Audit Report

2025-08-05

Pragma - Amaru Treasury Contracts Audit

Contents

1 - Summary

This report provides a comprehensive audit of the Amaru Treasury contracts, an extension of the SundaeSwap Treasury contracts which provide the Amaru maintainer committee a set of tools to manage its budget. These contracts ensure that fund expenditures are done in accordance with the scope defined in the budget and authorized by the relevant scope owners.

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 Amaru Treasury Contracts are comprised of 4 validators:
- Scopes validator: for creating the Scopes UTxO, which is used to hold the scope owners of the protocol. Shared across all treasuries.
- Permissions validator: used as a companion script for the Treasury script. It defines the ownership rules in addition to the logic corresponding to the sweep, reorganize, disburse and fund actions.
- Amaru Treasury Registry validator: for creating the Treasury Registry UTxO. Needed by the Treasury script. One per scope, since each scope has its own associated Treasury script.
- Treasury validator: holds the funds available to a specific scope. From Treasury Contracts.

The protocol has 3 kinds of UTxOs:
- Scopes UTxO: holds the scope owners in its datum. Each scope owner is identified by a multisignature.
- Treasury Registry UTxOs: holds the script hash of the related Treasury script. One per scope.
- Treasury UTxOs: holds the funds available to a specific scope.

The main UTxO of the protocol is the Scopes UTxO, which defines all the scope owners of the protocol, each of them identified by a multisignature. This UTxO is shared across all treasuries.

Each scope is related to a single Treasury, Permissions, and Treasury Registry script.
- The Treasury script is used to hold the funds available to the scope.
- The Permissions script is a withdraw script used to define the ownership rules in addition to the logic corresponding to the sweep, reorganize, disburse and fund actions over the Treasury script.
- The Treasury Registry script is used to create the Treasury Registry UTxO, needed by the Treasury script.

The Amaru contributors are paid by disbursing funds from the Treasury UTxO. There's no use for the Vendor script machinery in this protocol.

1.b - Process

Our audit process involved a thorough examination of Amaru 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 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 Amaru team through Discord. Diagrams illustrating the necessary transaction structure for proper interaction with the protocol are attached as part of this report. The Amaru 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 - Scopes UTxO

The Scopes UTxO holds an NFT that identifies it and contains in its datum the owners (multisigs) of each scope.

- Address:
  - Payment part: Scopes validator script hash. Parameters:
    - `seed`: seed used to mint the NFT
  - Staking part: any
- Value:
  - 1 amaru scopes
- Datum: `Scopes`
  - ledger: MultisigScript
  - consensus: MultisigScript
  - mercenaries: MultisigScript
  - marketing: MultisigScript

2.a.b - Amaru Treasury Registry UTxO

The Amaru Treasury Registry UTxO is identified by an NFT and contains in its datum the related Treasury script hash.

- Address:
  - Payment part: Amaru Treasury Registry validator script hash. Parameters:
    - `seed`: seed used to mint the NFT
    - `scope`: allows defining one policy per scope if needed, but the scope value is not used by the validator to perform any checks.
  - Staking part: any
- Value:
  - 1 REGISTRY
- Datum: `ScriptHashRegistry as Registry`
  - `treasury`: Treasury script hash
  - `vendor`: Vendor script hash (unused)

2.b - Assets

2.b.a - Scopes NFT

Minted by a one-shot minting policy. Locked into the Scopes UTxO.

- Policy ID: hash of Scopes validator script hash.
- Token name: `amaru scopes`

2.b.b - Amaru Treasury Registry NFT

Minted by a one-shot minting policy. Locked into the Registry UTxO.

- Policy ID: hash of Amaru Treasury Registry validator script hash.
- Token name: `REGISTRY`

2.b.c - Treasury Assets

External assets used for paying to Amaru contributors throughout the Treasury. Any assets can be used, but 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.

2.c - Transactions

2.c.a - Scopes

The Scopes UTxO holds an NFT that identifies it and contains in its datum the different scope owners (identified as multisigs):

- Ledger
- Consensus
- Mercenaries
- Marketing
- Contingency: not stored in the datum because it is the union of the other 4.

2.c.a.a - Create Scopes

The Scopes NFT is minted and paid to the Scopes output, which contains the scopes owners in its datum.

Involved redeemers:

- `Data` redeemer, `Mint` purpose: for minting the amaru scopes NFT.

