alwaystrue

Security Audit Report

29/05/2025

TxPipe - GitHoney

Robertino Martinez

Contents

1 - Summary

GitHoney is a bounty management system tailored for open-source contributions. The central part of this system is the GitHoney protocol, which enforces the rules for value transfer. This report provides a comprehensive audit of the GitHoney protocol.

The investigation lasted several weeks, during which critical and major vulnerabilities were discovered and subsequently resolved by the project team. The most severe issues allowed attackers to steal bounty rewards and avoid paying fees. The project team has resolved all the non-enhancement findings and greatly improved the code quality and security in the process.

The audit was conducted only on validators (ignoring off-chain and tests) and 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 - Project Overview

GitHoney is a bounty management system tailored for open-source software development tasks. It operates as a bot within GitHub, interacting with users through issues, PRs, and comments. This bot is responsible for setting up the on-chain contracts and monitoring all interactions. The process is straightforward: a maintainer creates a bounty on a GitHub issue. Then, a contributor can accept this bounty and eventually link it to a PR with the proposed solution. Finally, once the PR is completed, reviewed, and merged, the contributor can claim their reward.

The protocol implemented by the audited files describes only the on-chain contracts and their interactions.

1.b - Audit Process

The audit process started with thoroughly examining the protocol validators and documentation. Then, the audit team actively worked on ways to steal from, manipulate, and break the protocol using all the previously gathered knowledge. This included vulnerabilities both common and specific to this protocol, taking into account interactions between contracts, extraction of value, disruption to other users, and other attack vectors.

Findings and feedback from the audit were shared regularly with the project team through Discord, using git (GitHub) to track changes in both the source code and this report. For each new finding:

1. The audit team shared a description of the finding and a recommendation on how to resolve it.
2. The project team acknowledged the finding and provided a fix in a pull request (if they decided to resolve it).
3. The audit team reviewed the fix and either provided feedback or recommended the fix to be merged.
4. The project team merged the fix, and the audit team updated the report accordingly.
5. The audit team kept looking for findings, considering the new changes.

This loop continued until the project team resolved all findings (or acknowledged but explicitly ignored them).

The project team was responsive, demonstrated a commitment to ensuring the protocol's security and robustness, and addressed these issues efficiently and in a timely manner.

2 - Specification

Since the audited files do not have duplicate file names, we remove the paths to the files in the report to aid readability. So, for example, if we refer to `checks.ak`, we mean `onchain/lib/checks.ak`. We'll provide the full path if there's any ambiguity.

2.a - Glossary

2.a.a - Roles

- **Maintainer**: An individual or team maintaining a GitHub repository and having a task that requires completion within a deadline.
- **Contributor**: An individual or team that can complete the task by submitting a Pull Request (PR).
- **Admin**: A control actor that approves the changes made by the contributor and checks that the maintainer doesn't close the bounty when the PR is correct. In other words, it works as an oracle of the work done. The Admin is responsible for confirming the contributor's payment or reclaiming the deposited assets for the maintainer.
- **GitHoney**: An individual or team that manages the protocol instance and its settings. See Section 2.a.c for more information.

2.a.b - Constants

These values are unchangeable once an instance of the protocol is deployed.

- **SETTINGS_TOKEN_NAME**: Asset name of the SettingsNFT. It is set to be `"settingsNFT"`.
- **MIN_ADA**: The minimum amount of ADA required to create a bounty. It is set to **3 ADA**.
- **MAX_AMOUNT_OF_TOKENS_ADMITTED**: Maximum number of **asset classes** that can be used as rewards for a bounty. It is set to **15**.

2.a.c - Protocol Settings

These values can change during the protocol's lifetime. They are set by GitHoney and stored in Settings UTxO's datum.

- **GitHoney's Address**: The address belonging to the group that controls a particular instance of the GitHoney protocol.
- **Bounty Creation Fee**: Fee paid by the maintainer to GitHoney in order to create a Bounty UTxO.
- **Bounty Reward Fee**: Fee paid to GitHoney's Address in order to mark a Bounty UTxO as "merged" (accepted by the Admin).

2.a.d - Tokens

- **SettingsNFT**: A one-time minted token used to validate the creation and authenticity of the Settings UTxO. It identifies the settings and ensures the correctness of the datum. Its name is hardcoded to be SETTINGS_TOKEN_NAME.
- **BountyIdToken**: Token used to validate the creation and authenticity of a Bounty UTxO. It identifies the bounty with its name and ensures the correctness of the datum.

2.a.e - Validators

Validators that are part of the protocol:

- **SettingsValidator**: Validator that locks the Settings UTxO. Located at `githoney_contract.ak:settings:spend()`.

- **SettingsPolicy**: Minting policy that mints/burns the SettingsNFT. Located at `githoney_contract.ak:settings_minting:mint()`.
- **BountyValidator**: Validator that locks the Bounty UTxO. Located at `githoney_contract.ak:githoney:spend()`.
- **BountyPolicy**: Minting policy that mints/burns the BountyIdToken. Located at `githoney_contract.ak:githoney:mint()`.

Validators that are not part of the protocol but could optionally be used as add-ons:

- **BadgesValidator**: Validator that allows its UTxOs to be spent only if the transaction is signed by GitHoney's Address. Located at `githoney_contract.ak:badges_contract:spend()`.
- **BadgesPolicy**: Minting policy that allows minting any amount of tokens of any token name **once**. Located at `githoney_contract.ak:badges_policy:mint()`.

2.b - UTxOs

2.b.a - Settings UTxO

There is one Settings UTxO per protocol instance. All the bounties, independent of the Maintainer or Admin, share the same settings.

The SettingsValidator contains this UTxO's spending validator.

Settings UTxO's key properties	
Creation is validated by the Deploy Settings transaction.	
Address	Hash of SettingsValidator with no parameters.
Datum	Type defined in `types.ak:SettingsDatum` containing the following fields: • GitHoney's Address (`githoney_address: Address`) • BountyCreationFee (`bounty_creation_fee: Int`) • BountyRewardFee (`bounty_reward_fee: Int`)
Value	• At least minADA of ADA • Exactly 1 (one) SettingsNFT
Scripts attached	• Has the BountyValidator validator attached.

2.b.b - Bounty UTxO

There is one Bounty UTxO holding the reward assets for the contributor per bounty. There's no limit to how many bounties one instance of the protocol could have.

The BountyValidator contains this UTxO's spending validator.

Bounty UTxO's key properties	
Creation is validated by the Create Bounty transaction.	
Address	Hash of BountyValidator parameterized with: • The SettingsNFT minting policy (`settings_policy_id: PolicyId`)
Datum	Type defined in `types.ak:GithoneyDatum` containing the following fields: • The admin's payment credential (`admin_payment_credential: Credential`) • The maintainer's address (`maintainer_address: Address`) • Maybe the contributor's address (`contributor_address: Option<Address>`) • The BountyRewardFee (`bounty_reward_fee: Int`)

	• The bounty's deadline (`deadline: Int`) • Is the bounty merged? (`merged: Bool`) • Initial value of the bounty (`initial_value: Pairs<PolicyId, Pairs<AssetName, Int>>`)
Value	Contains at least: • minADA • Exactly 1 (one) BountyIdToken • The bounty's reward assets

2.c - Transactions

There are dependencies between some transactions. The intended order of transactions is as follows:

1. GitHoney administrators deploy the settings of the dApp.
2. Then, for each bounty:
   1. The Maintainer creates a bounty
   2. Anyone can *optionally* add rewards to the bounty
   3. A Contributor can assign themselves to the bounty
   4. The bounty's Admin either closes the bounty or merges the bounty
   5. And finally, the Contributor can claim the bounty if it was merged.
3. At any point, GitHoney administrators can update the settings or close the settings of the dApp.

2.c.a - Deploy Settings

This transaction deploys the Settings UTxO, which holds the DApp's global parameters. The SettingsNFT minted from the SettingsPolicy uniquely identifies the Settings UTxO and validates the datum's correctness. In addition to the settings, this UTxO will also have the BountyValidator attached.

This Settings UTxO will be used both as a reference input and a reference script for all the bounty-related transactions.

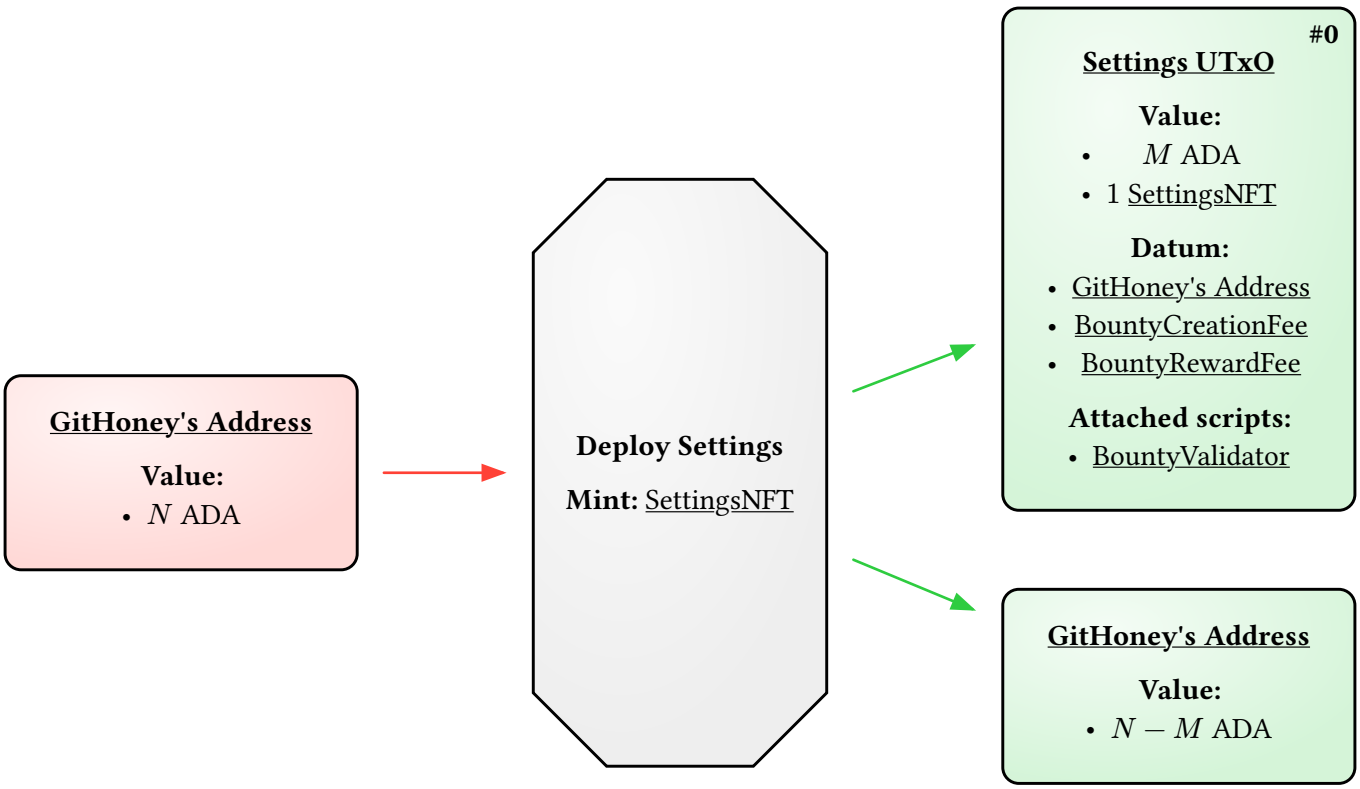

Figure 1: Transaction that deploys the Settings UTxO.

