CODESPECT

Solana Unlocker-V2

SECURITY ASSESSMENT REPORT

26 March, 2025

Prepared for

TOKENTABLE

Contents

1 About CODESPECT

CODESPECT is a specialized smart contract security firm dedicated to ensure the safety, reliability, and success of blockchain projects. Our services include comprehensive smart contract audits, secure design and architecture consultancy, and smart contract development across leading blockchain platforms such as Ethereum (Solidity), Starknet (Cairo), and Solana (Rust).

At CODESPECT, we are committed to build secure, resilient blockchain infrastructures. We provide strategic guidance and technical expertise, working closely with our partners from concept development through deployment. Our team consists of blockchain security experts and seasoned engineers who apply the latest auditing and security methodologies to help prevent exploits and vulnerabilities in your smart contracts.

Smart Contract Auditing: Security is at the core of everything we do at CODESPECT. Our auditors conduct thorough security assessments of smart contracts written in Solidity, Cairo, and Rust, ensuring that they function as intended without vulnerabilities. We specialize in providing tailored security solutions for projects on EVM-compatible chains and Starknet. Our audit process is highly collaborative, keeping clients involved every step of the way to ensure transparency and security. Our team is also dedicated to cutting-edge research, ensuring that we stay ahead of emerging threats.

Secure Design & Architecture Consultancy: At CODESPECT, we believe that secure development begins at the design phase. Our consultancy services offer deep insights into secure smart contract architecture and blockchain system design, helping you build robust, secure, and scalable decentralized applications. Whether you're working with Ethereum, Starknet, or other blockchain platforms, our team helps you navigate the complexity of blockchain development with confidence.

Tailored Cybersecurity Solutions: CODESPECT offers specialized cybersecurity solutions designed to minimize risks associated with traditional attack vectors, such as phishing, social engineering, and Web2 vulnerabilities. Our solutions are crafted to address the unique security needs of blockchain-based applications, reducing exposure to attacks and ensuring that all aspects of the system are fortified.

With a focus on the intersection of security and innovation, CODESPECT strives to be a trusted partner for blockchain projects at every stage of development and for each aspect of security.

2 Disclaimer

Limitations of this Audit: This report is based solely on the materials and documentation provided to CODESPECT for the specific purpose of conducting the security review outlined in the Summary of Audit and Files. The findings presented in this report may not be comprehensive and may not identify all possible vulnerabilities. CODESPECT provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, is entirely at your own risk.

Inherent Risks of Blockchain Technology: Blockchain technology is still evolving and is inherently subject to unknown risks and vulnerabilities. This review focuses exclusively on the smart contract code provided and does not cover the compiler layer, underlying programming language elements beyond the reviewed code, or any other potential security risks that may exist outside of the code itself.

Purpose and Reliance of this Report: This report should not be viewed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. Third parties should not rely on this report for any purpose, including making decisions related to investments or purchases.

Liability Disclaimer: To the maximum extent permitted by law, CODESPECT disclaims all liability for the contents of this report and any related services or products that arise from your use of it. This includes but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services: CODESPECT does not warrant, endorse, or assume responsibility for any third-party products or services mentioned in this report, including any open-source or third-party software, code, libraries, materials, or information that may be linked to, referenced by, or accessible through this report. CODESPECT is not responsible for monitoring any transactions between you and third-party providers. We strongly recommend conducting thorough due diligence and exercising caution when engaging with third-party products or services, just as you would for any other product or service transaction.

Further Recommendations: We advise clients to schedule a re-audit after any significant changes to the codebase to ensure ongoing security and reduce the risk of newly introduced vulnerabilities. Additionally, we recommend implementing a bug bounty program to incentivize external developers and security researchers to identify and disclose potential vulnerabilities safely and responsibly.

Disclaimer of Advice: FOR AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, AND ANY ASSOCIATED SERVICES OR MATERIALS SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

3 Risk Classification

Severity Level	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Table 1: Risk Classification Matrix based on Likelihood and Impact

3.1 Impact

- **High** - Results in a substantial loss of assets (more than 10%) within the protocol or causes significant disruption to the majority of users.
- **Medium** - Losses affect less than 10% globally or impact only a portion of users, but are still considered unacceptable.
- **Low** - Losses may be inconvenient but are manageable, typically involving issues like griefing attacks that can be easily resolved or minor inefficiencies such as gas costs.

3.2 Likelihood

- **High** - Very likely to occur, either easy to exploit or difficult but highly incentivized.
- **Medium** - Likely only under certain conditions or moderately incentivized.
- **Low** - Unlikely unless specific conditions are met, or there is little-to-no incentive for exploitation.

3.3 Action Required for Severity Levels

- **Critical** - Must be addressed immediately if already deployed.
- **High** - Must be resolved before deployment (or urgently if already deployed).
- **Medium** - It is recommended to fix.
- **Low** - Can be fixed if desired but is not crucial.

In addition to High, Medium, and Low severity levels, CODESPECT utilizes two other categories for findings: **Informational** and **Best Practices**.

a) **Informational** findings do not pose a direct security risk but provide useful information the audit team wants to communicate formally.

b) **Best Practices** findings indicate that certain portions of the code deviate from established smart contract development standards.

4 Executive Summary

This document presents the security assessment conducted by CODESPECT for the Unlocker Solana programs of TokenTable. Unlocker is part of a larger suite of protocols designed to streamline token ownership registration and distribution.

This audit focuses on the review of two new Solana programs, which allow users to unlock token distribution for a smaller group of recipients, such as investors or development teams. It offers unique advantages such as unruggability and complete decentralization.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of programs, execution testing.

CODESPECT found 15 points of attention, four classified as `Medium`, five classified as `Low`, two classified as `Informational` and four classified as `Best Practices`. All of the issues are summarised in Table 2.

Organization of the document is as follows:

- **Section 5** summarizes the audit.
- **Section 6** describes the system overview.
- **Section 7** presents the issues.
- **Section 8** discusses the documentation provided by the client for this audit.
- **Section 9** presents the compilation and tests.

