IMPORTANT INFORMATION FROM WISDOMTREE DIGITAL

WisdomTree Digital, which operates through one or more subsidiaries of WisdomTree Inc., is providing access to the security audit reports conducted by OpenZeppelin which follow this page. By accessing the accompanying reports, you hereby acknowledge and agree that you have read and understood the contents herein.

Statements in the accompanying reports regarding the design, implementation, integration, suitability or functionality of the token standards framework or one or more smart contracts, such as those related to compliance, fraud prevention, risk management, security, audits, regulatory adherence and operations, address activities, events or developments that generally will or may occur in the future, are based on certain assumptions of the audit firm, are not tailored to any particular entity and no assurances can be made regarding the foregoing, notwithstanding statements in the accompanying reports such as those related to ensuring, enabling, effectiveness, robustness or suitability of certain functionality, operations or otherwise. Actual implementation, integration, suitability or functionality are all highly dependent on potential technical implementation and an issuer's, intermediary's or other applicable market participants' specific considerations, which may include those related to domicile, current or future investor or user base, regulatory requirements and other regulatory considerations, operating, compliance or risk management processes or procedures, desired functionality, technological know-how or comfort, general expectations or predictions, all of which are subject to a number of uncertainties, may change, may not come to fruition or may not work as intended.

In addition, blockchain and related technology, including smart contracts, are a relatively new and untested technology. The findings in the accompanying reports should not be considered a comprehensive list of security or other issues, flaws or defects in the framework, system or codebase. Token frameworks and smart contracts are subject to vulnerabilities and coding errors and no assurance can be made regarding the token standards framework or smart contract suitability, functionality, quality or reliability, even after audit recommendations have been implemented. WisdomTree Digital expressly disclaims all liability arising from the use of this information or reliance on the information provided and to the fullest extent permitted by applicable law, all warranties, whether express or implied, are hereby disclaimed by WisdomTree Digital. Use of this information is at the sole risk of the user. WisdomTree Digital does not provide financial, legal, regulatory or other professional advice.

This disclosure statement and the accompanying reports cannot and do not disclose all the risks and other factors necessary to evaluate a potential product or service from WisdomTree Digital. Before you decide to purchase or use a product or service from WisdomTree Digital or its affiliates, carefully read the specific information regarding such product or service (e.g., prospectus or other disclosure document, as applicable), including applicable risk disclosures.

OpenZeppelin | security

WisdomTree Solana Tokenized Funds Audit

January 14, 2026

Table of Contents

Summary

Type	RWA Tokenization
Timeline	From 2025-11-03 To 2025-12-19
Languages	Rust

Total Issues	56 (49 resolved, 2 partially resolved)
Critical Severity Issues	4 (4 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	12 (12 resolved)
Low Severity Issues	17 (14 resolved, 2 partially resolved)
Notes & Additional Information	21 (17 resolved)
Client Reported Issues	2 (2 resolved)

Scope

OpenZeppelin performed an audit of the wisdomtreeam/wisdomtree_tokenization repository. The audit included a review of three versions of the codebase: an initial version at commit 00a0e14, a second version at commit 5e096cb, and a third version at commit 0c20dce. The issues identified in the initial phase are enumerated in the Phase I Issues section of the report, while all other issues outside this section are from the second and third phases.

For the first phase, the following files were in scope:

```
programs/
├── admin-operations/
│   └── src/
│       ├── errors.rs
│       ├── instruction.rs
│       ├── lib.rs
│       ├── pda.rs
│       ├── state.rs
│       └── processor/
│           ├── batch_clawback.rs
│           ├── batch_freeze.rs
│           ├── batch_mint_fund_tokens.rs
│           ├── batch_thaw.rs
│           ├── burn_fund_tokens.rs
│           ├── clawback.rs
│           ├── freeze.rs
│           ├── mint_fund_tokens.rs
│           ├── mod.rs
│           ├── pause.rs
│           ├── resume.rs
│           └── thaw.rs
├── compliance-config/
│   └── src
│       ├── errors.rs
│       ├── events.rs
│       ├── lib.rs
│       ├── state.rs
│       ├── handlers/
│       │   ├── initialize_config.rs
│       │   ├── mod.rs
│       │   ├── register_context.rs
│       │   ├── remove_context.rs
│       │   ├── set_enforce_sbt_admin_ops.rs
│       │   └── update_permanent_delegate.rs
│       └── instructions/
│           ├── initialize_config.rs
│           ├── mod.rs
```

```
│           ├── register_context.rs
│           ├── remove_context.rs
│           ├── set_enforce_sbt_admin_ops.rs
│           └── update_permanent_delegate.rs
├── compliance-roles/
│   └── src/
│       ├── errors.rs
│       ├── events.rs
│       ├── lib.rs
│       ├── state.rs
│       ├── handlers/
│       │   ├── bootstrap_admin.rs
│       │   ├── grant_role.rs
│       │   ├── initialize_children.rs
│       │   ├── mod.rs
│       │   └── revoke_role.rs
│       └── instructions/
│           ├── bootstrap_admin.rs
│           ├── grant_role.rs
│           ├── initialize_children.rs
│           ├── mod.rs
│           └── revoke_role.rs
├── sbt-registry/
│   └── src/
│       ├── errors.rs
│       └── lib.rs
└── transfer-hook/
    └── src/
        ├── errors.rs
        ├── lib.rs
        ├── processor.rs
        └── state.rs
```