Inputs | Outputs

Note: Tx fees paid by UTxOs outside the protocol.

2.c.a.a - Code

Creation is validated by the checks in the SettingsPolicy parameterized by:
- The reference of a UTxO **freely chosen** by the Tx submitter (`utxo_ref`).
- The SettingsValidator's `Address` (`settings_script_addr`).

2.c.a.b - Expected failure scenarios

Transaction
• $N \neq 1$ **asset classes** (specific `PolicyId` and `AssetName` combination) are being minted/burnt in the transaction • $N \neq 1$ **amount of assets** are being minted in the transaction • There's **no input** UTxO with reference equal to the `utxo_ref` parameter

`script_output`	
• The **first** output of the transaction has a **different** `Address` as the SettingsValidator's one (`settings_script_addr`). We call this output `script_output`.	
Datum	• **Doesn't** have an inline datum • Has a **different shape** than `types.ak:SettingsDatum`
Value	• Its value **doesn't** contain at least one SettingsNFT

2.c.b - Update Settings

Updates the global parameters of the dApp by changing the datum of the Settings UTxO.

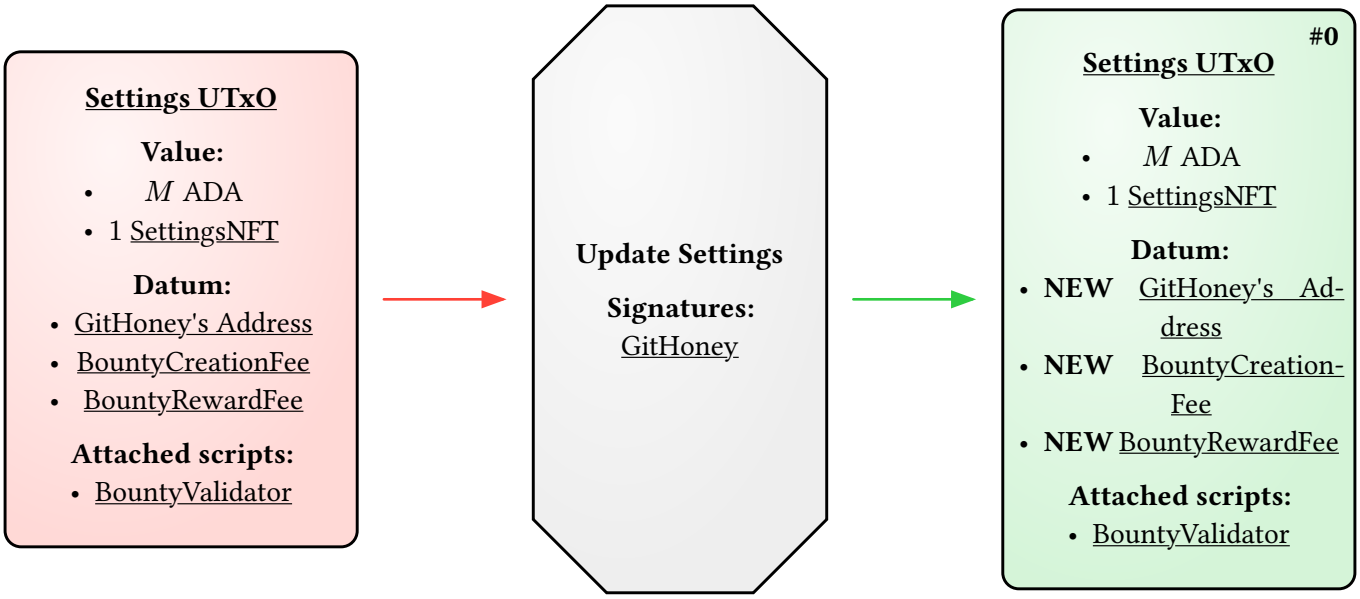

Figure 2: Transaction that updates the Settings UTxO.

Inputs | Outputs

Note: Tx fees paid by UTxOs outside the protocol.

2.c.b.a - Code

The transaction is validated by the checks in SettingsValidator with redeemer `types.ak:SettingsRedeemer:UpdateSettings`.

2.c.b.b - Expected failure scenarios for all transactions consuming the Settings UTxO

Transaction
• **Is not** signed by GitHoney's Address

`script_input`	
• There is $N \neq 1$ **script** inputs being spent. Since we're running this validator, the script input is the Settings UTxO. We call this input `script_input`.	
Datum	• **Doesn't** have a datum. • Has a **different shape** than `types.ak:SettingsDatum`.
Value	• **Doesn't** contain **exactly one** SettingsNFT and **no other native token**, excluding Lovelace.

2.c.b.c - Expected failure scenarios specific to this transaction

`script_output`

• The **first** transaction output has a **different** `Address` than the `script_input`'s one (Settings UTxO). We call this output `script_output`.	
Datum	• **Doesn't** have an inline datum. • Has a **different shape** than `types.ak:SettingsDatum`.
Value	• **Doesn't** contain **exactly one** SettingsNFT and **no other native token**, excluding Lovelace.

2.c.c - Close Settings

It deletes the Settings UTxO, burns the SettingsNFT, and refunds the ADA locked to GitHoney. This effectively stops the protocol from working.

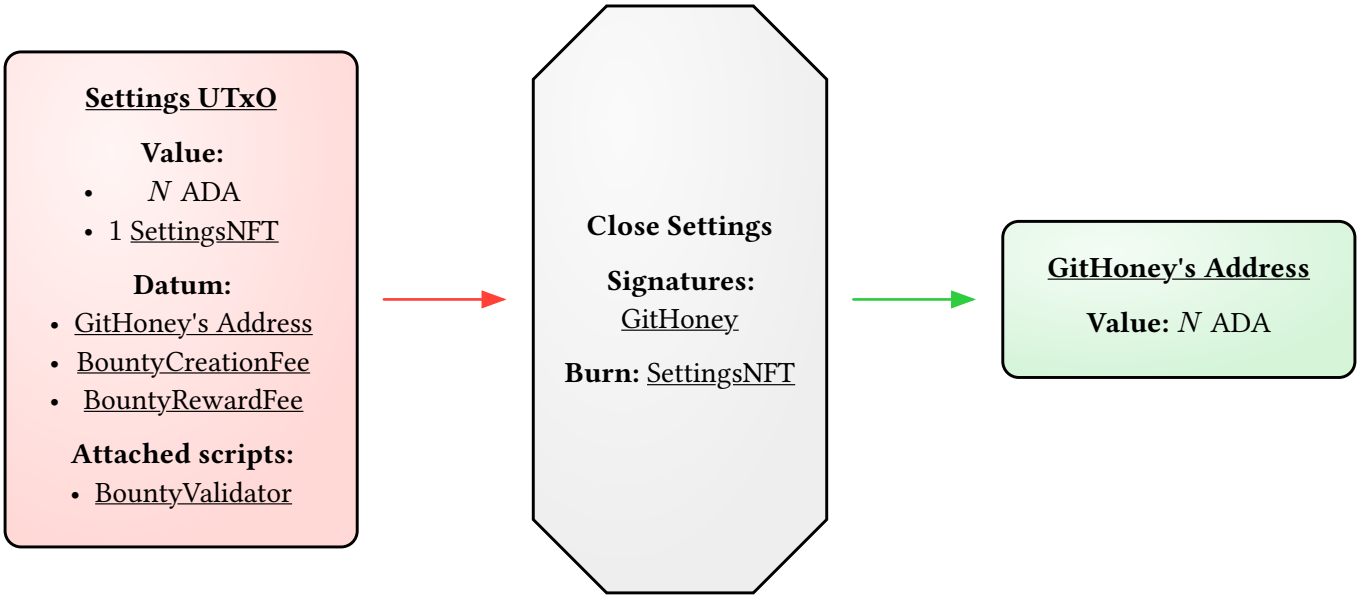

Figure 3: Transaction that closes the Settings UTxO.

Inputs | Outputs

Note: Tx fees paid by UTxOs outside the protocol.

2.c.c.a - Code

The transaction is validated by the combined checks in:
- The SettingsValidator with redeemer `types.ak:SettingsRedeemer:CloseSettings`. The logic requires running the SettingsPolicy to burn the SettingsNFT.
- The SettingsPolicy parameterized by the reference for a UTxO **freely chosen** by the Tx submitter (`utxo_ref`).

Note: The SettingsPolicy doesn't require the SettingsValidator to be run if burning the SettingsNFT.

2.c.c.b - Expected failure scenarios

2.c.c.b.a - On the SettingsValidator side:

- Expected failure scenarios for all transactions consuming the Settings UTxO.

Transaction
• **Is burning** $N \neq 1$ tokens with name equal to SETTINGS_TOKEN_NAME and `PolicyId` equal to the token locked in the Settings UTxO.

2.c.c.b.b - On the SettingsPolicy side:

Transaction
• We're **burning** $N \neq 1$ SettingsNFT tokens in the transaction. Split into two checks: ‣ $N \neq 1$ **asset class** (specific `PolicyId` and `AssetName` combination) is being burnt in the transaction ‣ $N \neq 1$ **amount of assets** are being burnt in the transaction

2.c.d - Create Bounty

Creates a Bounty UTxO locking the reward assets plus MIN_ADA and a BountyIdToken. It sets the Maintainer, deadline, Admin, and merged flag (set to `False`) in the datum.