Issues found:

Severity	Unresolved	Fixed	Acknowledged
Medium	0	4	0
Low	0	5	0
Informational	0	2	0
Best Practices	0	3	1
Total	**0**	**14**	**1**

Table 2: Summary of Unresolved, Fixed, and Acknowledged Issues

5 Audit Summary

Audit Type	Security Review
Project Name	TokenTable
Type of Project	Token Unlocker
Duration of Engagement	10 Days
Duration of Fix Review Phase	2 Days
Draft Report	March 25, 2025
Final Report	March 26, 2025
Repository	tokentable-unlocker-solana
Commit (Audit)	7516b8c86cb305f9d9eb3ac77e7fcd7c6b60cc2f
Commit (Final)	22e51755ffe9cf1ca3a3e1a5c56bef2d80a618aa
Documentation Assessment	Medium
Test Suite Assessment	Medium
Auditors	JecikPo, shaflow01

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	File	LoC
0	unlocker-v2-solana/src/events.rs	55
1	unlocker-v2-solana/src/state/claiming_delegate.rs	6
2	unlocker-v2-solana/src/state/mod.rs	8
3	unlocker-v2-solana/src/state/config.rs	6
4	unlocker-v2-solana/src/state/unlocker.rs	15
5	unlocker-v2-solana/src/state/misc.rs	6
6	unlocker-v2-solana/src/constants.rs	3
7	unlocker-v2-solana/src/instructions/deploy.rs	22
8	unlocker-v2-solana/src/instructions/transfer_program_admin.rs	19
9	unlocker-v2-solana/src/instructions/disable_withdraw.rs	17
10	unlocker-v2-solana/src/instructions/transfer_ownership.rs	20
11	unlocker-v2-solana/src/instructions/transfer_actual.rs	36
12	unlocker-v2-solana/src/instructions/set_claiming_delegate.rs	33
13	unlocker-v2-solana/src/instructions/disable_cancel.rs	17
14	unlocker-v2-solana/src/instructions/renounce_ownership.rs	16
15	unlocker-v2-solana/src/instructions/utils.rs	123
16	unlocker-v2-solana/src/instructions/withdraw_deposit.rs	61
17	unlocker-v2-solana/src/instructions/delegate_claim.rs	195
18	unlocker-v2-solana/src/instructions/disable_create.rs	17
19	unlocker-v2-solana/src/instructions/mod.rs	40
20	unlocker-v2-solana/src/instructions/initialize.rs	40
21	unlocker-v2-solana/src/instructions/cancel.rs	79
22	unlocker-v2-solana/src/instructions/disable_transfer_actual.rs	20
23	unlocker-v2-solana/src/instructions/set_fee_token.rs	22
24	unlocker-v2-solana/src/instructions/create_actual.rs	78
25	unlocker-v2-solana/src/instructions/create_preset.rs	97
26	unlocker-v2-solana/src/instructions/claim.rs	316
27	unlocker-v2-solana/src/instructions/deposit.rs	51
28	unlocker-v2-solana/src/errors.rs	40
29	unlocker-v2-solana/src/lib.rs	157
30	unlocker-v2-solana/src/models/preset.rs	63
31	unlocker-v2-solana/src/models/mod.rs	4
32	unlocker-v2-solana/src/models/actual.rs	27
33	fee-collector/src/traits/mod.rs	0
34	fee-collector/src/state/constants.rs	2
35	fee-collector/src/state/mod.rs	2
36	fee-collector/src/instructions/withdraw.rs	68
37	fee-collector/src/instructions/set_custom_fee_fixed.rs	40
38	fee-collector/src/instructions/init_fee_token.rs	31
39	fee-collector/src/instructions/transfer_ownership.rs	15
40	fee-collector/src/instructions/set_default_fee.rs	28
41	fee-collector/src/instructions/renounce_ownership.rs	15
42	fee-collector/src/instructions/utils.rs	58
43	fee-collector/src/instructions/set_custom_fee_bips.rs	42
44	fee-collector/src/instructions/mod.rs	22
45	fee-collector/src/instructions/initialize.rs	21
46	fee-collector/src/instructions/get_fee.rs	31
47	fee-collector/src/instructions/collect_fee.rs	90
48	fee-collector/src/errors.rs	12
49	fee-collector/src/lib.rs	56
50	fee-collector/src/models/fee.rs	7
51	fee-collector/src/models/mod.rs	4
52	fee-collector/src/models/fee_collector_storage.rs	7
53	fee-collector/src/event.rs	15
	Total	**2275**

5.2 Findings Overview

	Finding	Severity	Update
1	Arithmetic overflow in `claim` instruction	Medium	Fixed
2	Collision between `PendingAmountClaimableForCancelledActualsAccount` may lead to stolen funds	Medium	Fixed
3	Incorrect `Preset` input parameters validation	Medium	Fixed
4	The `pending_amount_claimable` accumulated in the `cancel` instruction cannot be claimed	Medium	Fixed
5	Claims fail when different Token program is used for fees and claimable tokens	Low	Fixed
6	Lack of `token_mint` validation in the `Deposit` instruction	Low	Fixed
7	Missing check whether `fee_token` and `token_mint` are consistent in the `init_fee_token` instruction	Low	Fixed
8	Missing the `is_withdrawable` check in the `withdraw_deposit(...)` instruction	Low	Fixed
9	Withdrawals from Fee Collector become impossible after renouncing ownership	Low	Fixed
10	Miscalculated `PresetAccount` size	Info	Fixed
11	Unnecessary account ownership validation	Info	Fixed
12	In some cases the `num_of_unlocks_for_each_linear` vector may waste rent	Best Practices	Fixed
13	Lack of two step ownership transfers	Best Practices	Acknowledged
14	Missing check for `fee_collector` in `Unlocker` account initialization	Best Practices	Fixed
15	Missing check to verify if the fields stored in the `Account` are consistent with the seed	Best Practices	Fixed

6 System Overview

TokenTable introduces two new Solana programs which work in tandem as an independent on-chain system to provide users with token distribution capabilities:

a) **Unlocker** program – used to store distributed assets and all the records necessary for distribution.