For the second and third phases, the following files were in scope:

```
programs/
├── fund-ops/
│   └── src/
│       ├── constants.rs
│       ├── errors.rs
│       ├── lib.rs
│       ├── pda.rs
│       ├── state.rs
│       └── handlers/
│           ├── add_sbt_context.rs
│           ├── assert_invariants.rs
│           ├── batch_freeze.rs
│           ├── batch_mint_fund_tokens.rs
│           ├── batch_thaw.rs
│           ├── freeze.rs
│           ├── initialize_compliance_config.rs
│           ├── mint_fund_tokens.rs
│           ├── mod.rs
│           ├── pause.rs
│           ├── remove_sbt_context.rs
```

```
│           ├── resume.rs
│           ├── rotate_authority.rs
│           ├── thaw.rs
│           └── update_compliance_config.rs
├── roles-registry/
│   └── src/
│       ├── constants.rs
│       ├── errors.rs
│       ├── events.rs
│       ├── lib.rs
│       ├── pda.rs
│       ├── state.rs
│       ├── handlers/
│       │   ├── bootstrap_admin.rs
│       │   ├── grant_role.rs
│       │   ├── initialize_children.rs
│       │   ├── mod.rs
│       │   └── revoke_role.rs
│       └── instructions/
│           ├── bootstrap_admin.rs
│           ├── grant_role.rs
│           ├── initialize_children.rs
│           ├── mod.rs
│           └── revoke_role.rs
├── sbt-ops/
│   └── src/
│       ├── errors.rs
│       ├── lib.rs
│       └── pda.rs
└── transfer-hook/
    └── src/
        ├── constants.rs
        ├── errors.rs
        ├── lib.rs
        └── state.rs
```

System Overview

The primary goal of the protocol under review is to support a regulated fund token the lifecycle of which can be governed through on-chain KYC credential verification and tightly scoped administrative actions, such as minting, freezing, and pausing. Each transfer of the fund token involves comprehensive checks ensuring that both sender and recipient satisfy the current compliance requirements, which are represented on-chain through approved soulbound tokens (SBTs). Each fund token is governed by a set of administrators, who retain reversible controls for regulatory or operational interventions.

Programs

In order to reduce the attack surface and to simplify the review, the codebase has been split to 4 separate programs, each having different responsibilities. These programs are briefly described below:

- The `roles-registry` program uses a mint-scoped hierarchical role system, where privileged roles are separately configured for each mint token account, both for SBT tokens and fund tokens. The available roles include:
  - the `DefaultAdmin` role, which governs the overall configuration and can grant and revoke any role
  - the `DelegatedAdmin` role, responsible for distributing operational roles within bounded limits
  - the `Registrar` role, responsible for multiple token operations, such as minting, pausing or freezing individual accounts
  - the `Issuer` role (currently unused)
- The `fund-ops` program is responsible for handling the privileged operations on fund tokens in the system. In particular, it enables minting, freezing, thawing, pausing, and resuming operations. Furthermore, for accounts with the `DefaultAdmin` role, it enables approving SBT tokens, which are used for on-chain compliance verification whenever a fund token is being transferred.
- The `sbt-ops` program enables minting SBT tokens to specified accounts. Such mints can be initiated by accounts with the `Registrar` role. Each account may only have at most 1 such token and this token is non-transferable.

- The `transfer-hook` program is called whenever a fund token transfer takes place. For each such transfer, it verifies that both the source and destination accounts meet the compliance rules, that is, that they both have the relevant SBT tokens. If this is not the case, the transfers are rejected.

Security Model and Trust Assumptions

This section outlines the observations regarding the privilege roles and trust assumptions of the codebase.

Privileged Roles

Throughout the codebase, multiple privileged roles were identified:

- **Default Administrator**: It can grant or revoke any other role in the system for a given mint account. There is a limit of 3 Default Admins for each mint account.
- **Delegated Administrator**: It can only grant the Registrar and Issuer roles and cannot grant more than 20 roles. Once this role is revoked, all roles granted by this role are revoked as well.
- **Registrar**: It can execute operational instructions (minting, freezing/thawing, pausing/resuming, SBT issuance). Accounts with this role cannot grant nor revoke any roles.
- **Permanent Delegate (external multisig account)**: The only entity capable of burning fund and SBT tokens, and transferring fund tokens from any account.
- **Upgrade Authority**: This is an account that can upgrade a program at any point in time. Each program has an upgrade authority.

Trust Assumptions

During the audit, the following trust assumptions were identified:

- All the privileged entities in the system will act in the best interest of the protocol and its users. In particular, that SBT tokens will be minted to correct accounts and only when the KYC procedure has been passed

- Accounts will be frozen and tokens will be burnt from the accounts only when it is justified.
- The logic of the programs will only be updated when necessary and that the new code will be sufficiently tested and audited before migration.
- Each privileged entity will keep their private keys secure.
- All programs and tokens in use will be configured correctly.
- All programs will be deployed to correct addresses, matching their IDs referenced in the code
- Each token will have correct privileged entities set and correct extensions enabled.
- The in-scope programs will be configured correctly, which includes whitelisting relevant SBT tokens and configuring correct roles for each token.
- The Permanent Delegate will be configured correctly for each token and that its address will correspond to a key-pair account, not a PDA.
- All code will be sufficiently tested before being deployed. All possible code paths will be fully covered by unit tests along with extensive integration tests, ensuring the correct interaction of each program, both with in-scope and out-of-scope components.