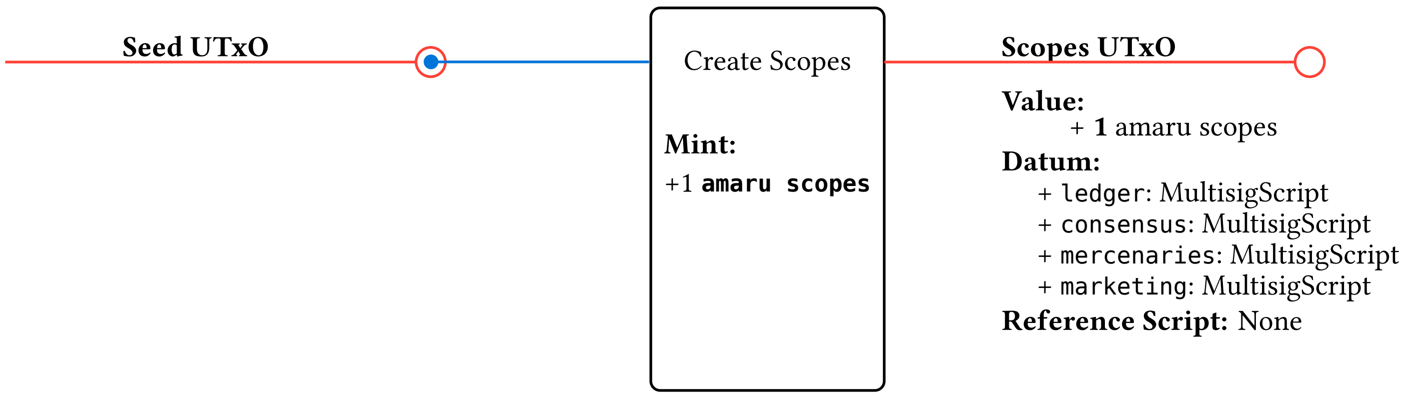


Figure 1: Create Scopes transaction

2.c.a.b - Update Scopes

The Scopes datum is updated by spending the Scopes UTxO. The General Assembly must sign the transaction, and the scopes NFT must be paid to the Scopes output.

Involved redeemers:

- `Data` redeemer, `Spend` purpose: for spending the Scopes UTxO.

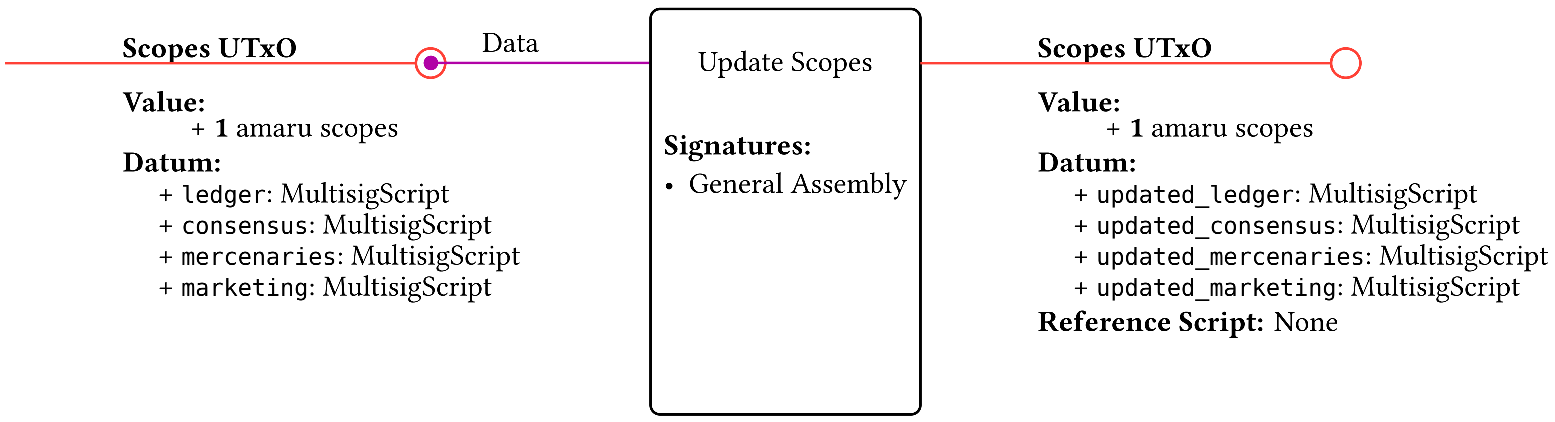

Scopes UTxO
Data
Value:
+ 1 amaru scopes
Datum:
+ ledger: MultisigScript
+ consensus: MultisigScript
+ mercenaries: MultisigScript
+ marketing: MultisigScript
Update Scopes
Signatures:
• General Assembly
Scopes UTxO
Value:
+ 1 amaru scopes
Datum:
+ updated_ledger: MultisigScript
+ updated_consensus: MultisigScript
+ updated_mercenaries: MultisigScript
+ updated_marketing: MultisigScript
Reference Script: None

Figure 2: Update Scopes transaction

2.c.a.c - Delete Scopes

The Scopes UTxO is deleted by spending the Scopes UTxO and burning the Scopes NFT. The General Assembly must sign the transaction, and it can only be done after the expiration date defined in the Amaru Contracts configuration.