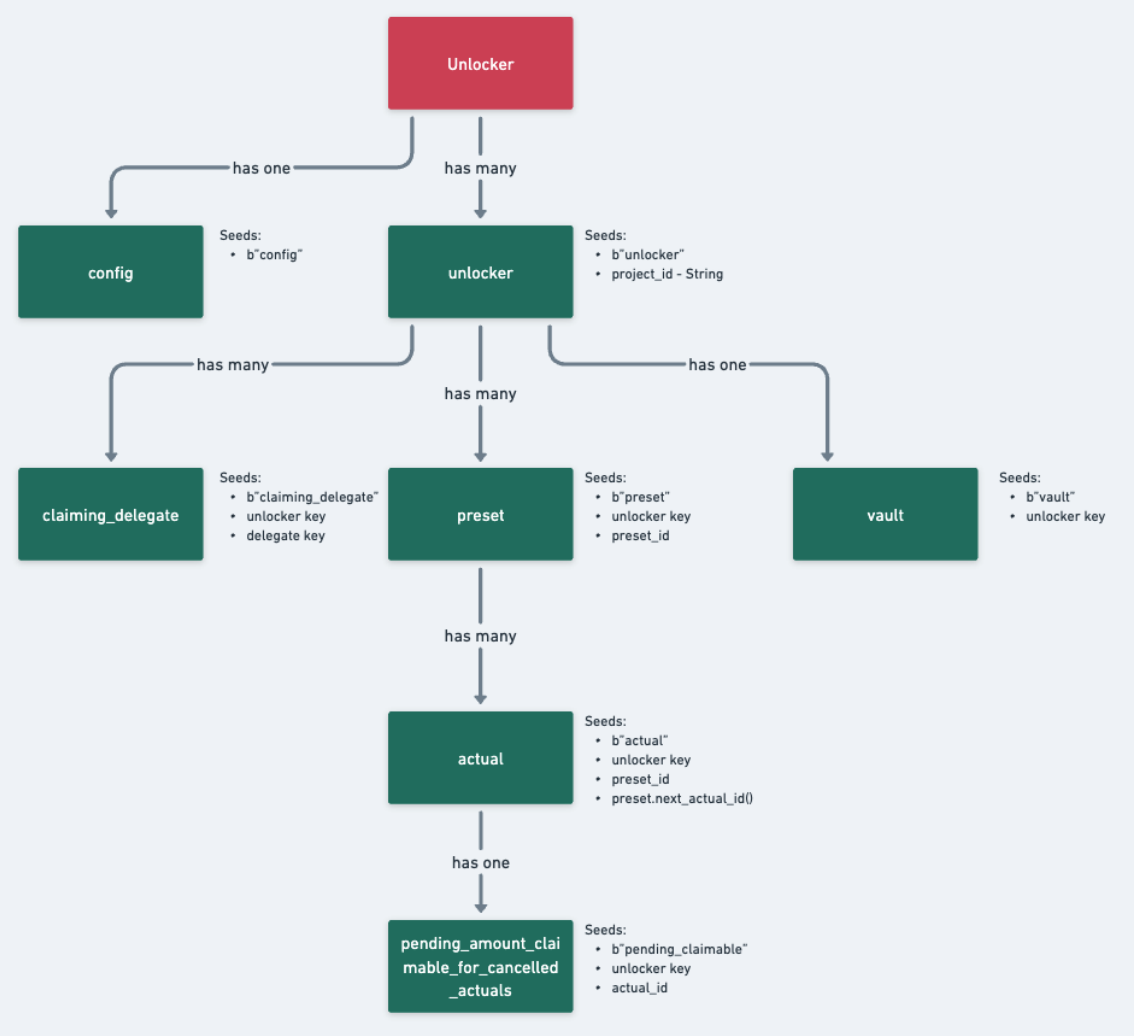
b) **Fee Collector** program – responsible for collecting protocol fees.

6.1 Unlocker

In the Unlocker program, the owner of the protocol creates an `unlocker` account a.k.a. Project (the protocol is not permissionless) that is a basis for a redistribution of a single asset. The following points describe a high-level view of a Project:

1. The protocol owner creates a Project (represented by a unique `unlocker` account). Permission is granted to that Project to a user.
2. The user creates Presets that represent the distribution schedule and Actuals that define recipients and their amounts.
3. A Recipient can claim the tokens accordingly.

The following picture presents the account structure of the Unlocker program:



The section below outlines the program's instructions, categorized by the entities authorized to call them.

Unlocker's instructions executable by the protocol owner:

1. `deploy` – sets the protocol owner account which is held within the `config`. Can be executed only once.
2. `initialize` – creates a new `unlocker` (a Project) and sets its owner.
3. `set_fee_token` – sets the `fee_token` for a given Project.
4. `transfer_program_admin` – transfer the protocol's ownership to a different account.

Unlocker's instructions executable by the Project owner:

1. `create_preset` - creates a `preset` account which describes the claiming schedule. Multiple Presets can be created.
2. `create_actual` - creates an `actual` account which holds claiming data relevant to a single Recipient.
3. `set_claiming_delegate` - A Project owner can delegate the claiming capability through this instruction. It creates an additional account `claiming_delegator` which holds the delegatee account and its status. A delegatee can claim the token and pay the fees on behalf of all Recipients. A Project owner can also revoke the delegatee status.
4. `cancel` - Cancels a given Recipient's `actual` account. It gives an option to let the Recipient claim the amount accrued till cancellation.
5. `disable_cancel` – Disables `cancel` instruction. Change is irreversible for the given Project.
6. `disable_create` – Disables `create_actual` instruction. Change is irreversible for the given Project.
7. `disable_transfer_actual` – Disables `transfer_actual` instruction. Change is irreversible for the given Project.
8. `disable_withdraw` – Disables `withdraw_deposit` instruction. Change is irreversible for the given Project.
9. `renounce_ownership` – The Owner of a Project can renounce ownership, this will effectively disable all owner's instructions. Change is irreversible for the given Project.
10. `transfer_ownership` – Owner transfers ownership to a new account.
11. `withdraw_deposit` – Allows the Owner of the Project to withdraw deposited tokens.