Phase I Issues

This section contains issues identified in the phase I of the engagement.

All phase I issues were resolved before phase II.

`GrantRole` Instruction May Be Used to Grant `DefaultAdmin` Role to Anyone

The `GrantRole` instruction of the `compliance-roles` program enables privileged entities to grant roles to a specified account. In order to ensure that the granter has sufficient privileges, it requires their signature and validates that their current role allows for granting the desired role.

However, this validation does not take into account the current granter's role, but the role specified as an argument, which can be arbitrary. As a consequence, anyone can sign a transaction containing the `GrantRole` instruction and pass `granter_role_u8` indicating the `DefaultAdmin` role and successfully grant this role to an arbitrary account.

Cross-Mint Role Granting and Revocation

The `GrantRole` and `RevokeRole` instructions of the `compliance-roles` program are designed to allow authorized wallets to manage roles for a particular token mint. A critical security assumption is that a user's permissions are confined to the specific mint for which they hold a role. The instruction handlers validate that the user is the owner of the provided role account and that this role has permission to grant or revoke the target role.

However, none of the handlers verify that the mint associated with the user's role account is the same as the mint of the role being acted upon. As a result, it is possible to escalate privileges from one mint across all mints managed by the program.

Duplicate Code

Instruction handlers of the `compliance-config` program are implemented in the `handlers` directory. However, the same handlers are also present in the `instructions` directory, where only `#[derive(Accounts)]` structs should be present.

For example, the same `register_context` handler is present both in the `handlers/register_context.rs` file and in the `instructions/register_context.rs` file. As a result, the `instructions` directory contains redundant code that is never used.

The following sections contain issues identified in phases II and III of the engagement.

Critical Severity

C-01 Incorrect Argument Order in `RevokeRole`

The `RevokeRole` instruction of the `roles-registry` program enables privileged accounts to revoke roles of other accounts. However, the `#[instruction(...)]` macro for this instruction lists arguments in an order that is different from the order in the instruction handler, which is used to generate function definitions in IDL.

As a result, Anchor will attempt to deserialize provided arguments to incorrect types without reporting a compilation error. In particular, the provided `revoker_role_u8` value will be interpreted as `cascade` boolean and vice versa, while the provided `target_role_u8` will be interpreted as the first byte of the provided `target_wallet`, making the `RevokeRole` instruction unusable.

Consider changing the order of arguments in the `#[instruction(...)]` macro so that it matches the order of arguments specified in the instruction handler.

Update: *Resolved at commit 8c834a5.*

C-02 Incorrect Instruction Discriminator

The `transfer-hook` program implements logic for verifying source and destination accounts for each token transfer. This logic is contained in the `Execute` instruction. However, this instruction's discriminator will be automatically generated by Anchor as the first 8 bytes of `sha256("global:execute")`, being equal to `[130, 221, 242, 154, 13, 193, 189, 29]`, while the discriminator required by the transfer hook interface is equal to the first 8 bytes of `sha256("spl-transfer-hook-interface:execute")`, being equal to `[105, 37, 101, 197, 75, 251, 102, 26]`. As a result, all calls from the `token-2022` program to the transfer hook will fail.

Consider manually specifying the correct discriminator for the `Execute` instruction through the custom discriminators Anchor feature.

Update: *Resolved at commit 793f8bc.*

C-03 Insufficient Ownership Check

The `transfer-hook` program is designed to enforce compliance rules at the moment of a token transfer. Its `execute` function is called by the Token-2022 program to ensure that both the sender and receiver of the tokens hold a specific Soulbound Token (SBT), proving they are compliant. This check is performed by inspecting SBT token accounts provided in the transaction's `remaining_accounts`.

The `execute` function and its helper, `check_sbt_balance`, correctly check that the provided SBT token accounts have a balance greater than zero and belong to the correct SBT mint. However, the logic does not verify that the source soul bound token account belongs to the source account's owner, and the destination soul bound token account belongs to the destination account's owner.

This omission allows an attacker to bypass the compliance check by providing a valid SBT account belonging to any third party. For example, a non-compliant user can receive tokens by passing the SBT account of a compliant user for the validation, completely defeating the purpose of the transfer hook. A similar issue is present in the `fund-ops` program, where the owner of SBT account and the owner of the fund token account are not ensured to be the same entity during token minting.

Consider verifying that the owner of a specified SBT account is the same as the owner of a given fund token account in the case of transfers and mints.

Update: *Resolved at commits 94512a2, 34516f4, and d1fcbe6. Specifically, for the transfer hook, SBT account ownership is now verified implicitly by deriving this account from a relevant address. The batch minting functionality was removed in commit fedb1c4.*

C-04 Missing Token Program Ownership Check Allows Spoofing of SBT Accounts

The `transfer-hook` program's `execute` function validates compliance by checking a user's Soulbound Token (SBT) holdings. It does this by examining SBT token accounts passed via `remaining_accounts` and calling the internal `check_sbt_balance` function to verify their state. The security of the entire hook relies on the authenticity of these accounts as genuine SPL token accounts.