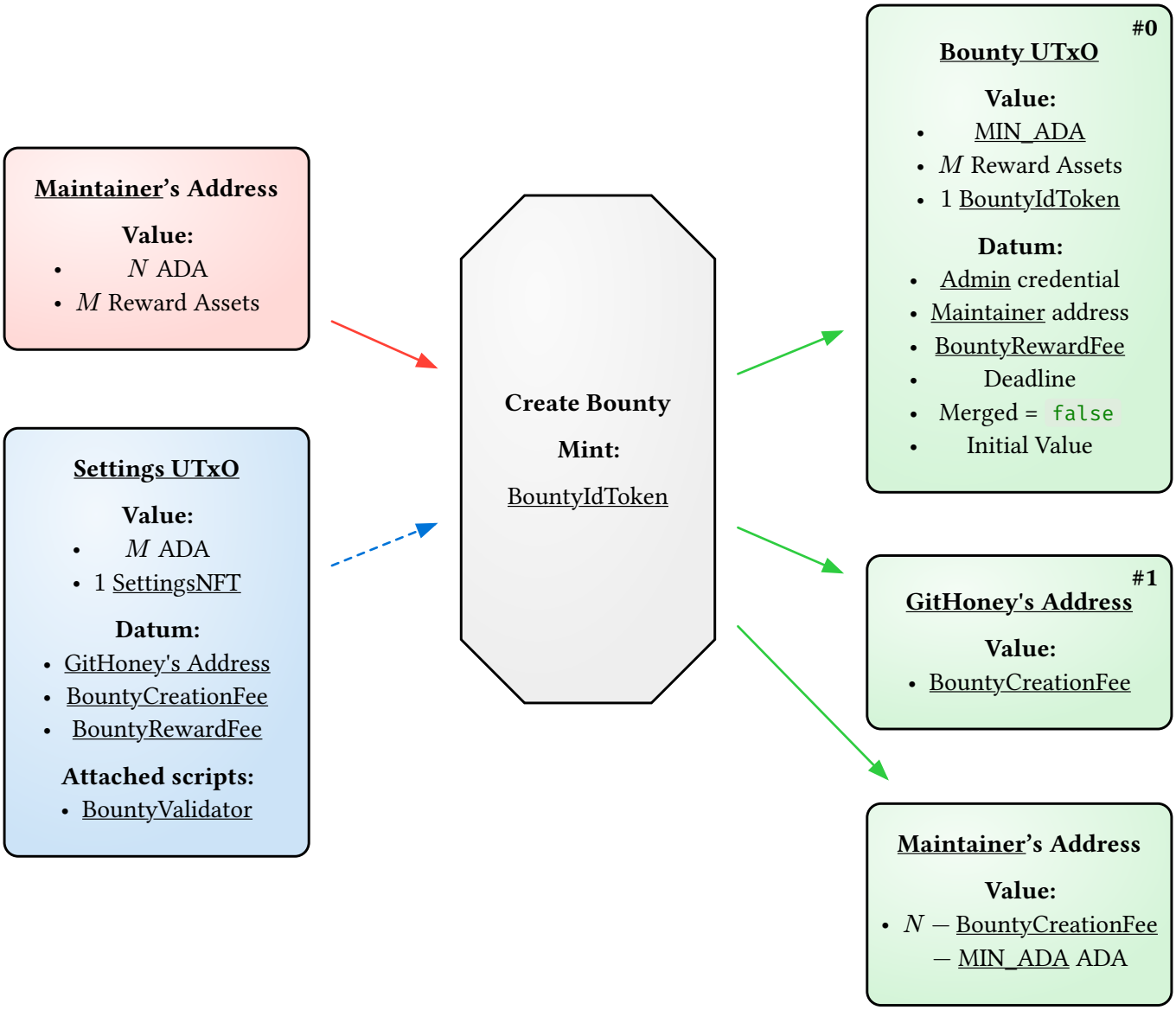

Figure 4: Transaction that creates a Bounty UTxO.

Inputs | Reference Inputs/Scripts | Outputs

Note: Tx fees paid by UTxOs outside the protocol.

2.c.d.a - Code

Creation is validated by the checks in the BountyPolicy parameterized by the SettingsPolicy's `PolicyId`

2.c.d.b - Expected failure scenarios

Transaction
• **Is** minting/burning $N \neq 1$ asset classes (specific `PolicyId` and `AssetName` combination). • **Is** minting $N \neq 1$ assets. • Validity range upper bound **is not** finite.

`ref_input`	
• There's $N \neq 1$ reference inputs. We call this input `ref_input`.	
Datum	• **Is not** an inline datum. • Has a **different shape** than `types.ak:SettingsDatum`.
Value	• Contains $N \neq 1$ SettingsNFT.

`script_output`	
• The **first** output of the transaction is **not** sent to an address with the **exact same** `payment_credential` as the BountyPolicy and no `stake_credential`. We call this output `script_output`; its address is the same as Bounty UTxO's.	
Datum	• **Is not** an inline datum. • Has a **different shape** than `types.ak:GithoneyDatum`. • `merged` field **is not** set to `False`. • `deadline` field **is smaller than** the validity range upper bound. • `contributor_address` **is not** empty. • `bounty_reward_fee` **is different** to the `bounty_reward_fee` specified in the Settings UTxO datum. • The `initial_value` field **is different** than the total value locked without the BountyIdToken.
Value	• Contains $N < 1$ **BountyIdTokens** or $M <$ **MIN_ADA** of ADA.

`githoney_output`	
• The **second** output of the transaction is **not** sent to the GitHoney's Address. We call this output `githoney_output`.	
Value	• Contains $N <$ **BountyCreationFee** of ADA.

2.c.e - Add Rewards to bounty

Adds additional reward assets to an existing Bounty UTxO. The bounty must not be merged (`merged = false`). Anyone can add rewards to the bounty, and this transaction takes the Settings UTxO as reference input and a Bounty UTxO to produce another Bounty UTxO.

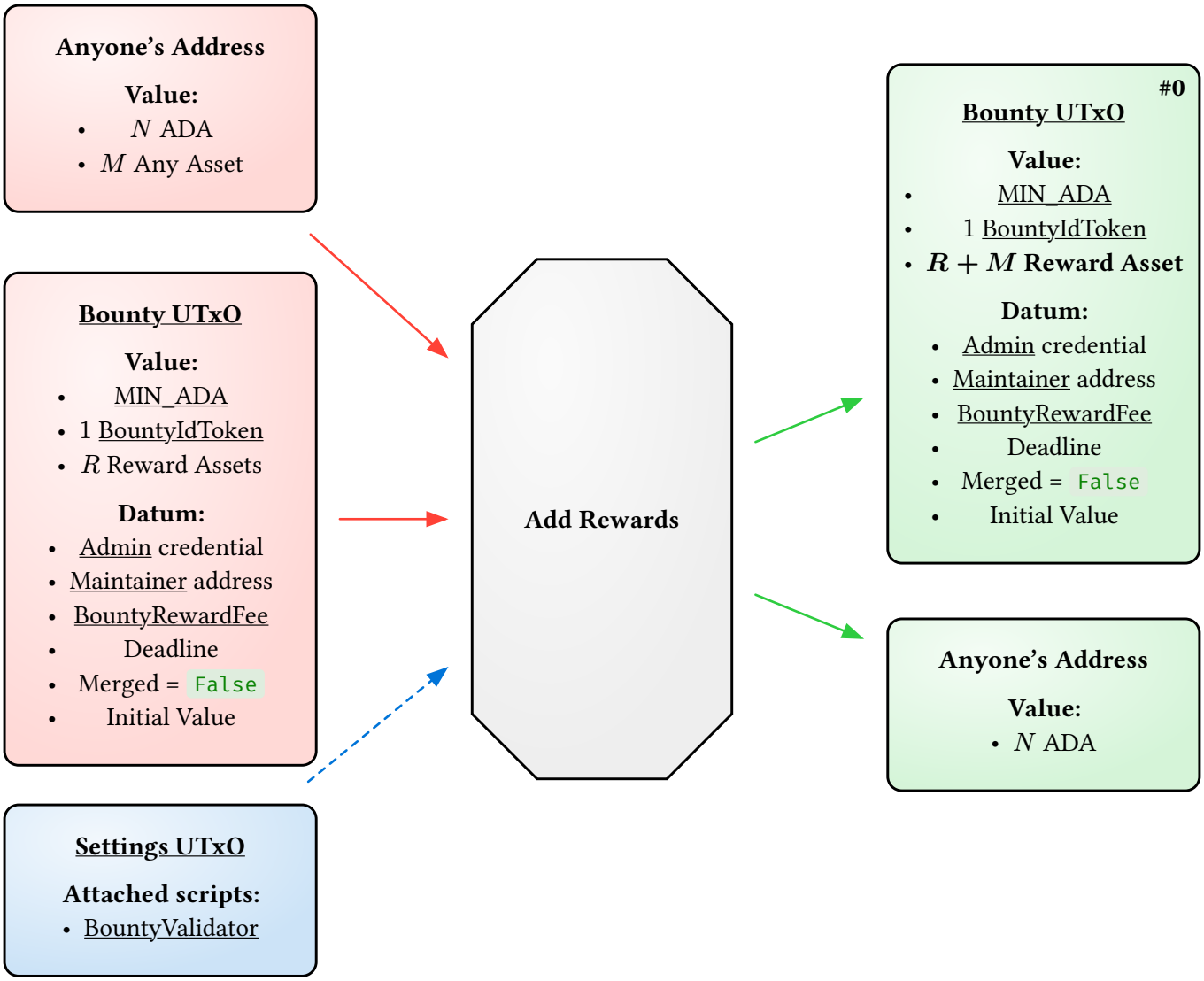

Figure 5: Transaction that adds rewards to a Bounty UTxO.

Inputs | Reference Inputs/Scripts | Outputs

Note: Tx fees paid by UTxOs outside the protocol.

2.c.e.a - Code

Creation is validated by the checks at the `types.ak:GithoneyContractRedeemers:AddRewards` path of BountyValidator parameterized by the SettingsPolicy's `PolicyId`.

2.c.e.b - Expected failure scenarios for all transactions consuming the Bounty UTxO

`script_input`	
• There is $N \neq 1$ **script** inputs being spent. Since we're running this validator, the script input is the Bounty UTxO. We call this input `script_input`.	
Datum	• **Doesn't** have a datum. • Has a **different shape** than `types.ak:GithoneyDatum`.
Value	• It contains $N \neq 1$ BountyIdToken

2.c.e.c - Expected failure scenarios for this transaction

Transaction
• Validity range upper bound **is not** finite.

`script_input`	
Datum	• `deadline` field is **smaller** than validity range upper bound • `merged` field is set to `True`

`script_output`	
• The **first** transaction output has a **different** `Address` than `script_input` (Bounty UTxO). We call this output `script_output`.	
Datum	• **It's not** an inline datum. • Has a **different shape** than `types.ak:GithoneyDatum`. • **It's different** than `script_input`'s datum.
Value	• **Has** either the same **exact same value** or **less of any token** than `script_input`'s value. • **Has more** than MAX_AMOUNT_OF_TOKENS_ADMITTED of asset classes.

2.c.f - Assign contributor

Sets the Contributor's `Address` in the Bounty UTxO's datum and adds the Contributor's MIN_ADA to the value.