Unlocker's instructions executable by a Recipient or Delegatee:

1. `claim` - claims the tokens based on the Recipient's `actual` account and a `preset` account where it belongs.
2. `delegate_claim` – A Delegatee can claim tokens on behalf of the Recipient. Delegatee pays the fees.
3. `transfer_actual` – A Recipient can transfer their Actual (and hence all future claims) to a new Recipient.

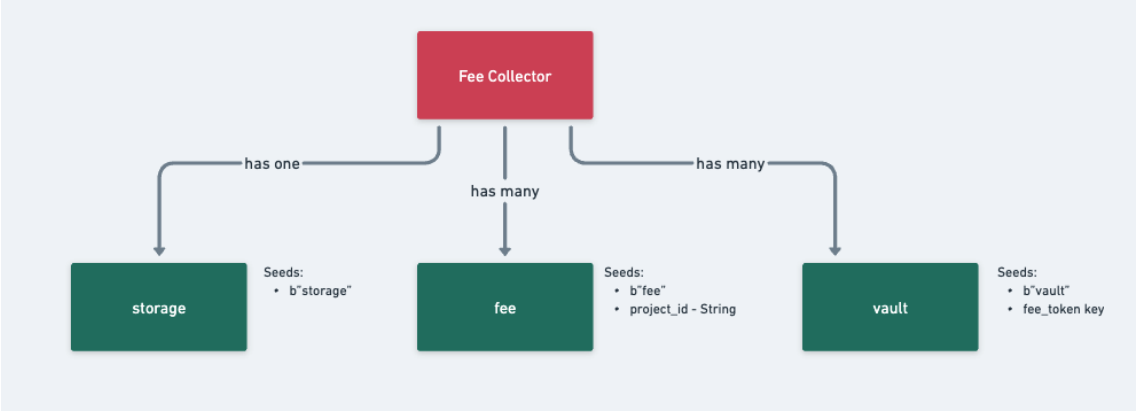
Unlocker's instructions executable by anyone:

1. `deposit` – Deposits a specified amount of tokens into the Project.

6.2 Fee Collector

The Fee Collector program is controlled only by the protocol owner. It is used to create specific `fee` accounts that hold fee rates for each Project. This program is also responsible for the correct calculation and collection of fees. The fees collected are stored in `vault` accounts owned by the program. Protocol owner can withdraw collected fees.

The following picture presents the account structure of the Fee Collector program:



The `storage` and `config` accounts are used to store the protocol's owner account and other global settings.

Fee Collector's instructions:

1. `collect_fees` – permissionless instruction to collect a fee - transfer it from the caller's account to the Fee Collector's `vault` account. Can be executed by anyone, but really designed to be CPIed from the Unlocker program.
2. `get_fee` – a read-only instruction to simulate fee based on the claimed amount.
3. `init_fee_token` – initializes the `vault` account for fee collection. Only the protocol owner can call this instruction.
4. `initialize` – sets the protocol owner. Can be called only once.
5. `renounce_ownership` – renounces ownership of the protocol. Only executable by the protocol owner.
5. `set_custom_fee_bips` – sets the fee percentage for a given `fee` account. Only executable by the protocol owner.
5. `set_custom_fee_fixed` – sets the fixed fee amount for a given `fee` account. Only executable by the protocol owner.
5. `set_default_fee` – sets the default fee in the `storage` account.
5. `transfer_ownership` – transfers ownership of the protocol to a new account. Only executable by the protocol owner.
5. `withdraw` – Withdraws collected fees from a given `vault` account. Callable by the protocol owner only.

The Fee Collector takes the following precedence to calculate the fee:

1. The fixed fee is taken from `fee.fixed`.
2. If the above-fixed fee is set to zero, the percentage-based fee is taken from `fee.bips`.
3. If the percentage-based fee is set to zero, the default fee is taken from `storage.default_fees_bips`
4. If the default fee is set to zero, no fees are charged.

The fee can be charged using either the fee token specified for a given `unlocker` account or, if it is not set, using SOL.

7 Issues

7.1 [Medium] Arithmetic overflow in `claim` instruction

File(s): `claim.rs`

Description: The `claim` instruction uses the internal function `_simulate_amount_claimable()` to calculate the amount of claimable tokens based on given `actual` and `preset` accounts.

When calculating `updated_amount_claimed`, which represents the new amount of claimed tokens, it multiplies two values, which are both 9 decimals big:

```
updated_amount_claimed =
  (updated_amount_claimed * actual.total_amount) / BIPS_PRECISION / TOKEN_PRECISION;
```

Provided that the `total_amount` is big enough, the calculation may overflow as it needs to fit into the `u64` variable size before division.

Impact: Under certain big enough values representing token amounts, claims will fail

Recommendation(s): Make the calculations on at least `u128` variable sizes.

Status: Fixed

Update from TokenTable: Switched to `u128` for calculations in d7357087b3a46d0c6eed3c240239d35d2f7ddc15.

7.2 [Medium] Collision between `PendingAmountClaimableForCancelledActualsAccount` may lead to stolen funds

File(s): `claim.rs`

Description: The `PendingAmountClaimableForCancelledActualsAccount` account is used to store amount assets that a recipient of a cancelled Actual can still claim. This account is provided to the `claim` instruction, and a recipient should be able to claim it.