Involved redeemers:

- `Data` redeemer, `Spend` purpose: for spending the Scopes UTxO.
- `Data` redeemer, `Mint` purpose: for burning the amaru scopes NFT.

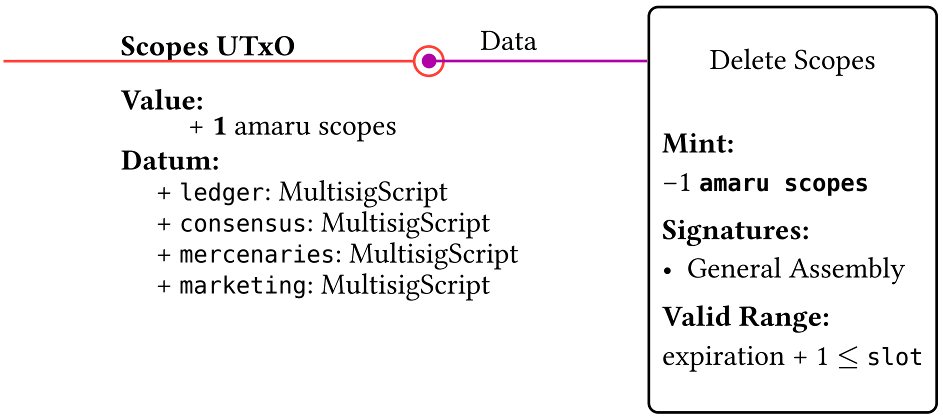


Figure 3: Delete Scopes transaction

2.c.b - Permissions

The Amaru permissions validator is a generic scope owner stake validator meant to be zero-withdrawn to approve actions over the related Treasury script.

Depending on the specific action over the Treasury, the approval must be done by:

- `Reorganize`: approval of only the scope owner suffices.
- `Disburse`: must be approved by the scope owner and *at least one* other scope owner.
- `Sweep`: same as `Disburse`.

If the scope owner of the permissions script is `Contingency`, the approval must be done by all the scopes.

Since the Amaru Contracts do not use the Vendor script machinery, Treasury `Fund` redeemer is not contemplated.