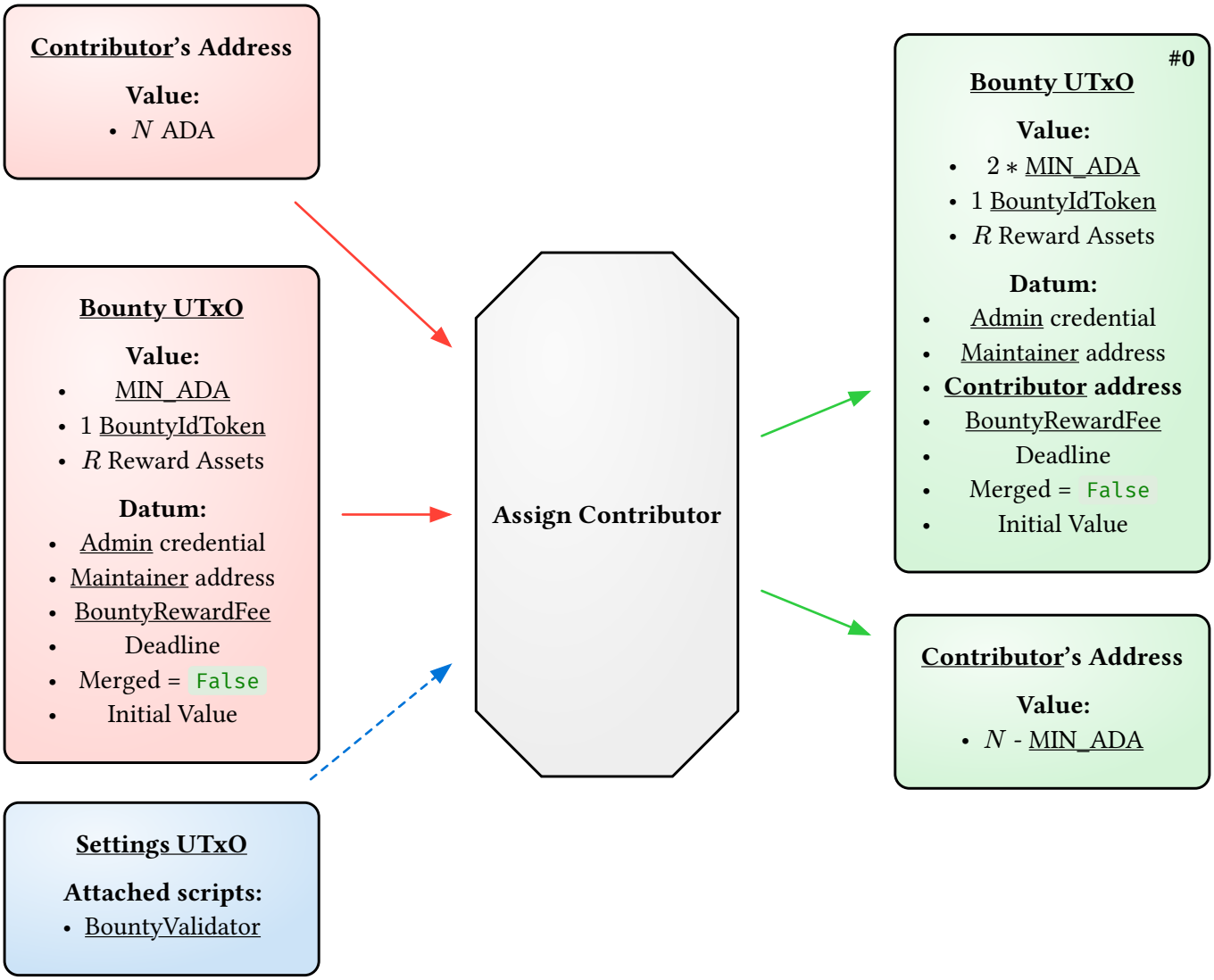

Figure 6: Transaction that assigns a contributor to a Bounty UTxO.

Inputs | Reference Inputs/Scripts | Outputs

Note: Tx fees paid by UTxOs outside the protocol.

2.c.f.a - Code

Creation is validated by checks at the `types.ak:GithoneyContractRedeemers:Assign` path of BountyValidator parameterized by the SettingsPolicy's `PolicyId`.

2.c.f.b - Expected failure scenarios

- Expected failure scenarios for all transactions consuming the Bounty UTxO

Transaction
• Validity range upper bound **is not** finite.

`script_input`	
Datum	• `deadline` field is **smaller** than validity range upper bound

	• `contributor_address` field is **not** empty

`script_output`	
• The **first** transaction output has a **different** `Address` than `script_input`. We call this output `script_output`.	
Datum	• **It's not** an inline datum. • Has a **different shape** than `types.ak:GithoneyDatum`. • **It's not** exactly the same as `script_input`'s datum except for the `contributor_address` field. • `contributor_address` field **is** empty (`None`).
Value	• **Is not** `script_input`'s value plus MIN_ADA.

2.c.g - Close Bounty

The Admin closes the Bounty UTxO, returning the reward assets to the Maintainer and burning the BountyIdToken. If a Contributor is assigned, the MIN_ADA is returned to them.

⚠ **Important:** If more rewards were added on top of the `initial_value`, the protocol doesn't regulate where they go. So, the Admin has sole discretion over where to send them. See the *Centralization* section for more details.

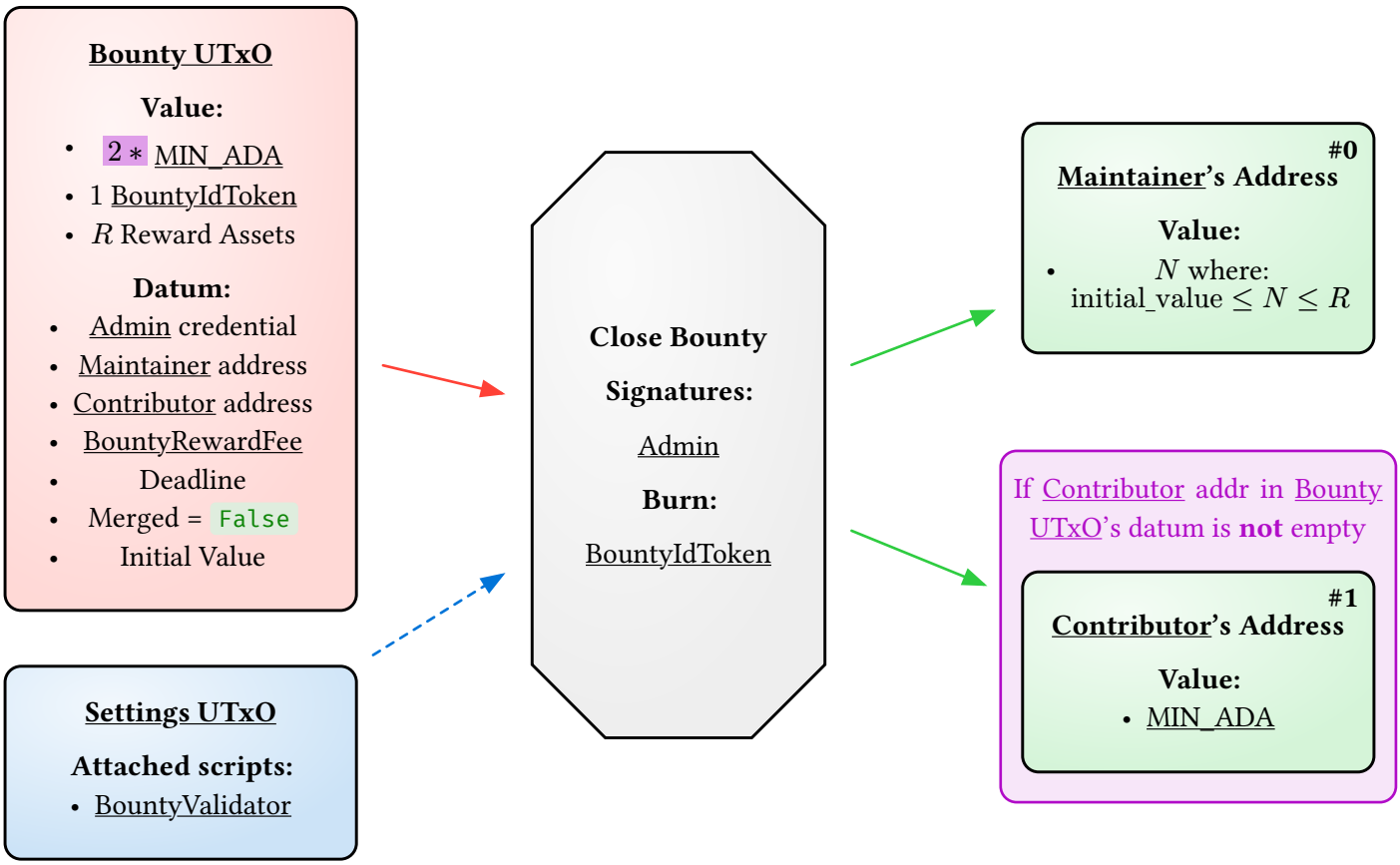

Figure 7: Transaction that closes a Bounty UTxO.

Inputs | Reference Inputs/Scripts | Conditional properties | Outputs

Note: Tx fees paid by UTxOs outside the protocol.

2.c.g.a - Code

The transaction is validated by the checks in:

- The `types.ak:GithoneyContractRedeemers:Close` path of BountyValidator parameterized by the SettingsPolicy's `PolicyId`. The logic requires running the BountyPolicy to burn the BountyIdToken.
- The BountyPolicy parameterized by the SettingsPolicy's `PolicyId`. Which, in this case **doesn't add any checks** that aren't already present in the BountyValidator.

Note: The BountyPolicy doesn't require the BountyValidator to be run.

2.c.g.b - Expected failure scenarios

2.c.g.b.a - On the BountyValidator side:

- Expected failure scenarios for all transactions consuming the Bounty UTxO

Transaction	
Signing	• **Is not** signed by Admin's credential.
Minting	• **Is not** burning `script_input`'s BountyIdToken. Split in two checks: ‣ **Is minting/burning** $N \neq 1$ asset classes (specific `PolicyId` and `AssetName` combination). ‣ **Is burning** $N \neq 1$ asset with `PolicyId` equal to BountyPolicy and `AssetName` equal to the one locked in `script_input`.

`script_input`	
Datum	• `merged` field is set to `True`

`maintainer_output`	
• The **first** transaction output has a **different** `Address` than the `maintainer_addres` from `script_input`'s datum. We call this output `maintainer_output`.	
Value	• **Less than** `initial_value` (from `script_input`'s datum) of assets.

`contributor_output`	
• **IF** `contributor_address` **IS NOT EMPTY:** The **second** transaction output has a **different** `Address` than the `contributor_addres` from `script_input`'s datum. We call this output `contributor_output`.	
Value	• **Less than** MIN_ADA.

2.c.g.b.b - On the BountyPolicy side:

Transaction	
Minting	• We're **burning** $N \neq 1$ BountyIdToken. Split into two checks: ‣ **Is** burning $N \neq 1$ asset class (specific `PolicyId` and `AssetName` combination). And since this policy is running, the `PolicyId` is BountyPolicy. ‣ **Is** burning $N \neq 1$ asset.

2.c.h - Merge Bounty

Pays GitHoney the reward assets multiplied by the `bounty_reward_fee` and updates the `merged` field to `True`. The Contributor's MIN_ADA remains in the Bounty UTxO.