The problem is that the account seeds don't contain the `preset_id` where the claim was present. Hence the following scenario is possible:

1. Unlocker is created with two Presets;
2. For each Preset one Actual is created: Actual-1 and Actual-2, for recipient-1 and recipient-2. As those Actuals are created for different Presets, they share the same `actual_id` (e.g. 1);
3. Unlocker's owner cancels Actual-1 and hence `PendingAmountClaimableForCancelledActualsAccount` is created using `actual_id` 1 as its seed;
4. The recipient-2 can call claim, and provide the above `PendingAmountClaimableForCancelledActualsAccount` account to claim the amount stored there, which does not belong to him;

Impact: Cancelled amounts can be claimed by other recipients.

Recommendation(s): Add the `preset_id` to the `PendingAmountClaimableForCancelledActualsAccount` seed.

Status: Fixed

Update from TokenTable: Added `preset_id` to the list of seeds used when deriving `PendingAmountClaimableForCancelledActualsAccount` in 3510bac21534d846d7f117ac89b11d83478a529f.

7.3 [Medium] Incorrect `Preset` input parameters validation

File(s): `create_preset.rs`

Description: The `create_preset` instruction is called to create a `PresetAccount` account which holds the airdrop schedule information. The instruction caller provides a `Preset` struct which is used to populate values for the `PresetAccount`. The input `Preset` struct is validated using `_preset_has_valid_format()` function. The function ensures the following:

- `preset.linear_bips` vector sum of values is equal to `BIPS_PRECISION` const;
- All vectors must be of the same length;
- The `preset.linear_start_timestamps_relative` vector last element must be smaller than `preset.linear_end_timestamp_relative`;
- Each `preset.num_of_unlocks_for_each_linear` element value must be smaller than relative timestamp spacing;

The first three conditions from the above list are enclosed within the following conditional instruction:

```
if
  !(total == BIPS_PRECISION) &&
  preset.linear_bips.len() == preset.linear_start_timestamps_relative.len() &&
  preset.linear_start_timestamps_relative[preset.linear_start_timestamps_relative.len() - 1] <
    preset.linear_end_timestamp_relative &&
  preset.num_of_unlocks_for_each_linear.len() == preset.linear_start_timestamps_relative.len()
{
  return false;
}
```

We can see that the logical operator `!` is not applied correctly, and hence it is possible that all the remaining checks after BIPS verification can be bypassed.

Impact: A `preset` account couldbe created with incorrect values. Depending on which specific values are incorrect (several are possible), it may become impossible to claim from the affected `Preset`.

Recommendation(s): Fix the logical expression.

Status: Fixed

Update from TokenTable: Fixed the parenthesis location in c5a96b7c73e0b37678436b4bce9adf9e3c8bf78f.

7.4 [Medium] The `pending_amount_claimable` accumulated in the `cancel` instruction cannot be claimed

File(s): `cancel.rs`

Description: In the `cancel()` instruction, if `should_wipe_claimable_balance` is false, the unclaimed rewards of the closed `ActualAccount` will be accumulated into the `PendingAmountClaimableForCancelledActualsAccount` account.

```
fn _cancel(
  ctx: Context<Cancel>,
  actual_id: u64,
  should_wipe_claimable_balance: bool,
  batch_id: u64
) -> Result<()> {
  let delta_amount_claimable = _calculate_amount_claimable(
    ctx.accounts.actual.clone(),
    ctx.accounts.preset.clone()
  )?.delta_amount_claimable;

  if !should_wipe_claimable_balance {
    ctx.accounts.pending_amount_claimable_for_cancelled_actuals.pending_amount_claimable_for_cancelled_actuals +=
      delta_amount_claimable;
  }
  //...
}
```

However, the user is unable to claim the accumulated balance in the `PendingAmountClaimableForCancelledActualsAccount` because the balance can only be claimed through the `claim()` and `delegate_claim()` instructions, both of which require the `ActualAccount` that has been closed by the `cancel()` instruction. Additionally, the corresponding `ActualAccount` cannot be re-init.

Impact: The user may lose the unclaimed tokens from before the execution of the `cancel` instruction.

Recommendation(s): It is recommended to separate the logic for claiming the tokens accumulated in `PendingAmountClaimableForCancelledActualsAccount` from the `claim()` and `delegate_claim()` instructions.

Status: Fixed

Update from TokenTable: Restructured the claiming process in f851e215e19904ea9a1d07cfa44b9cc34afce11e. `delegate_claim()` has been combined into `claim()` with logic changes pertaining to `authority`, `recipient`, and `recipient_ata` accounts. Split `claim()` logic into `claim()` and `claim_cancelled_actual_tokens()` in order to allow claiming the previously orphaned tokens present in `PendingAmountClaimableForCancelledActualsAccount`.

7.5 [Low] Claims fail when different Token program is used for fees and claimable tokens

File(s): `collect_fee.rs` `claim.rs`

Description: The claim instruction is used to make two token transfers:

- The claimable token to the recipient from the `unlocker` vault;
- The fee token from the recipient to the protocol fee-collector vault;

The problem is that in the claim instruction, those two transfers are handled by a single program account specified in the instruction context. If those two tokens are of two different kinds, i.e. Token and Token2022, then it will be impossible to claim the tokens as the transfer of the other token will fail due to an incorrect program applied in the CPI.

Impact: A certain token/fee-token combination won't be possible if they are of two different types. The protocol owner can always adjust the fee-token to match the token in an already set up `unlocker`.

Recommendation(s): Provide two Token Program account inputs in the `claim` context definition. One for the distributed token and the other for the fee token. The CPI for the `collect_fee` should use the fee token program account.

Status: Fixed

Update from TokenTable: Added `fee_token_program` account to `claim()` and `claim_cancelled_actual_tokens()` which is used in the CPI to the fee collector program in 22e51755ffe9cf1ca3a3e1a5c56bef2d80a618aa.

7.6 [Low] Lack of `token_mint` validation in the `Deposit` instruction

File(s): `deposit.rs`

Description: The `Deposit` instruction in the `Unlocker` program is used by the owner of the `unlocker` account to deposit the initial amount of tokens for later claiming. When this instruction is called for the first time a new `vault` account is created for a specified `token_mint` account to hold the assets tied to the specific `unlocker` account (identified by `_project_id`). The Deposit instruction can be called by anyone as there is no validation of the owner.