However, the `check_sbt_balance` function does not verify that the provided SBT accounts are owned by the SPL Token-2022 program. The function deserializes the account data and checks its `mint` and `amount`, but it never checks the `owner` field of the `AccountInfo`

struct itself. This allows an attacker to pass a malicious account that is not a real token account but contains crafted data mimicking the structure of one. The deserialization can succeed on this fake data, causing the function to return `true` and incorrectly approve the transfer.

This vulnerability, which occurs in the main validation loop, allows an attacker to completely bypass the SBT compliance check using a spoofed account. The same issue exists in the `fund-ops` program during fund token minting, where the SBT account ownership is not checked against the Token-2022 program.

Consider ensuring that all the SBT token accounts passed for transfers and minting are owned by the Token-2022 program to prevent spoofing by malicious users.

Update: *Resolved at commits 34516f4 and d1fcbe6. For the transfer hook, the ownership check is now implicitly performed by requiring the provided SBT accounts to be PDAs derived from the Token-2022 program.*

Medium Severity

M-01 Insufficient `parent_children` Validation in `RevokeRole`

The `RevokeRole` instruction of the `roles-registry` program allows admins to revoke the role of a certain account. The process works in several steps. First, after validating that revoker has privileges to perform the operation, it is validated that all of the target's children role accounts have been supplied. Afterwards, the revocation target is removed from the children list of its parent.

However, this list is not sufficiently validated: it is only validated that it belongs to the `roles-registry` program, that the `parent_children` account contains data of the correct type, and that the children list contains the revocation target. In particular, it is not validated that the `parent_children` account corresponds to a valid `mint` account and to the `RolePDA` account which was used to grant the role to the `target_wallet`. As such, it allows situations when the role is revoked from an account, but an incorrect `parent_children` list (either for a wrong `mint` or potentially for a different parent for the same mint, in case when two different accounts grant roles to the same target) is updated.

Consider validating the `parent_children` account more thoroughly to ensure that it has been derived from the correct seeds. This could be done by using the `Option` wrapper for accounts and applying Anchor constraints to such account in the `#[derive(accounts)]` struct. Alternatively, the `seeds` Anchor constraint could be used on `UncheckedAccount` and the account may be specified as mandatory for the instruction and non-emptiness of data would only be required in case when the account is determined to be necessary in the handler.

Update: *Resolved at commit e5920c6.*

M-02 Children List of `DelegatedAdmin` Is Not Always Updated

The `GrantRole` instruction of the `roles-registry` program updates the children list of role granter with the account being granted a role whenever the granter has the `DelegatedAdmin` role. However, the code explicitly allows for circumventing this mechanism by not providing the account storing the list of granter's children.

This results in a situation where the children list is not properly updated and, whenever the impacted `DelegatedAdmin` gets their role revoked, their children will not have their roles revoked and would need to be handled manually. A similar problem exists in the `RevokeRole` instruction, where the children of the target account can be provided optionally.

Consider always requiring the `granter_children` account to be present in the `GrantRole` instruction whenever the `granter` has the `DelegatedAdmin` role. Furthermore, consider adding a similar requirement for the `target_children` account in the `RevokeRole` instruction. Skipping `target_children` account could still be allowed in emergency situations, for example when a `DelegatedAdmin` repeatedly grants and revokes roles for new accounts in order to avoid being revoked.

Such a possibility, however, should be only reserved for accounts with `DefaultAdmin` role, treated as an emergency situation, used only when absolutely necessary, and followed by the manual revocation of all children and the closing of the `target_children` account. This should be clearly reflected in the documentation.

Update: *Resolved at commits 1acba38 and 1a1b799. `DelegatedAdmin` was removed in commit 66a8ac2.*

M-03 Insufficient Child Account Validation

The handler of the `RevokeRole` instruction requires the `remaining_accounts` list to contain all the children of the account from which the `DelegatedAdmin` role is being revoked. The validation includes ensuring that the currently processed account is owned by the `roles-registry` program, that its role has been granted by the `target_wallet`, that its `wallet` field corresponds to the given child, and that its `role` is `Registrar` or `Issuer`.

However, this validation does not ensure that the account corresponds to a valid `mint` account. Hence, child accounts related to a different `mint` can be provided to be closed. As a result, it is possible that not all child accounts for the correct `mint` will be closed and that some other child accounts corresponding to different `mint` will be closed unexpectedly.

Consider verifying that child accounts correspond to the correct `mint` account.

Update: *Resolved at commit fc1b111.*

M-04 Insufficient Associated Token Account Validation

The `sbt-ops` program is designed to issue Soulbound Tokens (SBTs), which are intended to be unique, non-transferable credentials for a specific wallet. The program includes a duplicate prevention mechanism that checks if a recipient's token account already holds a balance before minting a new token. This logic relies on checking a single, predictable token account for each user to enforce uniqueness. However, the `mint_sbt` and `batch_mint_sbt` instructions do not validate that the provided `recipient_sbt_account` is the canonical Associated Token Account (ATA) for the given `recipient` and `sbt_mint`.

As a consequence, a user can generate multiple standard token accounts owned by them for the same SBT mint and by passing each of these different token accounts to the `mint_sbt` function, they can bypass the duplicate prevention check and acquire multiple SBTs. This undermines the "soulbound" nature of the token, and allows the wallet to still retain privilege even if one of its SBT token is burned. The issue stems from the account validation in the `MintSbt` context, which accepts any mutable `AccountInfo` without deriving and checking against the ATA address.