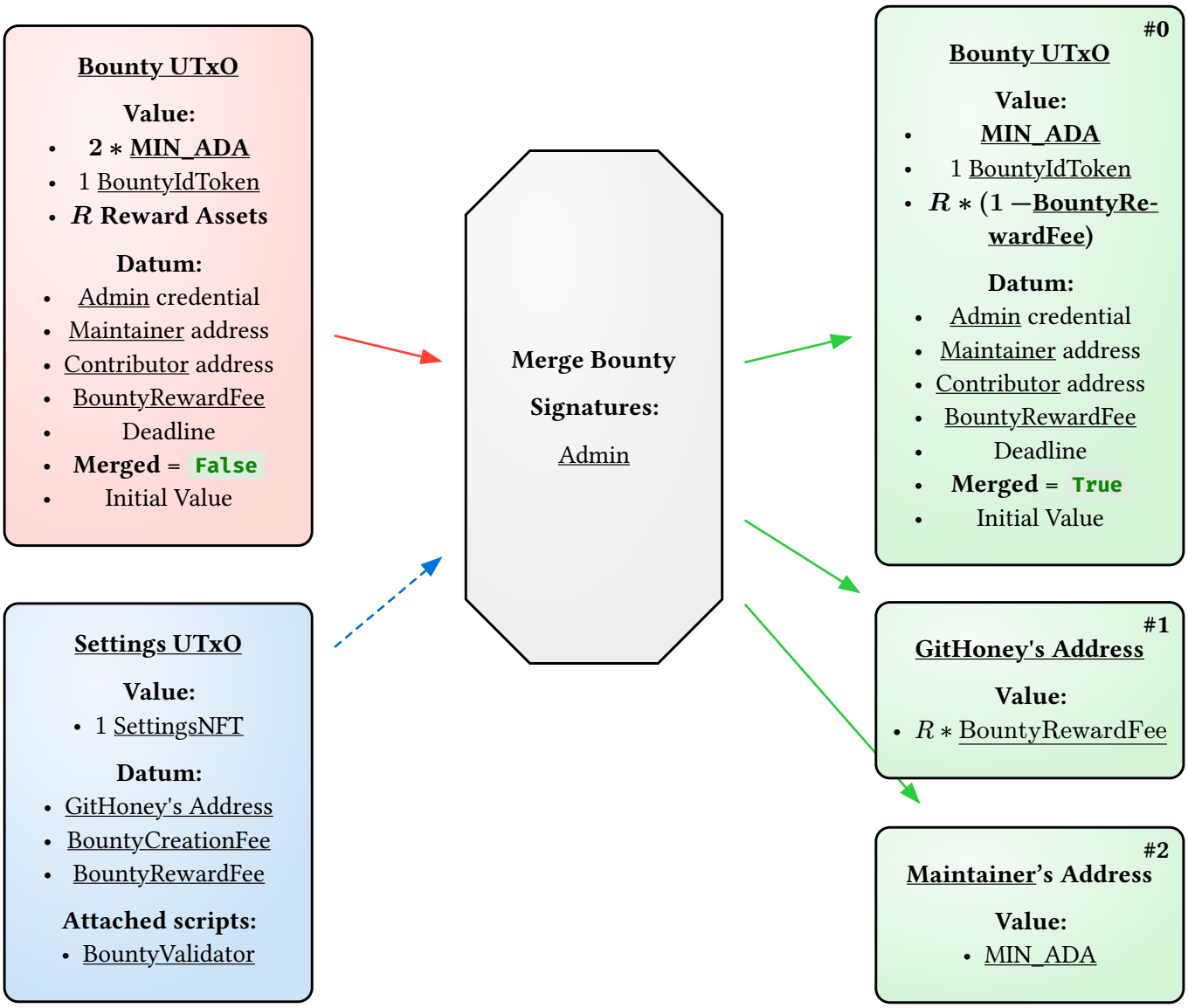

Figure 8: Transaction that merges a Bounty UTxO.

Inputs | Reference Inputs/Scripts | Outputs

Note: Tx fees paid by UTxOs outside the protocol.

2.c.h.a - Code

Creation is validated by the checks at the `types.ak:GithoneyContractRedeemers:Merge` path of BountyValidator parameterized by the SettingsPolicy's `PolicyId`.

2.c.h.b - Expected failure scenarios

- Expected failure scenarios for all transactions consuming the Bounty UTxO

Transaction	
• Validity range upper bound **is not** finite.	
Signing	• **Is not** signed by Admin's credential.

`ref_input`	
• There's $N \neq 1$ reference inputs. We call this input `ref_input`.	
Datum	• **Is not** an inline datum. • Has a **different shape** than `types.ak:SettingsDatum`.
Value	• It contains $N \neq 1$ SettingsNFTs.

`script_input`	
Datum	• `deadline` field is **smaller** than validity range upper bound • `contributor_address` field **is empty**. • `merged` field **is** set to `True`

`script_output`	
• The **first** transaction output has a **different** `Address` than `script_input`. We call this output `script_output`.	
Datum	• **Is not** an inline datum. • Has a **different shape** than `types.ak:GithoneyDatum`. • It's **different than** `script_input`'s datum with the `merged` field set to `True`.
Value	• **Different** than V_r where: $$\text{Total Rewards} = \text{script\_input's value} - 2 * \text{MIN\_ADA} - \text{BountyIdToken}$$ $$\text{Contributor Rewards} = \text{Total Rewards} * (1 - \text{BountyRewardFee})$$ $$V_r = \text{Contributor Rewards} + \text{MIN\_ADA} + \text{BountyIdToken}$$

`githoney_output`	
• The **second** transaction output has a **different** `Address` than GitHoney's Address defined in `ref_input`'s (Settings UTxO) datum. We call this output `githoney_output`.	
Value	• **Less than** V_f where: $$\text{Total Rewards} = \text{script\_input's value} - 2 * \text{MIN\_ADA} - \text{BountyIdToken}$$ $$V_f = \text{Total Rewards} * \text{BountyRewardFee}$$

`maintainer_output`	
• The **third** transaction output has a **different** `Address` than the Maintainer's one defined in `script_input`'s (Bounty UTxO) datum. We call this output `maintainer_output`.	
Value	• **Less than** MIN_ADA.

2.c.i - Claim Bounty

Pays the Contributor the remaining reward assets and burns the BountyIdToken.

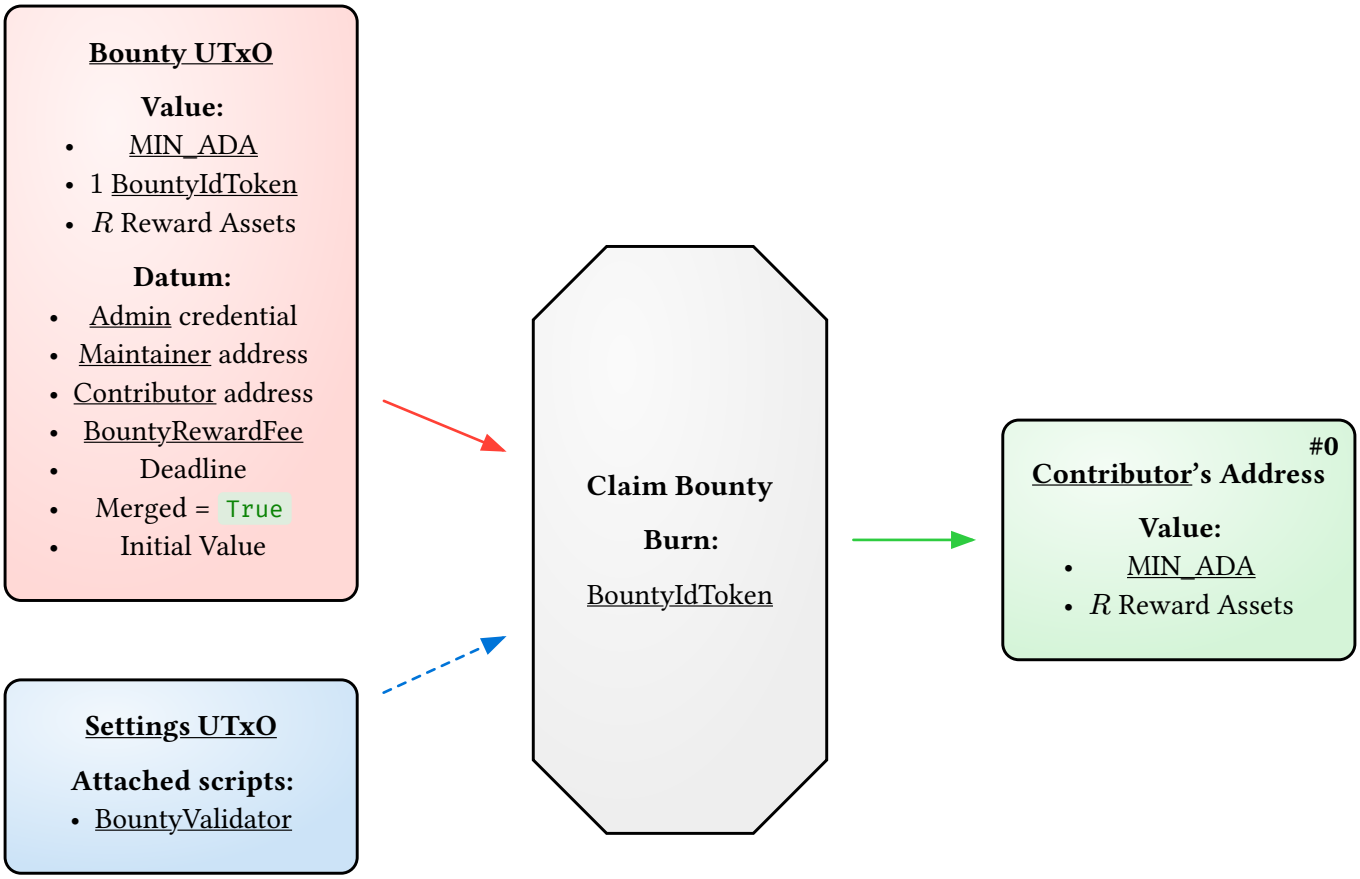

Figure 9: Transaction that claims a Bounty UTxO.

Inputs | Reference Inputs/Scripts | Outputs

Note: Tx fees paid by UTxOs outside the protocol.

2.c.i.a - Code

The transaction is validated by the combined checks in:

- The `types.ak:GithoneyContractRedeemers:Claim` path of BountyValidator parameterized by the SettingsPolicy's `PolicyId`. The logic requires running the BountyPolicy to burn the BountyIdToken.
- The BountyPolicy parameterized by the SettingsPolicy's `PolicyId`. Which, in this case **doesn't add any checks** that aren't already present in the BountyValidator.

Note: The BountyPolicy doesn't require the BountyValidator to be run.

2.c.i.b - Expected failure scenarios

2.c.i.b.a - On the BountyValidator side:

- Expected failure scenarios for all transactions consuming the Bounty UTxO

Transaction	
Minting	• **Is not** burning `script_input`'s BountyIdToken. Split into two checks:

	▸ **Is minting/burning** $N \neq 1$ asset class (specific `PolicyId` and `AssetName` combination). ▸ **Is burning** $N \neq 1$ asset with `PolicyId` equal to BountyPolicy and `AssetName` equal to the one locked in `script_input`.

`script_input`	
Datum	• `merged` field is set to `False` • `contributor_address` field **is empty**.

