`. This could be applied to the other redeemers as well, unless we trust on permissioned participants to not include reference scripts in their transactions.

16.c - Resolution

Resolved in commit `7653da0`.

A Appendix

A.1 Terms and Conditions of the Commercial Agreement

A.1.1 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.

A.1.2 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 IOG 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.

A.1.3 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: `validators/oneshot.ak` **Hash**: `486b661193aa7d84e9e2cd8f284ee5e3b79b89dd458c1cbea0c773260f55bc03`
Filename: `validators/treasury.ak` **Hash**: `b0130d0276ebf5efdfaa5dbc0d38e23e00657ccacc660b0999db0185b1fdee11`
Filename: `validators/vendor.ak` **Hash**: `cfb359b9c823b993148ec51031591f287398e3ce3d3b81f1b3f87bb7f67cda2c`
Filename: `lib/types.ak` **Hash**: `4b5517f4cd9fcd75b545342f43ed610ebe8969b8ade0b7afb08ee6981363f41e`
Filename: `lib/utilities.ak` **Hash**: `4697438429cd21d6cbd0315a26dd3022f2fbd01136a8bb0f3925675124973995`
Filename: `lib/logic/treasury/disburse.ak` **Hash**: `036c02aa5c0f287dea4ff9703b09906ff44467a9fbac78671a08b7b89bd2875c`
Filename: `lib/logic/treasury/fund.ak` **Hash**: `75d4bd6734e18b207c08e43a89943781d350ba9998868a6808b0e1b218849a3c`
Filename: `lib/logic/treasury/reorganize.ak` **Hash**: `5713f10f8f101a21c07cd9245312e5ef21dfdef8dcf9cce515c252227fce93ee`
Filename: `lib/logic/treasury/sweep.ak` **Hash**: `a6d5282818ae8f8b2b62834afadbd4911fdd24cb4d3219a3404ecebef50112ab`
Filename: `lib/logic/treasury/withdraw.ak` **Hash**: `a702bb19a814c376a0071ee15bcd4681ceaccaf0563e67745e9078b5e18a55d0`
Filename: `lib/logic/vendor/adjudicate.ak` **Hash**: `78ddcc86593503bde90911b5497ff14e3a3347fc2d6ffd8bb5a9b9136f26852f`
Filename: `lib/logic/vendor/malformed.ak` **Hash**: `0b03157fb8f01a80f34bb49d57ac74f4bab71ae179a3977c24b459d47bfd7041`
Filename: `lib/logic/vendor/modify.ak` **Hash**: `9c58a16ae2fce0aad52e64b03b60c5d78a8f03e5ab9cca3761ea2c64eee735c2`
Filename: `lib/logic/vendor/sweep.ak` **Hash**: `57e1a2763c3b497e598d5375814425bc6b68fa74c6619a5a103652963b3d35dc`
Filename: `lib/logic/vendor/withdraw.ak` **Hash**: `21d5021638f0e459bd718d6f453c4c6e0141957ca951596d2f5191efee4a3e2c`

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.

A.2 Issue Guide

A.2.1 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.

A.2.2 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.

A.3 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.

A.4 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.

A.4.1 Links

- Website
- Email
- Twitter