Furthermore, the authority of the provided ATA account is not verified against the recipient account. This omission allows a caller to provide a valid token account that belongs to an arbitrary user, pair it with a different recipient public key, and successfully mint the token. This leads to a state inconsistency where the SBT is minted to an incorrect wallet, which can cause

misleading on-chain events and failed downstream operations that rely on the recipient having the SBT. The issue exists in both the single mint and the batch mint logic.

A similar issue exists for the fund token, where during the minting operation, it is not verified that the token account receiving tokens is a canonical ATA derived from the token recipient. Similarly, in a transfer hook, the `source` and `destination` accounts are not ensured to be canonical ATAs corresponding to sender and recipient. As a consequence, it is possible to mint or transfer fund tokens to an account which is not an ATA, but an auxiliary account.

As such, it is possible to transfer fund tokens from or to an account which does not have a relevant SBT token by changing the authority of relevant auxiliary account to the intended recipient's `Pubkey`. This could be done by splitting fund tokens into multiple auxiliary accounts and transfers of any amount can be performed by gifting a relevant auxiliary account to the recipient, even if the recipient does not have SBT tokens. The recipient can then transfer all these tokens to arbitrary account by changing the authority of the auxiliary account once again.

Consider enforcing that the `recipient_sbt_account` is the canonical ATA derived from the `recipient` and `sbt_mint` accounts. This will ensure that each wallet can have at maximum one SBT and that tokens are minted to the correct accounts. Furthermore, consider ensuring that tokens are always minted to a canonical ATA address derived from the desired recipient and `mint` account. Similar validation needs to be performed in the transfer hook both on the `source` and `destination` accounts.

Update: *Resolved at commits fc1b111, 0783c79, and 7f8bd0f.*

M-05 Child Roles Remain Active After Delegated Admin Revocation

The `RevokeRole` instruction of the `roles-registry` program is designed to revoke a given role from a specified entity. In case when the `DelegatedAdmin` role is being revoked, all child roles that have been granted by it are revoked as well. This action is critical for ensuring a complete and atomic removal of permissions from the system.

However, the instruction does not perform the revocation of child roles correctly. While the `DelegatedAdmin`'s own role account is closed, the handler logic does not close the child `RolePDA` accounts. Since these child `RolePDA` accounts are not closed on-chain by the program, they remain active with their full privileges. This allows users to retain their granted roles even after the revoking authority has been removed, undermining the security of the role management system.

Consider modifying the `revoke_role` handler to explicitly close all child `RolePDA` accounts. The program should iterate through the `remaining_accounts` that correspond to the children and programmatically close each one. The `target_children` account should be closed as well if it corresponds to a delegated admin.

Update: *Resolved at commits ca61ff7 and 31b7a62.*

M-06 Transfers and Mints May Fail Even if Valid SBT Tokens Are Provided

The `Execute` instruction of the `transfer-hook` program verifies that the source and destination accounts have a non-zero balance of any of the allowed SBT tokens. The allowed SBT tokens are specified through the `primary_sbt_mint` and `additional_contexts` fields of the `ComplianceConfigPDA` account. In order to pass a verification, it is necessary to specify SBT accounts corresponding to all source and destination accounts and related to all contexts. The transfer will only succeed if, for any allowed SBT tokens, both accounts have a non-zero balance.

However, the `check_sbt_balance` function, which is used for balance verification, returns an error in case the data for any of the provided SBT accounts does not unpack correctly. It causes the entire transaction to revert before all SBT accounts are validated. As a consequence, if an `n` th context from the list is used for transfer verification, both source and destination accounts need to have existing accounts associated with the primary SBT mint and `n-1` previous contexts, even if they have never used these tokens.

Furthermore, in case the contexts change, both source and destination accounts need to keep track of these changes and create associated accounts for all added contexts in order to be able to send or receive tokens. Moreover, the `Execute` instruction always requires accounts related to all registered contexts, while only one valid pair of SBT accounts is actually needed for validation. Similar problems are present in the `mint_fund_tokens` instruction handler of the `fund-ops` program where all contexts need to be provided, and the instruction will fail if any of the accounts being iterated over have empty data.

Consider requiring that only one pair of SBT accounts is provided for the transfer hook and for the `MintFundTokens` instruction of the `fund-ops` program and validating that they correspond to any of the valid contexts before validating the balance. Furthermore, consider explicitly handling a situation where an account with empty data has been passed by reverting the transaction with an informative error instead of trying to deserialize empty data.

Update: *Resolved at commits d888574 and 1207622.*

M-07 Front-Running Attacks Possible on `fund-ops` and `roles-registry`

The `InitializeComplianceConfig` instruction of the `fund-ops` program and the `BootstrapAdmin` instruction of the `roles-registry` program are permissionless and enable the caller to create accounts storing config for `roles_registry` and `transfer_hook` programs.

In the case of the `BootstrapAdmin` instruction, the caller creates a `MintConfigPDA` account and grants the initial `DefaultAdmin` role, which enables gaining privileges related to fund token, and in case of the `InitializeComplianceConfig` instruction, caller can create the `ComplianceConfigPDA` account and configure contexts related to SBT tokens, which are checked against during transfer and mint operations.