`contributor_output`	
• The **first** transaction output has a **different** `Address` than the `contributor_addres` from `script_input`'s datum. We call this output `contributor_output`.	
Value	• **Is different** than `script_input`'s value minus one BountyIdToken

2.c.i.b.b - On the BountyPolicy side:

Transaction	
Minting	• We're **burning** $N \neq 1$ BountyIdToken. Split into two checks: ▸ **Is** burning $N \neq 1$ asset class (specific `PolicyId` and `AssetName` combination). And since this policy is running, the `PolicyId` is BountyPolicy. ▸ **Is** burning $N \neq 1$ asset.

2.c.j - Mint badges (outside protocol)

Mints "badge" tokens. This transaction is not part of the protocol (can be ignored or used outside the protocol), but it is included here for completeness.

Important: The BadgesPolicy can be run once per `PolicyId` (one shot), so it doesn't allow burning the tokens once they are minted.

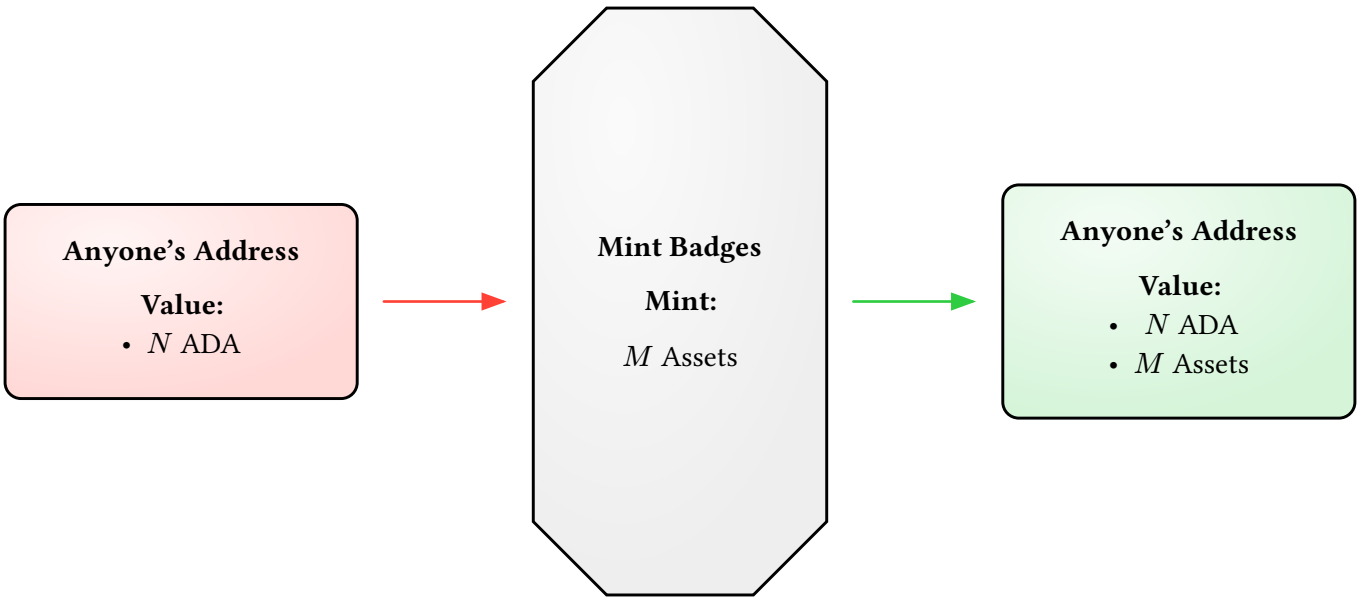

Figure 10: Transaction that mints BadgesPolicy tokens.

Inputs | Outputs

Note: Tx fees paid by UTxOs outside the protocol.

2.c.j.a - Code

The transaction is validated by the checks in the BadgesPolicy parameterized by any UTxO reference (`utxo_ref`) and an `Int` (`nonce`).

2.c.j.b - Expected failure scenarios

Transaction
• **Is not** consuming the `utxo_ref` UTxO as input. • nonce < 0

2.c.k - Unlock badges (outside protocol)

Unlocks the value locked in the BadgesValidator, which is assumed to be "badge" tokens. This transaction is not part of the protocol (can be ignored). However, it relies on the Settings UTxO.

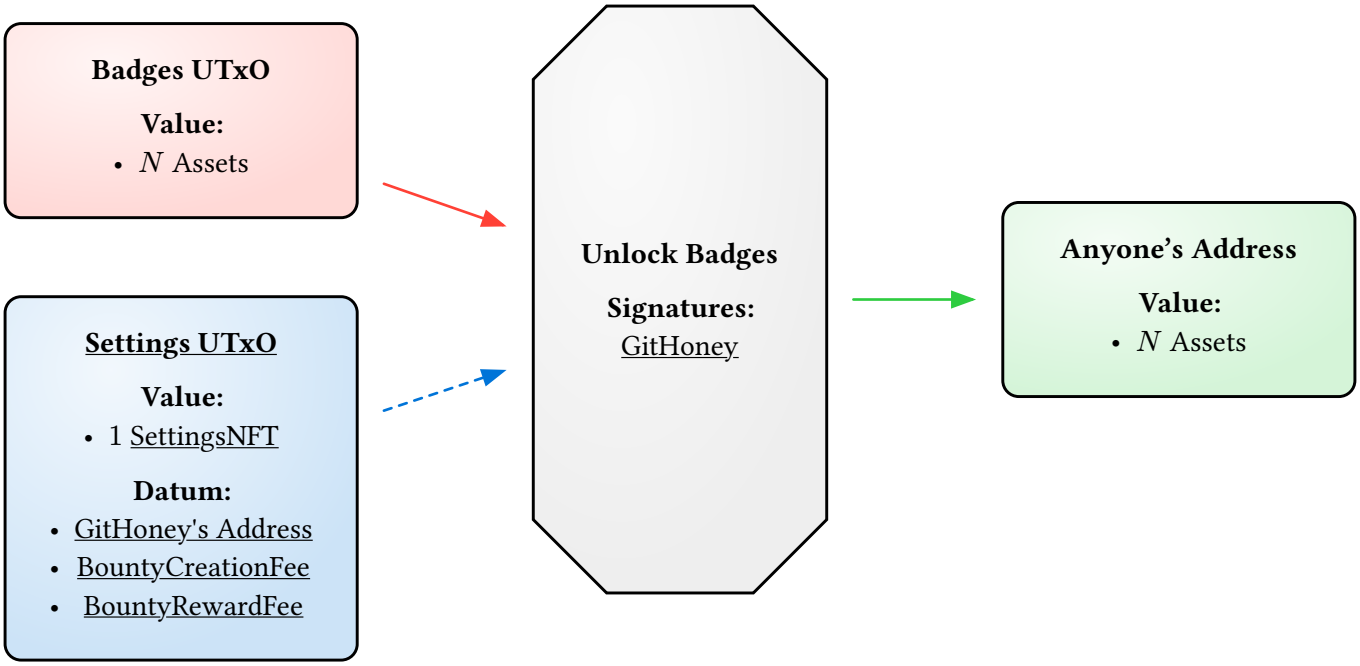

Figure 11: Transaction that unlocks value locked in the BadgesValidator.

Inputs | Reference Inputs/Scripts | Outputs

Note: Tx fees paid by UTxOs outside the protocol.

2.c.k.a - Code

The transaction is validated by the checks in the BadgesValidator parameterized by the SettingsPolicy's `PolicyId`.

2.c.k.b - Expected failure scenarios

Transaction
• **Is not** signed by GitHoney's credential.

`ref_input`	
• There's $N \neq 1$ reference inputs. We call this input `ref_input`.	
Datum	• **Is not** an inline datum. • Has a **different shape** than `types.ak:SettingsDatum`.
Value	• Contains $N \neq 1$ SettingsNFT.

2.d - Centralization

This section focuses on the protocol's centralization aspects, identifying potential vulnerabilities and recommendations for enhancing decentralization. Centralization can have different meanings in this context, including control distribution among participants, the concentration of resources, and the decision-making processes inherent to the protocol.

2.d.a - GitHoney's Control over the Protocol

By controlling the Settings UTxO, GitHoney has disproportionate control over the protocol. This means that:

- GitHoney can unilaterally stop the protocol on its tracks at any point by deleting the Settings UTxO. This is because the Settings UTxO not only contains information needed for some transactions but also contains the BountyValidator script itself.
- GitHoney can unilaterally update the settings of the protocol by changing the Settings UTxO's datum. This means that GitHoney can change the BountyCreationFee and the BountyRewardFee at any time. However, if GitHoney changes the BountyCreationFee, protocol users have the choice to stop creating more bounties without repercussions. On top of that, if GitHoney changes the BountyRewardFee, it won't affect already created Bounty UTxOs since those use the pre-established BountyRewardFee recorded in their Datums.

So, there's no issue with GitHoney updating the Settings UTxO, but **what if GitHoney deletes the Settings UTxO?**

In that case, the BountyValidator script will still be available on the blockchain, allowing motivated users to extract it from the blockchain history and use it. This, however, doesn't mean that the protocol is restored (since the SettingsNFT is lost), but it means that **all transactions that don't rely on the Settings UTxO's Datum** could still be performed.

Crucially, the close bounty transaction and the claim bounty transaction don't use the Settings UTxO, so they can still be performed. This means that **even if GitHoney deletes the Settings UTxO, bounties that were already "merged" can still be consumed, and the ones that weren't "merged" can still be "closed".** So, the work performed by Contributors won't be left unpaid **unless** the deletion of the Settings UTxO happens between the Contributor finishes the PR and the Admin creates the merge bounty transaction (which won't be possible to do anymore). And even in that case, the Admin would still be able to "close" the bounty and **choose** to send the rewards to the Contributor through a transaction **outside the protocol**.

> Overall, the audit team considers that **GitHoney's control over the current implementation of the protocol is not only acceptable, but an example on decentralized protocols**, since the worst case scenario (a malicious GitHoney operator) still allows the honest parties to recover the value locked by the protocol.

2.d.b - Admin's Control over bounties and rewards

By:

- Being **the only actor** capable of merging bounties and closing bounties
- Having **discretionary power** to choose where the **"extra" rewards** (rewards added through the add reward transaction) go when merging bounties.

The Admin has disproportionate control over the bounties and rewards assigned to them.

This means that the Admin could:

- Choose not to "merge" a bounty that was in fact merged outside the blockchain, leaving the Contributor unpaid.
- Choose not to "close" a bounty that was in fact closed outside the blockchain, leaving the Maintainer without access to the locked assets.
- Choose to send the "extra" rewards to anyone, including themselves, when "closing" a bounty.

Overall, the audit team considers that **the Admin's control over bounties and rewards is currently too high for a trustless interaction**. Meaning that: **both the Maintainer and the Contributor need to trust the Admin**. However, due to the inherent complexity derived from the use case (the Admin needs to decide if a Pull Request solves an issue), we understand that current technical limitations block the project team from delivering a fully trustless setup.

Mitigation strategy: One way to mitigate this issue could be to encourage Maintainers to use multi-sig wallets for Admins, so that there's at least *some* decentralization over the Admin's power.