A malicious user could call the Deposit instruction and provide a junk `token_mint` account and hence a `vault` tied to that `unlocker` will be created. Further deposits of the intended SPL token become impossible because the `vault` is already created and the entire `unlocker` becomes useless.

Impact: Malicious user could DoS a created `unlocker` account before the initial funds are deposited. No funds are lost, but another `unlocker` needs to be created for the owner by the program's owner.

Recommendation(s): Validate the provided `token_mint` against the `unlocker.project_token` either through Anchor context definition (recommended for clarity) or inside the handler.

Status: Fixed

Update from TokenTable: Added `token_mint` constraint in 6bccb7d0ebefd3cdaaa82278c9b0567c4f01f13e.

7.7 [Low] Missing check whether `fee_token` and `token_mint` are consistent in the `init_fee_token` instruction

File(s): `init_fee_token.rs`

Description: In the `init_fee_token` instruction, there is no check to ensure that `fee_token` and `token_mint` are consistent. This may result in the vault account's seed not matching the corresponding mint. Causing the vault account to be unable to properly collect fees.

```
pub struct InitFeeToken<'info> {
  //...
  #[account(
    init_if_needed,
    payer = authority,
    seeds = [b"vault".as_ref(), fee_token.as_ref()],
    bump,
    token::mint = token_mint,
    token::authority = storage
  )]
  pub vault: Option<InterfaceAccount<'info, TokenAccount>>,
  pub token_mint: Option<InterfaceAccount<'info, Mint>>,
  //...
}
```

Impact: When `fee_token` and `token_mint` are inconsistent, the vault account created by the `init_fee_token` instruction will be unable to properly receive fees.

Recommendation(s): It is recommended to check that `fee_token` and `token_mint` are the same.

Status: Fixed

Update from TokenTable: Added constraint to `token_mint` in eb09b6b3d1da4774d72cd75b99e7eb9cf86bb7e9.

7.8 [Low] Missing the `is_withdrawable` check in the `withdraw_deposit(...)` instruction

File(s): `withdraw_deposit.rs`

Description: In the `Unlocker account`, `is_withdrawable` is set to control whether the unlocker owner is allowed to withdraw tokens from the vault. However, the `withdraw_deposit()` instruction does not check the `is_withdrawable` field, causing the unlocker owner's withdrawals to be always allowed.

Impact: The unlocker owner's withdrawals will not be controlled by the `is_withdrawable` field.

Recommendation(s): Add the `is_withdrawable` check in the `withdraw_deposit()` instruction.

Status: Fixed

Update from TokenTable: Added `is_withdrawable` check to the `withdraw_deposit()` function in commit e8521cef.

7.9 [Low] Withdrawals from Fee Collector become impossible after renouncing ownership

File(s): `renounce_ownership.rs`

Description: The `renounce_ownership` instruction in the Fee Collector program is available for the owner. The following instructions become impossible to call when ownership is renounced:

- `init_fee_token`;
- `set_custom_fee_bips`;
- `set_custom_fee_fixed`;
- `set_default_fee`;
- `transfer_ownership`;
- `withdraw`;

There are two major consequences to the protocol when ownership is renounced:

- The whole protocol will operate normally for existing `unlockers`, however, new ones could not have their own `fee` accounts, hence all fees will fallback to the default fee: `storage.default_fees_bips`;
- It will be impossible to withdraw fees;

Impact: Withdrawals shall be blocked if ownership is renounced.

Recommendation(s): Specify what the ownership renouncement should block from happening or remove the instruction for safety reasons.

Status: Fixed

Update from TokenTable: Removed `renounce_ownership()` in e8c372dbee51e66a152f6683d002dbc2e297a07b.

7.10 [Info] Miscalculated `PresetAccount` size

File(s): `preset.rs`

Description: When creating a `PresetAccount`, the size of the `PresetAccount` is initialized based on the `Preset` struct passed in, using the `calculate_size()` function.

```
#[account(
  //...
  space = 8 + _preset.clone().calculate_size()
)]
pub preset: Account<'info, PresetAccount>,
```

The calculation seems to be implemented incorrectly, as the total size of non-vector items should be `linear_end_timestamp_relative` + `next_actual_id` + `stream` + `preset_id` = 25 instead of 32.

```
pub fn calculate_size(self) -> usize {
  let mut size = 0;
  // @audit incorrect size
  size += 32; // Takes care of all non-vector items
  size += 4 + 8 * self.linear_start_timestamps_relative.len();
  size += 4 + 8 * self.linear_bips.len();
  size += 4 + 8 * self.num_of_unlocks_for_each_linear.len();
  size += 4 + self.project_id.len();

  size
}
```

Impact: This would result in unnecessary rent wastage.

Recommendation(s): Calculate account space using the correct size.

Status: Fixed

Update from TokenTable: Updated base size to `25` in 85e56b4993b46006e5cb36e08df56f49ac4a535e.

7.11 [Info] Unnecessary account ownership validation

File(s): `initialize.rs` `deploy.rs`

Description: Initialize and Deploy instructions are used the create new accounts: `unlocker` and `config` respectively. Those instruction's context definitions contain the `init` attribute for the above accounts. Indicating that the accounts must not exist prior to calling those instructions. In the instruction handler, however exists an additional check that validates if those are new accounts; however, this check will always be true because the Anchor context definition ensures that they did not exist, hence the below `require` statement is unnecessary:

```
require!(ctx.accounts.config.admin == Pubkey::default(), TokenTableError::AlreadyDeployed);
```

Impact: No impact on code functionality.

Recommendation(s): Remove those `require` statements.

Status: Fixed

Update from TokenTable: Removed `require!()` statements in ade803ab2628944df96a406546d5573692a13469.