Both `ComplianceConfigPDA` and `MintConfigPDA` accounts' addresses are derived only based on the constant string ("compliance_config" and "mint_config", respectively) and the provided `mint` account's `Pubkey`, which means that for each `mint` account, there will be only one account of each type. This design enables front-running attacks, where an attacker calls the `InitializeComplianceConfig` or the `BootstrapAdmin` instruction and blocks subsequent calls of these instructions for any given `mint`, thus denying the possibility of using the token it represents in the system.

Consider always calling the `InitializeComplianceConfig` and the `BootstrapAdmin` instructions atomically, in one transaction, in order to avoid front-running attacks. This could also be batched with an instruction creating the `mint` account. Alternatively, the `mint` account can be created later on a predetermined address. These requirements should be included in the documentation of both instructions.

Update: *Resolved at commits 9fe07d5 and 7822b1f. Now, the signature of the mint or freeze authority is required for any given mint in the `BootstrapAdmin` instruction.*

M-08 Incorrect Program IDs

For a Solana program to be deployed and interacted with on mainnet, its on-chain address must be consistently defined. The `declare_id!` macro in the source code sets the program's canonical address, while the `[programs.mainnet]` section in `Anchor.toml` is used by the toolchain for deployment and client generation. These two definitions must be identical for a successful mainnet deployment.

However, for all four programs (`roles_registry`, `fund_ops`, `sbt_ops`, and `transfer_hook`), the program ID defined in the `[programs.mainnet]` section of `Anchor.toml` does not match the canonical ID set in the `declare_id!` macro. This mismatch will prevent successful deployment to mainnet. Furthermore, the `fund-ops` program hardcodes incorrect addresses of `roles_registry` and `transfer-hook` programs, which do not match their IDs declared in relevant source files. The same is true for the `transfer-hook` program.

Consider synchronizing the `mainnet` program IDs in `Anchor.toml` to match the addresses defined in the `declare_id!` macro for each program. Furthermore, consider using program IDs imported from relevant programs instead of hardcoding them in the code.

Update: *Resolved at commit 0656a66. Instead of being hardcoded, the program IDs have been updated to match the Anchor config for `localnet`. It is trusted that before the deployment, these IDs will be set to values corresponding to the `mainnet` config.*

M-09 SBT Compliance Checks Are Bypassed in Minting Functions

The `fund-ops` program provides handlers for minting new fund tokens both individually, via `mint_fund_tokens`, and in groups, via `batch_mint_fund_tokens`. These administrative functions are intended to be controlled by a compliance framework configured in the `transfer-hook` program. This framework uses a `ComplianceConfigPDA` account containing an `enforce_sbt_on_admin_ops` flag to mandate Soul-Bound Token (SBT) ownership for recipients, even during privileged operations like minting.

Two distinct issues result in the bypass of this compliance mechanism. First, in the `mint_fund_tokens` handler, the `ComplianceConfigPDA` is treated as an optional account. A privileged user can therefore bypass the SBT check by simply omitting this account from the transaction, causing the validation logic to be skipped. Second, the `batch_mint_fund_tokens` handler does not perform any compliance verification. It loops through recipients and mint tokens without checking their SBT status, thus entirely omitting the `enforce_sbt_on_admin_ops` check.

Consider redesigning both minting handlers to enforce compliance checks consistently. The `mint_fund_tokens` handler should be modified to require the `ComplianceConfigPDA` as a mandatory account, ensuring that the `enforce_sbt_on_admin_ops` flag is always evaluated. Furthermore, the `batch_mint_fund_tokens` handler should be updated to incorporate the full compliance validation logic.

This includes accepting `ComplianceConfigPDA` and the recipients' SBT accounts, and then verifying the SBT status of each recipient in the batch according to the rules defined in the configuration. As the required changes to the `batch_mint_fund_tokens` handler will introduce significant complexity to the code, consider removing batch minting functionality entirely and batching multiple single mints in transactions instead.

Update: *Resolved at commit c77c06b.*

M-10 Incorrect Account Context Used for Batch Minting

The `fund-ops` program defines a `BatchMintFundTokens` instruction intended for minting tokens to multiple recipients. However, the handler for this instruction is incorrectly implemented to use the `MintFundTokens` account context, which is designed for single mints. This leaves the `BatchMintFundTokens` struct unused and creates a mismatch between the instruction's name and its underlying account structure, reducing code clarity.

Furthermore, since the `MintFundTokens` struct specifies an additional `recipient_token_account` account, which is unused during batch minting, users would be forced to always pass this dummy account for each batch minting, which is confusing and increases the Compute Unit cost.

Consider refactoring the `batch_mint_fund_tokens` instruction to use the `BatchMintFundTokens` account context as intended.

Update: *Resolved at commit 861181c.*

M-11 Transfer Restrictions May Be Bypassed Through Delegates

The `transfer-hook`'s `Execute` instruction ensures that both `source` and `destination` accounts' owners both have an SBT token from at least one of the allowed contexts when fund tokens are attempted to be transferred. This prevents sending and receiving fund tokens without having been granted the right to do so during the KYC procedure.