2.e - Recommendations

This section provides recommendations to the TxPipe team for enhancing the protocol's security and usability. For centralization aspects, see the *Centralization* section.

2.e.a - Solve GH-301

We recommend prioritizing the resolution of issue GH-301 for the next release. Details about what the issue is and why it is important to solve it are in GH-301 finding.

2.e.b - Explore ways to reduce trust in the Admin

Due to reasons described in the *Centralization* section, we recommend exploring ways to reduce the Admin's control over bounties and rewards.

2.f - 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 `8118116c0f7873aa85c8a596aefe481af5508e83`.

Filename
onchain/validators/githoney_contract.ak
onchain/lib/checks.ak
onchain/lib/types.ak
onchain/lib/utils.ak
onchain/lib/validations.ak

3 - Findings Overview

ID	Title	Severity	Status
GH-001	Multiple Bounty Satisfaction	Critical	Resolved
GH-002	Resin instead of honey	Critical	Resolved
GH-003	Fatal exception on initialization	Critical	Resolved
GH-101	Honey-lock	Major	Resolved
GH-301	Claim on Merge	Enhance-ment	Acknowl-edged
GH-302	Don’t Repeat Yourself	Enhance-ment	Resolved
GH-303	It’s a match!	Enhance-ment	Resolved
GH-304	Comparison is the thief of joy	Enhance-ment	Resolved
GH-305	My Wallet Address	Enhance-ment	Resolved
GH-306	Paying myself fairly	Enhance-ment	Resolved
GH-307	The credential is enough	Enhance-ment	Resolved
GH-308	One UTxO per user	Enhance-ment	Resolved

GH-309	Restoring from the trash can	Enhance-ment	Resolved
GH-310	Get even faster!	Enhance-ment	Resolved
GH-311	Just a little bit DRYer	Enhance-ment	Resolved
GH-312	Implicit check	Enhance-ment	Resolved
GH-401	Spring Cleaning	Info	Resolved
GH-402	404 Not Found	Info	Resolved
GH-403	Misleading message	Info	Resolved
GH-405	Dirty boots	Info	Resolved
GH-406	Why are you hitting yourself?	Info	Resolved

4 - GH-001 Multiple Bounty Satisfaction

Category	Commit	Severity	Status
Bug	18997a07c0af182d41899e8f02e062dbffcb682b	Critical	Resolved

4.a - Description

The Bounty UTxO's uniqueness is based on the name of the BountyIdToken. However, **the name uniqueness is not enforced**. This can lead to two possible exploits:

- If two or more BountyIdTokens have the same name by accident, anyone can consume them with the "Add rewards" transaction and re-create only the one with more locked assets. **Stealing everything else**.
- Anyone can create a new Bounty UTxO with the same BountyIdToken as an existing Bounty UTxO and release one of those BountyIdToken to their wallet. Opening the door to creating **fake bounties and bounties with the wrong value/datum**.

4.b - Recommendation

We recommend checking that only one Bounty UTxO is being consumed per transaction.

4.c - Resolution

Resolved in commit `c0fcd2b1cdb60d7dc304cb0e406a39d5465897af`

5 - GH-002 Resin instead of honey

Category	Commit	Severity	Status
Bug	18997a07c0af182d41899e8f02e062dbffcb682b	Critical	Resolved

5.a - Description

In `checks.ak:is_utxo_value_valid():there_are_some_reward`, someone could provide a dummy token to get a non-zero value while locking less than MIN_ADA of ADA.

5.b - Recommendation

We recommend checking that the value of all tokens is greater than MIN_ADA + BountyIdToken.

5.c - Resolution

Resolved in commit `20dcabb13f8fb155f88b6637eb42f61c20657357`

6 - GH-003 Fatal exception on initialization

Category	Commit	Severity	Status
Bug	c0fcd2b1cdb60d7dc304cb0e406a39d5465897af	Critical	Resolved

6.a - Description

While solving GH-402, the project team introduced the `utxo_at_paid_to_address()` check in the SettingsPolicy. This check ensures that the address we're sending the SettingsNFT is the same as the one derived from the SettingsPolicy. However, the validator containing the SettingsPolicy doesn't have a spending validator, and because of that, the SettingsNFT will be locked by the default validator, which is an "always false" validator.

This means the team deploying the protocol will never be able to spend the SettingsNFT. Hence, they'll never be able to initialize the protocol.

6.b - Recommendation

Here are three possible solutions:

- Change the SettingsPolicy checks to check that the address where the SettingsNFT is being sent corresponds to the SettingsValidator (encouraged).
- Change the SettingsPolicy checks to ignore where the SettingsNFT is being sent (discouraged since it reintroduces GH-402).
- Put the SettingsPolicy and the SettingsValidator under the same multi-validator (discouraged since it'll make all Update Settings transactions unnecessarily larger due to the SettingsPolicy being needlessly embedded.

6.c - Resolution

Resolved in commit `206cf28355b29e411383b12df204c22c973b97f1`

7 - GH-101 Honey-lock

Category	Commit	Severity	Status
Bug	18997a07c0af182d41899e8f02e062dbffcb682b	Major	Resolved

7.a - Description

Putting the `githoney_policy():minting_policy()settings_datum` check before the split between minting and burning tokens makes the burning path **overly restrictive**, since we shouldn't need the Settings UTxO to burn the BountyIdToken. So, if GitHoney deletes the Settings UTxO, **the Contributor won't be able to claim already merged bounties!** This would forever lock already earned rewards.

7.b - Recommendation

The `githoney_policy():minting_policy()settings_datum` check should be moved inside the `else` clause to restrict only the minting path, but not the burning path.

7.c - Resolution

Resolved in commit `7765cb95397940f85f416a7e2e5568ac4b4f7552`

8 - GH-301 Claim on Merge

Category	Commit	Severity	Status
Optimization	18997a07c0af182d41899e8f02e062dbffcb682b	Enhancement	Acknowledged

8.a - Description

Once a bounty has been merged, the Admin creates the Merge Bounty transaction that pays the BountyRewardFee to GitHoney and MIN_ADA to the Maintainer, leaving only the rewards and updating the `merged` field in the datum of the new Bounty UTxO to `True`. Then, the Contributor creates the claim the bounty transaction to consume this new Bounty UTxO and get the rewards.

Even though this works, it is unnecessarily complicated since the same transaction that merges the bounty could also pay the Contributor their rewards and **avoid the claim transaction altogether**!

8.b - Recommendation

Merge the two transactions into one:
- Send the rewards to the Contributor in the Merge Bounty transaction.
- Remove the Claim Bounty transaction.
- Remove the `merged` field from the Bounty UTxO's datum since it is not needed anymore. If the Bounty UTxO exists, it means it's not merged.

This would SIGNIFICANTLY reduce the protocol's overall cost and attack surface and improve user experience.

8.c - Resolution

The project team decided not to resolve this finding.

9 - GH-302 Don't Repeat Yourself

Category	Commit	Severity	Status
Robustness	18997a07c0af182d41899e8f02e062dbffcb682b	Enhance-ment	Resolved

9.a - Description

Some parts of the codebase are repeated in multiple places. Increasing the code complexity and making it harder to maintain:
1. `close():is_maintainer_pay_valid` is repeated.
2. Some checks, such as `script_input_and_token_id_info`, are present in all transactions.

9.b - Recommendation

1. Move `close():is_maintainer_pay_valid` outside `are_payments_valid`.
2. Move repeated checks to the top of the validator.

9.c - Resolution

Resolved in commit `c0fcd2b1cdb60d7dc304cb0e406a39d5465897af`

10 - GH-303 It's a match!

Category	Commit	Severity	Status
Robustness	18997a07c0af182d41899e8f02e062dbffcb682b	Enhance-ment	Resolved

10.a - Description

- The `settings_amount` binding in `update_settings()` is only used to check if it equals `1` with the `is_only_one_token_input` check. This is unnecessary since one could pattern-match to `1` and avoid the `is_only_one_token_input` check altogether.
- The same is true for the `minted_quantity` binding in `close_settings()`; however, this one makes logical sense due to the context, so we leave that to the developer's preference.

10.b - Recommendation

- Remove the `is_only_one_token_input` check in `update_settings()` and pattern-match on `1` instead of binding the value to `settings_amount`.
- Consider the same for `minted_quantity == -1` check in `close_settings()`, but it is not strictly necessary to resolve this finding.

10.c - Resolution

Resolved in commit `c0fcd2b1cdb60d7dc304cb0e406a39d5465897af`

11 - GH-304 Comparison is the thief of joy

Category	Commit	Severity	Status
Optimization	18997a07c0af182d41899e8f02e062dbffcb682b	Enhancement	Resolved

11.a - Description

In `githoney_policy():minting_policy():is_creation_fee_paid`, we're comparing **all assets** in the value when **we only care about the amount of ADA**. This could be incredibly wasteful!

11.b - Recommendation

Use `lovelace_of()` to get the amount of ADA in the value and compare it directly to the Bounty-CreationFee as an `Int`.

11.c - Resolution

Resolved in commit `73210013a72a23afc39ded65227c4bee8d488195`

12 - GH-305 My Wallet Address

Category	Commit	Severity	Status
Optimization	18997a07c0af182d41899e8f02e062dbffcb682b	Enhancement	Resolved

12.a - Description

The custom `Wallet` type stores the parties' credentials. However, this type doesn't provide anything on top of the `Address` type that already exists in the Aiken standard library, and using `Wallet` is detrimental to performance because the `value_paid_to` function must transform `Wallet` values to `Address` values before using them.

12.b - Recommendation

Use `Address` instead of `Wallet` and remove the `value_paid_to` function.

12.c - Resolution

Resolved in commit `df73a658d1d89b1c24126dabc0d8cb5f14f9c62a`

13 - GH-306 Paying myself fairly

Category	Commit	Severity	Status
Optimization	18997a07c0af182d41899e8f02e062dbffcb682b	Enhancement	Resolved

13.a - Description

The `validations.ak:close_settings()` set of checks includes these two checks:

- `is_signed` that checks if GitHoney signed the transaction.
- `is_githoney_pay_valid` that checks if all the locked assets (except for the SettingsNFT that is burned in the transaction) are paid to GitHoney.

Since GitHoney is the only one that can sign the transaction, we can assume everything not burned in the transaction is being sent to GitHoney's Address, so the payment is valid.

13.b - Recommendation

Remove the `is_githoney_pay_valid` check to reduce transaction costs.

13.c - Resolution

Resolved in commit `1f3109cbfe2875bb1b7958fda7a055bdac601401`

14 - GH-307 The credential is enough

Category	Commit	Severity	Status
Optimization	18997a07c0af182d41899e8f02e062dbffcb682b	Enhancement	Resolved

14.a - Description

A `Wallet` data structure containing the payment and staking credentials is used in `types.ak:GithoneyDatum` to store the Admin's credentials. However, the protocol only uses this information to check if the tx is signed by the Admin. So, saving a whole `Wallet` type is unnecessary if a `PaymentCredential` is enough.

Notes:
- See GH-305 for related finding.
- This finding used to be identified as GH-311 when it was fixed by the project team in PR-109.