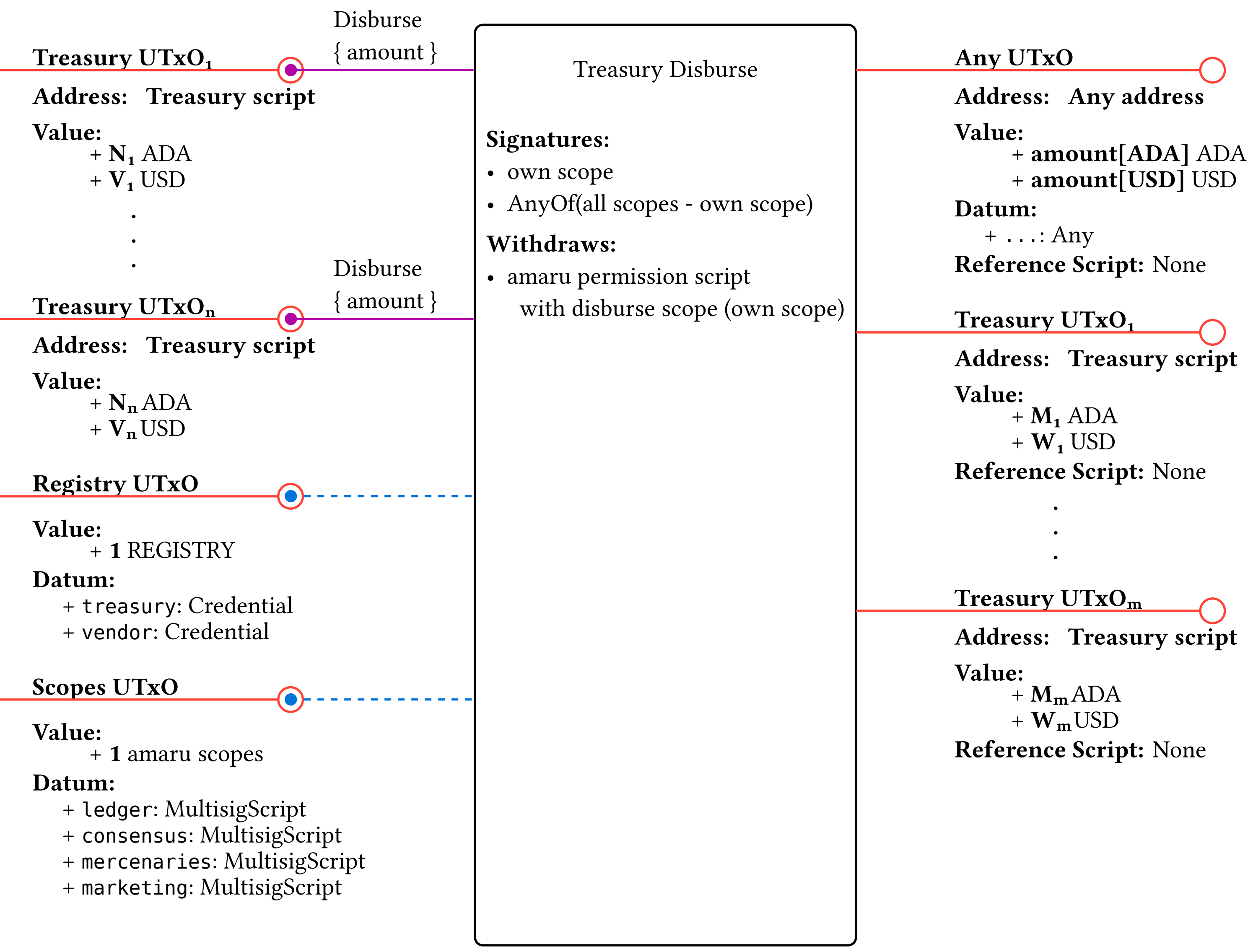
2.c.b.a - Amaru Treasury Disburse

Used to pay Amaru contributors. Must be approved by the scope owner of the Treasury and *at least one* other scope owner.

Involved redeemers:

- `Data` redeemer, `Withdraw` purpose: for zero-withdrawing Permissions script.
- `Disburse { amount }` redeemer, `Spend` purpose: for spending each of the Treasury inputs.

When the expiration time has passed, only native assets can be disbursed.



Note:

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

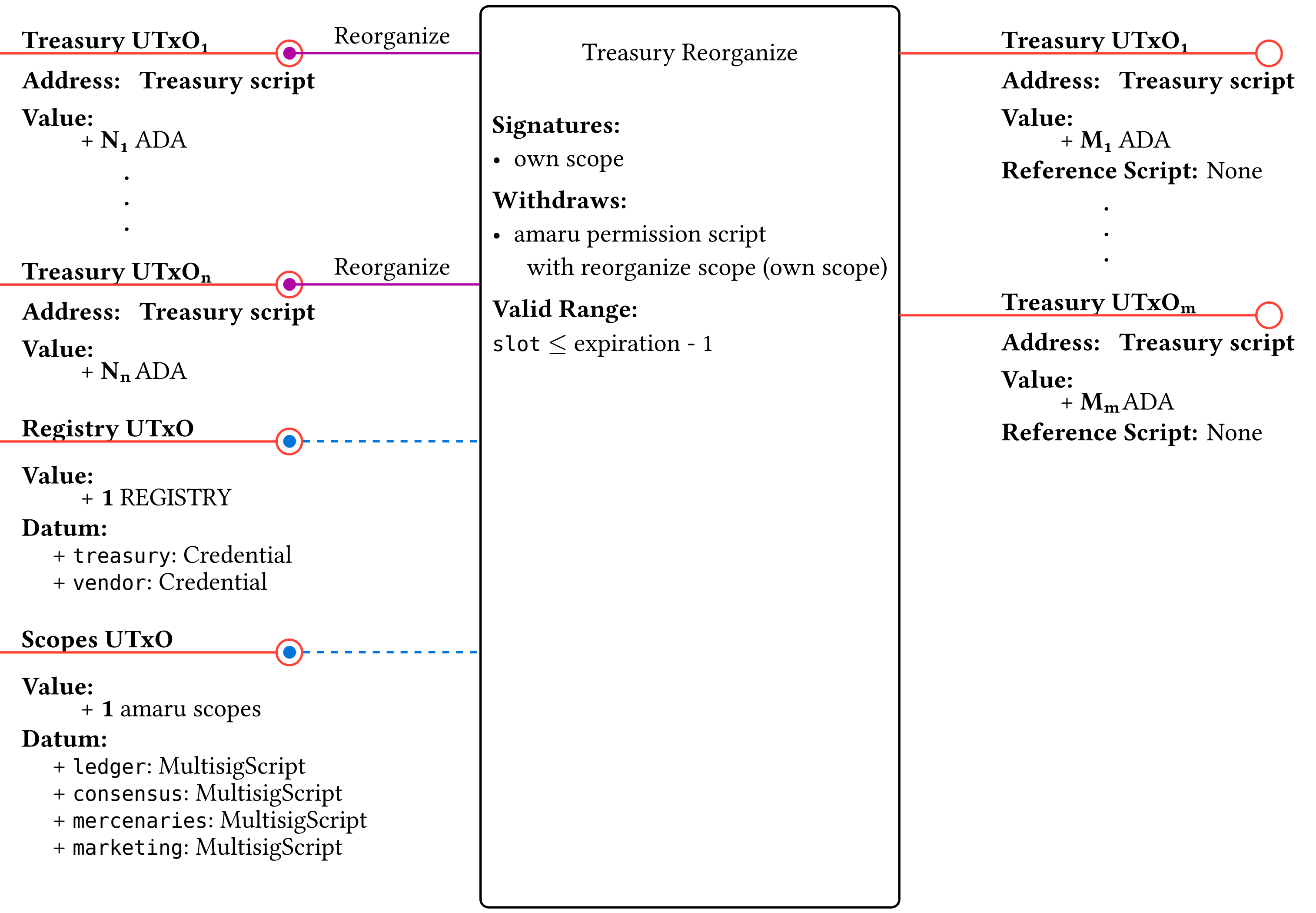
Figure 4: Treasury Disburse transaction

2.c.b.b - Treasury Reorganize

The Scope owner of the Treasury reorganizes the Treasury UTxOs. With its own approval suffices.