However, the actual SBT balance verification assumes that tokens are always sent by the `source` account's owner, yet it is possible to register a delegate for an account, who can initiate a transfer on behalf of the owner. This creates a possibility to implicitly transfer tokens to any account, not necessarily having any SBT tokens, by granting them a `delegate` role for

their account and approving any amount for them to use. Such a delegate can then transfer tokens as if they owned them, despite not having relevant SBT tokens.

Consider verifying that the `authority` account passed to the hook indeed corresponds to the `source` account's owner. Alternatively, if the behavior described in this issue is intended, consider clearly stating it in the documentation.

Update: *Resolved at commits 2f0e04e, f431756, and c77c06b.*

M-12 `RotateAuthority` Instruction Causes `fund-ops` Program to Malfunction

The `RotateAuthority` instruction of the `fund-ops` program allows changing the authority for the minting and freezing operations on fund tokens. However, these authorities are never meant to be changed, as `fund-ops` instructions always derive them as PDAs, from constant seeds.

As a result, any change of authorities will effectively block mint, freeze and thaw operations along with their batch counterparts. Furthermore, there is no need to ever change authorities, as PDAs being used as authorities cannot be compromised, as they lie outside the Ed25519 curve.

Consider removing the `RotateAuthority` instruction from the `fund-ops` program. Alternatively, if revoking privileges from the `fund-ops` is an intended consequence of the `RotateAuthority` instruction, consider clearly documenting it and renaming the instruction accordingly so that the new name clearly suggests that it drops privileges from the `fund-ops` program.

Update: *Resolved at commit 8af2733. The documentation has been updated to highlight the risks related to the `RotateAuthority` instruction. In addition, a boolean variable has been added to the instruction's handler as an additional safety mechanism. The WisdomTree team stated:*

> The `RotateAuthority` instruction in fund-ops is intended strictly as an emergency/migration mechanism, not for routine use. Only `Default_Admin` can call it, and it wraps Token-2022's `set_authority` to allow us to migrate Mint/Freeze/Pause/TransferHook authorities to new PDAs (e.g., when moving to a new program) or to regain control in a compromise scenario. Permanent Delegate rotation is explicitly disallowed. PD remains an EOA (Fireblocks multisig) for clawback. We understand that

> invoking rotate_authority is a breaking change for normal operations and treat it as a break-glass control governed by formal procedures rather than an everyday tool.

Low Severity

L-01 Delegation Count Not Maintained

The `delegation_count` field of the `RolePDA` account is intended to track and manage granted child roles. However, although there are functions present in the codebase for updating this field, they are not used and as a result the counter is not incremented when a `DELEGATED_ADMIN` grants a new role, nor it is decremented when a role granted by a `DELEGATED_ADMIN` is revoked, leading to persistent inaccuracies in the count.

Consider modifying both the `grant_role` and `revoke_role` handlers to correctly increment and decrement the `delegation_count` of the granter's `RolePDA` to ensure data consistency and accurate tracking of delegated roles.

Update: *Resolved at commit 66a8ac2. The `roles-registry` program has been simplified by removing the logic related to the `DelegatedAdmin` role.*

L-02 Manual Account Size Calculation

Throughout the codebase, sizes of accounts are calculated manually by adding the size of each member. For example, this is the case for `RolePDA`. However, this reimplements the functionality already provided by the `InitSpace` macro in Anchor and increases potential error surface as size calculations would need to be manually adjusted on each change of accounts' structure.

Consider using the `InitSpace` macro in order to calculate the required size automatically, thereby reducing the error surface and making the code more concise. Each such size would need to be increased by 8 bytes needed for account's discriminator.

Update: *Resolved at commits a57652e and 9c6579d.*

L-03 Inefficient Logging

Programs in the codebase extensively use the `msg!` macro, which internally invokes the `sol_log_` syscall. For example, this code from the `fund-ops` program invokes this syscall 6 times, just to log the accounts which have been provided for the instruction. However, each such syscall consumes at least `syscall_base_cost` Compute Units each time it is invoked.

The current value of `syscall_base_cost` is 100 and this cost is charged even for short messages. As a result, programs consume more Compute Units that it is necessary. Furthermore, such excessive logging, while useful during debugging, makes the logs harder to parse in a programmatic way and provides no additional value for the end users, who still need to cover the cost for emitting them.

Consider reducing the number of `msg!` macro invocations throughout the codebase, only leaving the most necessary ones. In many cases, such logs could be completely removed or replaced with events emission to still provide relevant information about the operations completed by the programs. In case when `msg!` invocations are needed, consider merging multiple consecutive `msg!` invocations into a single, bigger one, in order to reduce Compute Units cost.

Update: *Partially resolved at commits 916f2aa and dc53821. However, there are still instances of `msg!` invocation in the `sbt-ops` program. The WisdomTree team stated:*

> *We have reduced `msg!` usage across the codebase by removing account-dump logging and consolidating/removing verbose debug logs. However, there is still a limited number of msg! invocations in sbt-ops (as noted in the report). These remaining logs are intentionally scoped to operational/exception paths where traceability is valuable. They do not affect authorization or state safety; the impact is limited to marginal compute overhead on those transactions.*

L-04 Valid Roles Could Be Indexed from 1

The `Role` enum of the `registry-roles` program is indexed from 0, meaning that its first, `DefaultAdmin` value is represented as 0. However, when accounts on Solana are created, their data is, by default, initialized with 0s. As a result, the 0 value is usually used to mark "`Uninitialized`" enum states in Solana programs. Even though the `role` field is always initialized during account creation, using 0 value to represent any valid role is not recommended as it could lead to bugs and confusion in the future.