14.b - Recommendation

Use `Credential` instead of `Wallet` to store the Admin's credentials in `types.ak:GithoneyDatum`.

14.c - Resolution

Resolved in commit `84e29887ff7a10b9deb3e8d5442a007c03ba1f79`

15 - GH-308 One UTxO per user

Category	Commit	Severity	Status
Optimiza-tion	18997a07c0af182d41899e8f02e062dbffcb682b	Enhance-ment	Resolved

15.a - Description

The protocol allows users to split the non-Bounty UTxO outputs into several UTxOs, which is more expensive and makes the protocol more complex. This is unnecessary since there are no incentives to create more than one UTxO per output type.

15.b - Recommendation

Simplify the protocol by allowing only one UTxO per output type:
- To GitHoney's Address
- To Maintainer's address
- To Contributor's address

15.c - Resolution

Resolved in commit `c0fcd2b1cdb60d7dc304cb0e406a39d5465897af`

16 - GH-309 Restoring from the trash can

Category	Commit	Severity	Status
Optimization	18997a07c0af182d41899e8f02e062dbffcb682b	Enhancement	Resolved

16.a - Description

In `validations.ak:44`, we throw away the amount of the BountyIdTokens to later search for it again.

16.b - Recommendation

When getting the BountyIdToken information, pattern-match and bind the amount to a variable. Better yet, match it to `1` since the protocol only allows one BountyIdToken per Bounty UTxO.

16.c - Resolution

Resolved in commit `df73a658d1d89b1c24126dabc0d8cb5f14f9c62a`

17 - GH-310 Get even faster!

Category	Commit	Severity	Status
Optimization	18997a07c0af182d41899e8f02e062dbffcb682b	Enhancement	Resolved

17.a - Description

1. The implementation of `value_grater_than_or_equal` contains an unoptimized version of `match()` from Aiken's standard library.
2. The `utils.ak:tokens_to_value()` transformation is expensive and unnecessary in the latest Aiken.

Notes:
- The second point of this finding used to be identified as GH-312 when it was fixed by the project team in PR-109.

17.b - Recommendation

1. Update to the latest version of Aiken and use the `match()` function as part of the implementation.
2. Update to Aiken's latest version, use the optimized `match()` function, and remove the `utils.ak:tokens_to_value()` function.

17.c - Resolution

Resolved in commit `20dcabb13f8fb155f88b6637eb42f61c20657357`

18 - GH-311 Just a little bit DRYer

Category	Commit	Severity	Status
Optimization	20dcabb13f8fb155f88b6637eb42f61c20657357	Enhancement	Resolved

18.a - Description

Even though the team solved GH-302, they also introduced a new instance of the same problem while solving other findings. In this case, all validation paths in the SettingsValidator contain the `is_signed` check to ensure the GitHoney's Address signed the transaction. Needlessly increasing the size of the validator.

18.b - Recommendation

Move the `is_signed` check to the top of the validator (where checks are shared by all paths) and remove it from all individual validation paths.

18.c - Resolution

Resolved in commit `206cf28355b29e411383b12df204c22c973b97f1`

19 - GH-312 Implicit check

Category	Commit	Severity	Status
Optimization	df73a658d1d89b1c24126dabc0d8cb5f14f9c62a	Enhancement	Resolved

19.a - Description

The `get_unique_script_input()` function has two checks inside it:
- The transaction has only script input
- That script input has the same `output_reference` as the UTxO we're currently validating.

This is not technically wrong. However, by checking that the transaction has only one script input and knowing that the current validator is running, we know that this script UTxO input has the same `output_reference` as the UTxO we're currently validating. So, having this extra check unnecessarily increases the cost of the transaction.

19.b - Recommendation

Remove this expression:

```
if input.output_reference == output_reference {
    input
} else {
    fail @"The input does not coincide with the output reference"
}
```

From the `get_unique_script_input()` function, return `input`. Then, remove and ignore the `input_ref` arguments since you're not using them anymore.

19.c - Resolution

Resolved in commit `9c90f222f7ff5d23eeeb63d5b665e08038a84e63`

20 - GH-401 Spring Cleaning

Category	Commit	Severity	Status
Robustness	18997a07c0af182d41899e8f02e062dbffcb682b	Info	Resolved

20.a - Description

All these functions are not used in the codebase and make it harder to read/analyze:
- `checks.ak:is_tx_after_deadline()`
- `utils.ak:is_script_utxo()`
- `utils.ak:get_script_output()`
- `utils.ak:is_output_script_utxo()`
- `utils.ak:get_input_with_asset()`

Notes:
- This finding used to be identified as GH-313 when it was fixed by the project team in PR-109.

20.b - Recommendation

Remove all unused functions.

20.c - Resolution

Resolved in commit `bc18bbe0a54ae2f897997bcb9aedffb70d70f631`

21 - GH-402 404 Not Found

Category	Commit	Severity	Status
Robustness	18997a07c0af182d41899e8f02e062dbffcb682b	Info	Resolved

21.a - Description

In `minting_policy_settings()`, we never check if the output address corresponds with the Settings UTxO validator. This is not a major issue since the protocol owner is the one creating it. And if it fails to create it properly, it can try again without repercussions. However, it somewhat undermines the checks related to `script_output` since one could create a perfectly good Datum and Value, but at the wrong address.

21.b - Recommendation

We recommend checking that the output address corresponds to the Settings UTxO validator.

21.c - Resolution

This finding was resolved in commit `c0fcd2b1cdb60d7dc304cb0e406a39d5465897af`. However, **it created GH-003** by doing so.

22 - GH-403 Misleading message

Category	Commit	Severity	Status
Robustness	18997a07c0af182d41899e8f02e062dbffcb682b	Info	Resolved

22.a - Description

In `validations.ak:48`, the `fail` message reads `"There should be exactly one bounty id token in the inputs"`. However, that's a wrong statement! Between all the checks inside `script_input_and_token_id_info()` function, we can only assert that there isn't one BountyIdToken **in this UTxO's value**. But there could be more BountyIdTokens in other inputs!

22.b - Recommendation

We recommend checking that only one Bounty UTxO is being consumed in the transaction, making the `fail` message correct. However, another solution could be changing the `fail` message to `"There should be exactly one bounty id token in this input"`.

22.c - Resolution

Resolved in commit `c0fcd2b1cdb60d7dc304cb0e406a39d5465897af`

23 - GH-405 Dirty boots

Category	Commit	Severity	Status
Robustness	c0fcd2b1cdb60d7dc304cb0e406a39d5465897af	Info	Resolved

23.a - Description

While solving GH-001, the project team introduced a new unused type: `types.ak:TokenInfo`. This type is not used anywhere in the codebase.

23.b - Recommendation

Remove the `TokenInfo` type.

23.c - Resolution

Resolved in commit `206cf28355b29e411383b12df204c22c973b97f1`

24 - GH-406 Why are you hitting yourself?

Category	Commit	Severity	Status
Robustness	c0fcd2b1cdb60d7dc304cb0e406a39d5465897af	Info	Resolved

24.a - Description

When consuming the Settings UTxO for any reason (updating or deleting the UTxO), the SettingsValidator checks that the value contains only one SettingsNFT and any amount of Lovelace. However, regarding the value of the **new Settings UTxO** when updating the settings, the SettingsValidator only checks if the value contains the SettingsNFT.

This means that GitHoney could (likely by mistake) add a new native token on top of the SettingsNFT to the new Settings UTxO, rendering that UTxO unspendable and forever locking the settings of that protocol instance.

The repercussions of this mistake are low: The worst-case scenario is that GitHoney loses whatever is locked inside that Settings UTxO and has to create a new instance if they want to have different settings. However, the protocol users will still be able to use it if they are OK with the current settings, and if they aren't, they can slowly transition from the old instance to the new one.

24.b - Recommendation

Change the logic for the udpating the settings transaction to check that the new Settings UTxO contains the same value than the Settings UTxO being consumed in the same transaction

24.c - Resolution

Resolved in commit `3f8ad67e22c1ddaf889bcd3f57cd41f6836f66f3`

25 - Appendix

25.a - Terms and Conditions of the Commercial Agreement

This Smart Contract Audit Report ("Report") is provided on an "as is" basis, for informational purposes only, and is not intended to constitute financial, investment, tax, legal, regulatory, or any other form of advice. The entities and individuals involved in preparing this Report ("Auditors") do not guarantee the accuracy, completeness, or usefulness of the information provided herein and shall not be held liable for any contents, errors, omissions, or inaccuracies in this Report or for any actions taken in reliance thereon.

25.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 the Auditors' written consent.

The violation of the aforementioned, as stated supra, shall empower the Auditors to pursue all of its rights and claims by any applicable law that protects or upholds these rights.

Therefore, in the event of any harm inflicted upon the Auditors' reputation or resulting from the misappropriation of trade secrets, the Auditors hereby reserve the right to initiate legal action against the client 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.

25.a.b - Service Extension and Details

This report does not endorse or disapprove of 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 to make use of the logo, name, or any other unauthorized reference to the Auditors except upon express authorization from the Auditors.

The Auditors 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 Auditors 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 Auditors 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.

Any conflict or controversy arising under this commercial agreement or subsequent agreements shall be resolved in good faith between the parties.

25.a.c - Disclaimer

The audit constitutes a comprehensive examination and assessment as of the date of report submission. The Auditors expressly disclaim 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 Auditors explicitly disclaim 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 the sha256 algorithm:

Filename: onchain/validators/githoney_contract.ak **Hash**: 885a97061a3fe5321cb948abd62eb940004e1cdf4a4f8f7e589d46d4b70a3519
Filename: onchain/lib/checks.ak **Hash**: ad0b29850fee7d6e9438a5e12fd57fc08458d2d15b790da2e06a129703194c4e
Filename: onchain/lib/types.ak **Hash**: d2233e4fd098961a7810b0fdc052a29d3abdfa7ed33cacf16a48965db3f2dff7
Filename: onchain/lib/utils.ak **Hash**: 7fff22fc7ad37aa73bc4aa3e2b2bf1bfa1a7256c4eb53975c7330c3e91523dd2
Filename: onchain/lib/validations.ak **Hash**: eb1f0a23d3d52f9619631b52af66c8d11631e158b5abd197e52fccdcdc242aae

The Auditors advocate 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. The Auditors strongly advise seeking multiple independent opinions on critical matters. It is the firm belief of the Auditors that every entity and individual is responsible for conducting their own due diligence and maintaining ongoing security measures.

25.b - Issue Guide

25.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, often leading to a disadvantage for the user.
Enhancement	Enhancement issues highlight changes to the protocol design or implementation that could improve performance, usability, or overall security. These are not bugs or vulnerabilities but rather suggestions for enhancement.
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.

25.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.

25.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.

25.d - About Us

AlwaysTrue is a blockchain technology company focused on the Cardano ecosystem. We specialize in smart contract security and optimization audits and are dedicated to ensuring safe, efficient, and scalable decentralized applications. Our expert team combines years of development experience in Cardano to deliver tailored solutions.

25.d.a - Links

- Website
- Email
- Twitter