7.12 [Best Practice] In some cases the `num_of_unlocks_for_each_linear` vector may waste rent

File(s): `create_preset.rs`

Description: When the `PresetAccount` enables the `stream` configuration, the `num_of_unlocks_for_each_linear` field becomes ineffective and can be set to an empty vector.

```
if preset.stream {
  num_of_unlocks_for_incomplete_linear = latest_incomplete_linear_duration;
} else {
  num_of_unlocks_for_incomplete_linear =
    preset.num_of_unlocks_for_each_linear[latest_incomplete_linear_index as usize];
}
```

However, during the `create_preset()` function, the `num_of_unlocks_for_each_linear` field must always be set to the same length as the `linear_start_timestamps_relative` vector, regardless of the situation.

```
if
  !(total == BIPS_PRECISION) &&
  preset.linear_bips.len() == preset.linear_start_timestamps_relative.len() &&
  preset.linear_start_timestamps_relative[preset.linear_start_timestamps_relative.len() - 1] <
    preset.linear_end_timestamp_relative &&
  preset.num_of_unlocks_for_each_linear.len() == preset.linear_start_timestamps_relative.len()
{
```

Impact: This will cause some rent waste for certain fields in the `PresetAccount` when the stream is enabled.

Recommendation(s): It is recommended that when the stream is enabled, the `num_of_unlocks_for_each_linear` field can be left empty. Additionally, the `_preset_is_empty()` function should be modified to remove the `num_of_unlocks_for_each_linear` field from it, based on the `stream` value.

Status: Fixed

Update from TokenTable: If `preset.stream` is set to `true`, we no longer enforce that `preset.num_of_unlocks_for_each_linear` has the same length as `preset.linear_start_timestamps_relative` in cd8e53d03b91c20af3153fb8feb6cbf1988d7bea.

7.13 [Best Practice] Lack of two step ownership transfers

File(s): `transfer_ownership.rs` `transfer_program_admin.rs`

Description: The program provides instructions for transferring ownership of the program admin and for the individual `unlocker` accounts (Projects). Those instructions allow transfer of the ownership to any arbitrary account. Currently, best practice dictates that there should be some form of control over who the ownership is transferred to. This, in a classical Solidity implementation, would involve two separate calls. On Solana, this canbe done in a simplified way - there would still be a single instruction however there should be a requirement added to the instructions' contexts that the new address should also be a Signer.

Impact: Accidental loss of control over the protocol or `unlocker` accounts.

Recommendation(s): Ensure Signer type of account for new owner accounts.

Status: Acknowledged

Update from TokenTable: Changed to two-step ownership transfers (using two transaction signers) in 4a176e0a. This logic was amended in 89889a1fef4fe1f879818eb8e2b8a6d80e8b76de, allowing the second signer to be `null` in calls to `transfer_ownership()` and `transfer_program_admin()`. In this case, the new owner/admin must call `receive_ownership()` or `receive_program_admin()`, respectively, to complete the permission transfer. After additional internal discussion, we have elected to roll back the two-step ownership transfer modifications for `transfer_ownership()` in 3dbd4b333432893f482acc7dc12b947c54ce324f. The added complexity does not justify the risks in typical usage (limited frontend verification is performed).

7.14 [Best Practice] Missing check for `fee_collector` in `Unlocker` account initialization

File(s): `initialize.rs`

Description: The `Unlocker` account stores the `fee_collector` field, which is initialized in the `initialization()` instruction and cannot be modified afterward. If an incorrect `fee_collector` is provided during initialization, it will be permanently set, preventing any future changes.

Impact: For this `Unlocker` is will be impossible to claim tokens through the `claim()` instruction.

Recommendation(s): It is recommended to check whether `fee_collector` is the expected account address during the `initialization()` instruction.

Status: Fixed

Update from TokenTable: Added the ability to change the `fee_collector` for a project rather than adding verification in a80d3c31d. We may need to change this address at some point in the future. This function is only callable by an admin (read: one of our wallet accounts), so errors should not happen in setting these values, and we would be able to fix any errors if need be.

7.15 [Best Practice] Missing check to verify if the fields stored in the `Account` are consistent with the seed

File(s): `create_preset.rs`

Description: The `PresetAccount` stores the `project_id` field, and the `ActualAccount` stores both the `project_id` and `actual_id` fields. The protocol does not check whether these stored fields match the fields used to generate the seed during account initialization.

Impact: No on-chain impact, but off-chain parsing may result in mismatched data in the account.

Recommendation(s): It is recommended to check whether the relevant fields in the Preset struct and Actual struct match.

Status: Fixed

Update from TokenTable: Added checks to ensure relevant fields match in Preset and Actual structs in 2079107e.

Update from CODESPECT: An unnecessary duplicated `project_id` argument is added to the instruction while the same value is present in the `actual` struct argument. The same goes for Preset creation.

8 Evaluation of Provided Documentation

The TokenTable team provided documentation in two forms:

- **Official Documentation Website:** The official documentation contains the protocol's design and implementation details, providing an overview of the protocol's purpose for both users and auditors. Unfortunately, the current state of the documentation website does not contain a version for Solana contracts.
- **Natspec Comments:** The code includes comments for key processes to help understand the logic. However, most functions lack comments, and expanding documentation coverage would enhance the overall comprehensibility of the code.

The documentation provided by TokenTable offered valuable insights into the protocol, significantly aiding CODESPECT's understanding. However, the public technical documentation could be further improved to better present the protocol's overall functionality and facilitate the understanding of each component.

Additionally, the TokenTable team was consistently available and responsive, promptly addressing all questions raised by CODESPECT during the evaluation process.