Involved redeemers:

- `Reorganize` redeemer, `Spend` purpose: for spending each of the Treasury inputs.
- `Data` redeemer, `Withdraw` purpose: for zero-withdrawing Amaru Permissions script.



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

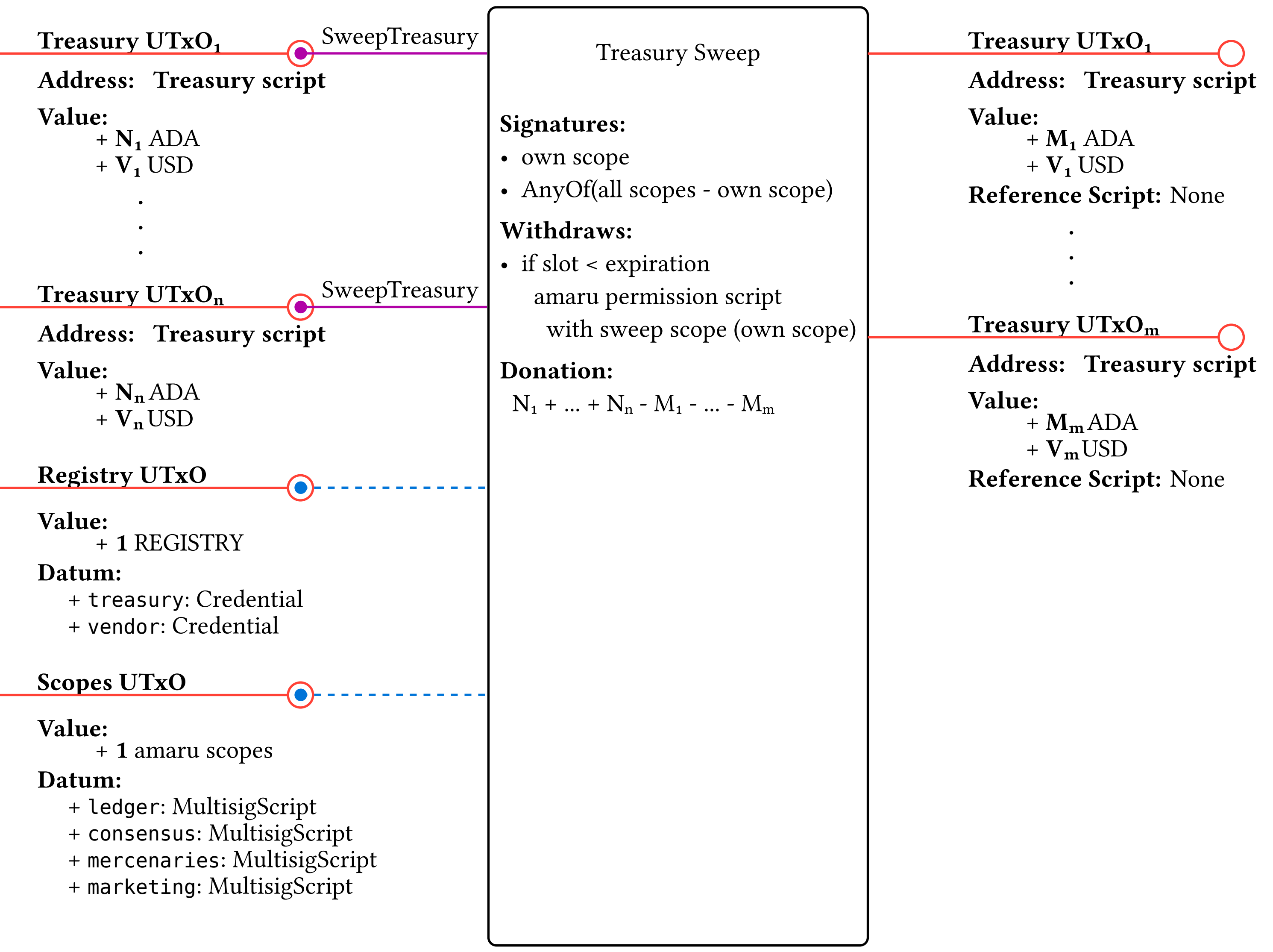
Figure 5: Reorganize Treasury transaction

2.c.b.c - Treasury Sweep

The funds are swept back to the Cardano treasury. If done before the expiration date, must be approved by the scope owner of the Treasury and *at least one* other scope owner. If done after the expiration date, anyone can sweep the funds.