9 Test Suite Evaluation

9.1 Compilation Output

```
> anchor build
   Compiling fee-collector v0.1.0 (/tmp/009-TokenTable-Solana-UnlockerV2/programs/fee-collector)
   Compiling merkle-token-distributor-solana v0.1.0
    ↪  (/tmp/009-TokenTable-Solana-UnlockerV2/programs/merkle-token-distributor-solana)
    Finished release [optimized] target(s) in 8.87s
   Compiling fee-collector v0.1.0 (/tmp/009-TokenTable-Solana-UnlockerV2/programs/fee-collector)
   Compiling merkle-token-distributor-solana v0.1.0
    ↪  (/tmp/009-TokenTable-Solana-UnlockerV2/programs/merkle-token-distributor-solana)
    Finished `test` profile [unoptimized + debuginfo] target(s) in 3.37s
     Running unittests src/lib.rs
      ↪  (/tmp/009-TokenTable-Solana-UnlockerV2/target/debug/deps/merkle_token_distributor_solana-d2e3c791e510eca8)
   Compiling fungible-token-distributor-solana v0.1.0
    ↪  (/tmp/009-TokenTable-Solana-UnlockerV2/programs/fungible-token-distributor-solana)
    Finished release [optimized] target(s) in 4.58s
   Compiling fungible-token-distributor-solana v0.1.0
    ↪  (/tmp/009-TokenTable-Solana-UnlockerV2/programs/fungible-token-distributor-solana)
    Finished `test` profile [unoptimized + debuginfo] target(s) in 1.50s
     Running unittests src/lib.rs
      ↪  (/tmp/009-TokenTable-Solana-UnlockerV2/target/debug/deps/fungible_token_distributor_solana-b64a3478f355697a)
   Compiling fee-collector v0.1.0 (/tmp/009-TokenTable-Solana-UnlockerV2/programs/fee-collector)
   Compiling unlocker-v2-solana v0.1.0 (/tmp/009-TokenTable-Solana-UnlockerV2/programs/unlocker-v2-solana)
    Finished release [optimized] target(s) in 11.19s
   Compiling fee-collector v0.1.0 (/tmp/009-TokenTable-Solana-UnlockerV2/programs/fee-collector)
   Compiling unlocker-v2-solana v0.1.0 (/tmp/009-TokenTable-Solana-UnlockerV2/programs/unlocker-v2-solana)
    Finished `test` profile [unoptimized + debuginfo] target(s) in 2.85s
     Running unittests src/lib.rs
      ↪  (/tmp/009-TokenTable-Solana-UnlockerV2/target/debug/deps/unlocker_v2_solana-342b1908c8252196)
   Compiling fee-collector v0.1.0 (/tmp/009-TokenTable-Solana-UnlockerV2/programs/fee-collector)
    Finished release [optimized] target(s) in 3.56s
    Finished `test` profile [unoptimized + debuginfo] target(s) in 0.16s
     Running unittests src/lib.rs
      ↪  (/tmp/009-TokenTable-Solana-UnlockerV2/target/debug/deps/fee_collector-d60107b082febe11)
```

9.2 Tests Output

Fee Collector's test output:

```
yarn run v1.22.22
$ /.../009-TokenTable-Solana-UnlockerV2/node_modules/.bin/ts-mocha -p ./tsconfig.json -t 1000000
 ↪  tests/_fee_collector.ts


  fee-collector init
     Is initialized! (461ms)
     Set Default Fee (475ms)
     Collect Fee (lamports) (464ms)
     Collect Fee (SPL) (945ms)
     Withdraw (fail - not owner) (471ms)
     Withdraw (succeed) (460ms)


  6 passing (5s)

  Done in 6.73s.
```

Unlocker's test output:

```
yarn run v1.22.22
$ /Users/jecikpo/Projects/solana/009-TokenTable-Solana-UnlockerV2/node_modules/.bin/ts-mocha -p ./tsconfig.json -t
↪  1000000 tests/unlocker-v2-solana.ts

  0 passing (1ms)

 token-table-unlocker-v2-solana
   Unlocker
     Core
       should initialize correctly (44.068583ms)
       should create a preset and enforce permissions (38.640667ms)
       should create an actual and enforce permissions, no skipping (43.092792ms)
       should forbit creating new actual if create is disabled (33.286333ms)
       should manage transfering an actual (39.026542ms)
       should let founder withdraw deposit and enforce permissions (40.934959ms)
       should calculate the correct claimable amount
         no skip
           key timestamps (41.722ms)
           random timestamps (92.381875ms)
         no skip (134.317625ms)
           Random amount skipped: 5727
         random skip
           key timestamps (41.256667ms)
           Random amount skipped: 4505
           random timestamps (94.991959ms)
         random skip (136.408375ms)
       should calculate the correct claimable amount (271.191709ms)
       should let investor claim the correct amount (58.067084ms)
       should let founders or cancelables cancel and refund the correct amount (42.2015ms)
       should let investor claim the correct amount (two projects) (88.420959ms)
       should let delegate claim the correct amount (60.668375ms)
       Fee Collector (lamports) (39.186333ms)
       Fee Collector (SPL) (43.763292ms)
     Core (843.912667ms)
   Unlocker (843.953917ms)
 token-table-unlocker-v2-solana (844.066917ms)
 tests 16
 suites 6
 pass 16
 fail 0
 cancelled 0
 skipped 0
 todo 0
 duration_ms 866.629333
  Done in 1.37s.
```

9.3 Notes about Test suite

The TokenTable team delivered a rather comprehensive test suite, showcasing a well-structured approach to ensuring the protocol's correctness and resilience. Key suggestions of the test suite include:

- **Missing Functionality:** There are certain instructions whose validation is not currently included in the test suite, e.g. `disable_cancel`, `disable_withdraw`. CODESPECT recommends adding them to the test suite.
- **Edge Cases:** Beyond basic operations, the test suite should cover some basic edge cases specific to values which can be provided by the users of the protocol. One example would be adding a test case to verify the smallest possible time difference between consecutive claims.
- **Fee Collector:** The Fee Collector program tests should include tests for all fee options, i.e. bips, fixed fee, and default fee.

Overall, the test suite reflects a mature development process and significantly enhances the reliability of the protocol.

CODESPECT also recommends explicitly defining strict invariants that the protocol must uphold. Incorporating tests to validate these invariants would ensure that critical assumptions about the system's behaviour are consistently maintained across all functionalities, further bolstering the protocol's security and stability.