Involved redeemers:

- `Sweep` redeemer, `Spend` purpose: for spending each of the Treasury inputs.
- `Data` redeemer, `Withdraw` purpose: for zero-withdrawing Amaru Permissions script.



Note:

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

Figure 6: Sweep Treasury transaction

2.c.c - Amaru Treasury Registry

The Amaru Treasury Registry is required by the Amaru Treasury script. Unlike Scopes UTxO datum, the Amaru Treasury Registry UTxO datum cannot be updated.

2.c.c.a - Create Registry

The Registry NFT is minted and paid to the Registry output, which contains the Treasury script hash in its datum.

Involved redeemers:
- `Data` redeemer, `Mint` purpose: for minting the Amaru Treasury Registry NFT.

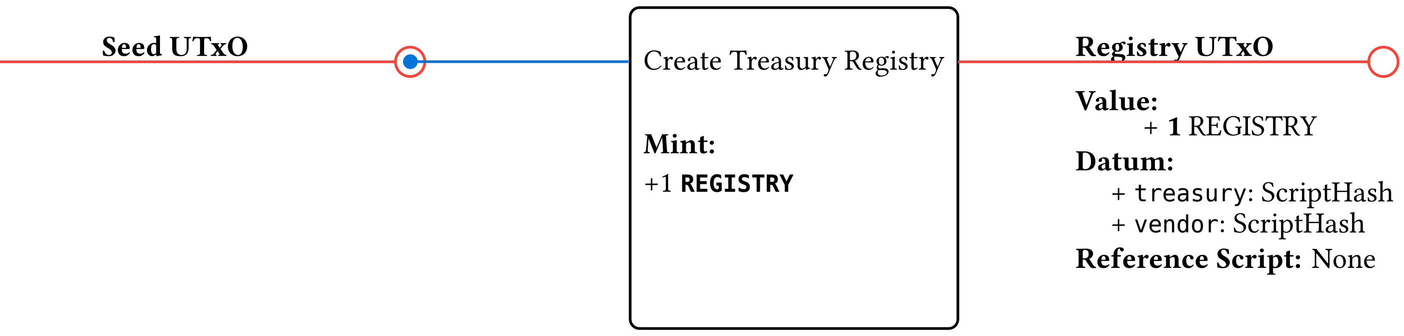


Figure 7: Create Treasury Registry transaction

2.c.c.b - Delete Registry

The Registry UTxO is deleted by spending the Registry UTxO and burning the Registry NFT. The General Assembly must sign the transaction, and it can only be done after the expiration date defined in the Amaru Contracts configuration.

Involved redeemers:
- `Data` redeemer, `Spend` purpose: for spending the Amaru Treasury Registry UTxO.
- `Data` redeemer, `Mint` purpose: for burning the Amaru Treasury Registry NFT.

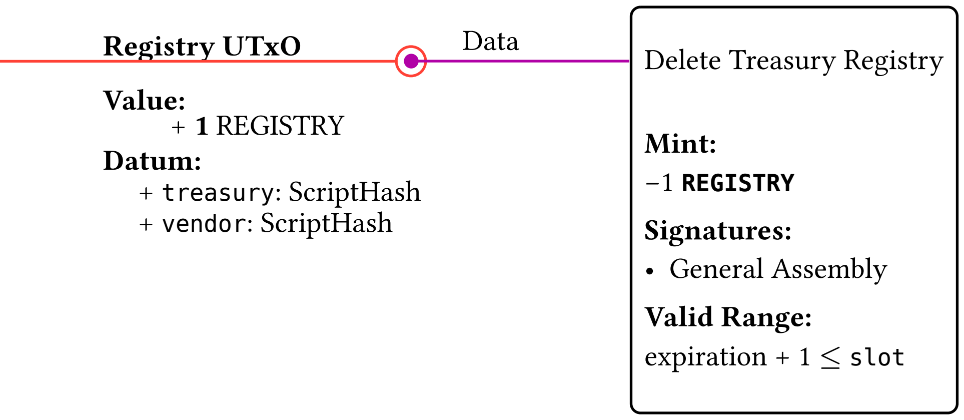


Figure 8: Delete Treasury Registry transaction

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 `c07ef7aa2003a0740f14fb0f0b2d4aca2ab27c8d`.

Filename
lib/address.ak
lib/inputs.ak
lib/outputs.ak
lib/registry.ak
lib/scope.ak
validators/permissions.ak
validators/traps.ak
aiken.toml
treasury-contracts/lib/utilities.ak
treasury-contracts/validators/treasury.ak

4 - Security Considerations

This section outlines the security scenarios analyzed during the audit to ensure comprehensive coverage of potential vulnerabilities.

4.a - Minting and Burning Trap Tokens in the Same Transaction

The minting policy for trap tokens validates as `mint or burn`, which theoretically could allow a scenario where one of `mint` and `burn` is `True` while both minting and burning occur in the same transaction.

However, this scenario is impossible in the protocol because both `mint` and `burn` functions verify that exactly one token is minted or burned respectively. This ensures that if one condition is `True`, the other must be `False`.

4.b - Permissions Script Withdrawing Against the Wrong Script

The Permissions script is designed to be used as a zero-withdrawal script against the Treasury script.

However, the Permissions validator only checks that there are script inputs from *one* script in the transaction. This does not guarantee that the Treasury script inputs are the ones being spent.

This is not a security issue since the Permissions script is not intended to hold any funds.

4.c - Double Satisfaction Scenarios

The protocol only contains Scopes and Treasury Registry UTxOs, both of which maintain state and are identified by NFTs. This design eliminates the possibility of double satisfaction scenarios.

4.d - Expiration Date Synchronization Between Treasury and Scopes Validators

Both validators must share the same expiration date to prevent the potential issue described in AMR-301.

While this is not a vulnerability in itself, it is a best practice to ensure both validators expire simultaneously. This synchronization is managed through the Amaru Contracts configuration and some off-chain machinery that must be used to ensure it.

5 - Findings

ID	Title	Severity	Status
AMR-001	Treasury funds can be kidnapped by including a datum hash	Critical	Resolved
AMR-301	Possible mismatch on `expiration` time between Treasury and Scopes validators	Info	Resolved
AMR-302	Improve Code Readability	Info	Resolved
AMR-303	Prevent inclusion of reference scripts	Info	Resolved
AMR-304	Trash tokens can be added to multiple UTxOs	Info	Resolved

6 - AMR-001 Treasury funds can be kidnapped by including a datum hash

Category	Commit	Severity	Status
Bug	39544114d4ef3a7b285b3d15f8064e2585e380e1	Critical	Resolved

6.a - Description

Treasury UTxOs don't make use of the datum, so it is left unchecked in all operations.

This means that any UTxO created at the treasury script by any operation can contain no datum, or an arbitrary inline datum, or an arbitrary datum hash. And with this last option lies the issue.

When a script UTxO contains a datum hash, in order to consume it, you need to provide the datum data in the transaction witness field. But this restriction is not there when creating a script UTxO.

This means that anyone could, in the normal flow of the contract, include a datum hash in any treasury output without making public the data that hashes to said datum. The funds don't leave the treasury script, so they are not stealing them, but by being the only ones with the necessary data to consume it, they are preventing the consumption and essentially holding the funds hostage.

Most operations need some kind of permission to be executed, so the possible attackers are those who have those permissions to begin with. But the attack vector grows significantly when the expiration date is reached. At that point, anyone can execute the `Sweep` operation, and keep any native tokens still at the treasury contract hostage. (Keep in mind that at most 5 ADAs can be kept in the treasury address after a sweep operation, the rest need to be donated, so the attack only works for native tokens)

This finding was first found and reported by Arnaud Bailey.

6.b - Recommendation

We recommend updating the treasury-contract validation to ensure that all treasury UTxOs created contain `NoDatum` in the datum field.

6.c - Resolution

The client disallowed the use of `DatumHash` in the treasury UTxOs datums. At commit `c07ef7a`.

7 - AMR-301 Possible mismatch on `expiration` time between Treasury and Scopes validators

Category	Commit	Severity	Status
Robustness	39544114d4ef3a7b285b3d15f8064e2585e380e1	Info	Resolved

7.a - Description

In the case that the `expiration` field of the Treasury script is greater than the expiration in the Amaru Contracts configuration, it could happen that the General Assembly burns the Scopes UTxO NFT, thus removing the Scopes UTxO from the protocol, before the expiration date of the Treasury script. This leads to having the Treasury funds locked until its expiration date passes, when permissionless Treasury `Sweep` could be executed.

7.b - Recommendation

There's the possibility of encoding the expiration information on-chain for being accessible to both validators, although it might require lots of code updates. It might be useful to take into account in future versions, but for now we recommend being aware of this, maybe implement some off-chain mechanism to ensure it.

7.c - Resolution

The off-chain mechanism is implemented by using Aiken's config feature, relying on the config defined in Amaru-treasury's aiken.toml as the single source of truth. Then, this script must be used to instantiate the Treasury Contracts since it pulls from the Amaru's config.

8 - AMR-302 Improve Code Readability

Category	Commit	Severity	Status
Improve-ment	39544114d4ef3a7b285b3d15f8064e2585e380e1	Info	Resolved

8.a - Description

A couple of points where the code could be simplified:

1. For the trap-type validators, we understand that the modularization is made for the purpose of isolating the part where those two validators (scopes and treasury registry) are different: allowing or disallowing datum updates.

However, we think that it makes them harder to understand, and the right amount of code duplication might be favorable if it helps with readability. Particularly, the `spend` function has too many responsibilities.

2. `mint_or_burn` function adds codebase hopping with no lines of code savings.
3. `with_inline_*` functions could be replaced with their definitions too: they don't save more than one line of code, since their 2nd parameter, the function parameter, is not being used. We prefer explicit checks on the datum shapes where needed.
4. `expect_single_script` has a 3rd parameter, a function parameter, that is not really used in the codebase.

8.b - Recommendation

1. Being more explicit *inside* validators' `spend` might be beneficial for readability.

```
spend(
  _datum: Option<Data<Scopes>>,
  _redeemer: Data,
  utxo: OutputReference,
  self: Transaction,
) {
  // scope or registry, token name
  let asset_name = ...
  let own_input = inputs.resolve(self.inputs, utxo)

  expect must_be_approved_by_general_assembly(self)
  expect Credential.Script(own_script) = own_input.address.payment_credential

  or {
    // either the token is burnt;
    (assets.tokens(self.mint, own_script) != dict.empty)?,
    // or the output is forwarded.
    {
      let output = outputs.forwarding_nft(own_script, asset_name, self.outputs)
      // 1. input has trapped NFT
      // 2.1. output correct address and NFT (checked by forwarding_nft)
      // 2.2. output datum property
      // 2.3. output doesn't have script ref
    },
  }
}
```

2. `mint_or_burn` could be just replaced by its definition where it is invoked
3. `with_inline_*` functions could be replaced with their definitions too.
4. The `return` parameter might be removed in favour of simplicity. Also, the returned script hash is not used in the validator, and futhermore, is not neccesarily the treasury script hash itself (as stated in the permissions validator code). So we recommend removing that return parameter too.

8.c - Resolution

1. The client considers that inlining the `spend` function might not be too beneficial for readability, but it modularizes well by isolating the common bits and making the divergence clearer at the top level. We agree with this.

2, 3 and 4: Resolved in commit `e74579d`.

9 - AMR-303 Prevent inclusion of reference scripts

Category	Commit	Severity	Status
Improve-ment	39544114d4ef3a7b285b3d15f8064e2585e380e1	Info	Resolved

9.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.

9.b - Recommendation

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

9.c - Resolution

Resolved in commit `cd43e80`.

10 - AMR-304 Trash tokens can be added to multiple UTxOs

Category	Commit	Severity	Status
Improve-ment	`39544114d4ef3a7b285b3d15f8064e2585e380e1`	Info	Resolved

10.a - Description

No outputs in the protocol have their values completely validated, that is, they don't have all contents of the value checked to ensure the value is exactly what is expected, no more, no less. This allows for extra "trash" tokens to be added to the UTxO.

Trash tokens can be included in the following cases:

- Creation and update of Scopes UTxO
- Creation and update of Treasury Registry UTxO

The presence of trash tokens would increase the minimum amount of ADA required for those UTxOs, and could make certain operations that rely on going over the value of UTxOs more expensive or even impossible to consume in any transaction.

10.b - Recommendation

We recommend updating the value validation for those outputs so no more tokens than needed can be stored.

10.c - Resolution

Resolved in commit `471f396`.

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 Pragma 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: `lib/address.ak` **Hash**: `9b4cfe214600e2bc9bc47cd9f779377658316a57a48f5d6c143931f35d131bb5`
Filename: `lib/inputs.ak` **Hash**: `dc4d3892f274faf9b189d6a1ca63e05414075ebdd7b0a1f90ca97e0d7592ddb1`
Filename: `lib/outputs.ak` **Hash**: `3e7d7d9b49bc27285ae021f7669acc80732e04cf797b8b7006d91edf8c1202e3`
Filename: `lib/registry.ak` **Hash**: `86388370ab26147b39160ee0e718d10f27c068ebb2105f5482e9a6e40bbb2a43`
Filename: `lib/scope.ak` **Hash**: `848355245201bc12df6a74a22f36dc092d8b56c3db394c9a2ae2e788662495d1`
Filename: `validators/permissions.ak` **Hash**: `ab608e49a242e24f1d78e1ca559a90261cb7e5da0129d405bc7aaccc81b3798c`
Filename: `validators/traps.ak` **Hash**: `acd262c60708c6397dffa7a0e894859f4a61532da2dc8d81fe8e4edcc927e8f5`
Filename: `aiken.toml` **Hash**: `92bf199cbbf025a4ffa0a9df953b345d568c41b1e821d67ff7fbede4cd6acc78`
Filename: `treasury-contracts/lib/utilities.ak` **Hash**: `c304b9f9a69ffb474454d1c045abcea7cc954a7659e20910da90ceb182e327a3`
Filename: `treasury-contracts/validators/treasury.ak` **Hash**: `e09827cbe91f78e43b56bd524b27171ac6067ba1027ea0c57774f9a06d6f3e57`

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