Consider representing valid roles as consecutive integers starting from 1 in order to avoid situation when a role is represented by the default 0 value. In order to do that, an additional, `Invalid` enum value with 0 value could be introduced to the `Role` enum and all other members should be shifted by 1.

Update: *Resolved at commits 8658959 and 3226622.*

L-05 `revoker_role` Is Unnecessarily Passed as Mutable

The `revoker_role` account is specified as a mutable to the `RevokeRole` instruction. However, the `RevokeRole` instruction does not modify this account in any way. As such, this constraint is not necessary.

Consider not requiring the `revoker_role` account to be mutable to increase the clarity and efficiency of the codebase.

Update: *Resolved at commit 3ef940d.*

L-06 Insufficient Governance Signers Validation

The `BootstrapAdmin` instruction of the `roles-registry` program requires at least 1 governance signer's signature in order to confirm the bootstrap operation. In the case where two signers are expected, the `require_two_signers` argument is set to `true`. However, the uniqueness of signers is not enforced correctly, as it is only required that the second provided signer account has signed the transaction, and it is not checked that this account is different from that of the first governance signer.

Consider enforcing the condition that governance signers passed to the `BootstrapAdmin` instruction are different in the case when two signatures are required.

Update: *Resolved at commit 23e042c.*

L-07 Misleading Errors

In the `remove_child` function, whenever an arithmetic underflow happens during child removal, the `ArithmeticOverflow` error is reported, while the root cause of this error is

arithmetic underflow. The same is true for the errors reported in the `decrement_admin_count` function and in the `decrement_delegations` function.

Consider fixing the instances of misleading errors listed above in order to improve the clarity of the codebase.

Update: *Resolved at commit 0106383.*

L-08 Over-Allocated Accounts

Throughout the codebase, multiple instances of allocating more-than-necessary space to accounts were identified:

- `MintConfigPDA` accounts get an allocation of 150 bytes, whereas only 119 bytes are necessary.
- `RolePDA` accounts get extra 50 bytes of allocation space.

However, it is not necessary to allocate these extra buffers, as in case when additional fields are to be added to these accounts, the accounts may be resized at any moment. Furthermore, if there is ever a necessity to insert additional data to these accounts, it would require tracking all accounts and applying changes manually, which is a costly and error-prone work. Hence, it is recommended to initialize accounts with all the necessary fields right from the beginning.

Consider only allocating the necessary space for each account in order to reduce the rent cost needed for them and to increase clarity of the codebase. Additionally, consider identifying which fields will be necessary for each account upfront in order to avoid having to update all accounts later on.

Update: *Resolved at commits a57652e and 9c6579d.*

L-09 Multiple Roles Possible for the Same Account and `mint`

Whenever an account is granted a role, a PDA representing this role is created. The PDA is derived from the relevant `mint` account, the target account's address, and the role identifier. This ensures that an account cannot be granted the same role twice at the same time for a specific `mint` account. However, the `GrantRole` instruction handler does not verify that the target account does not possess any other role, hence it is possible that an account is granted two different roles, for example `Registrar` and `Issuer`, at the same time.

Consider verifying that whenever an account is granted a role, it does not possess any other role. This could be done by supplying PDA accounts representing all the remaining 3 roles and verifying that their data is empty, or by removing the role identifier from `RolePDA` derivation. If the latter solution is chosen, the `RolePDA` existence and non-emptiness at a given address cannot longer be used to guarantee that a given entity possesses certain role and the `RolePDA.role` will need to be used for verification instead. Alternatively, if an account having multiple roles at the same time is acceptable, consider documenting it in the comments.

Update: *Resolved at commit 5735e28. The design choice of allowing the same account to have multiple roles for a single `mint` account is now documented.*

L-10 Incorrect PDA Derivation

The `derive_role_pda` helper function derives the role PDA account address from the specified `mint` account, role name, and wallet address. However, the derived address does not correspond to the actual PDA address generated in the code, as it uses `role_name` directly for derivation, whereas the role enum value should be used instead, as can be seen in case of `GrantRole` instruction. Furthermore, the code and comments in the `pda.rs` file refer to the `GLOBAL_SCOPE` constant which is not referred to anywhere else in the code and the key of `mint` account is always used for `RolePDA` address derivation instead.

Consider modifying the `derive_role_pda` function so that it returns correct role PDA addresses and removing the unused `GLOBAL_SCOPE` constant. Comments inside the `pda.rs` file will need to be changed accordingly.

Update: *Resolved at commit 5735e28.*

L-11 Manual Constraints Validation

Throughout the codebase, multiple instances of certain constraints being manually validated in instruction handlers instead of using Anchor helpers were identified:

- `registrar_role` account validation in the `MintSbt` handler could either be performed by the `seeds` constraint and requiring that the PDA has been derived from correct `role`, `wallet` and `mint`, or by using the `constraint` Anchor constraint. The same is true for the similar validation in the `BatchMintSbt` handler.
- Account ownership checks in `MintSbt` handler and in `BatchMintSbt` handler could be performed automatically by Anchor by using constraints and account types from the

`token_interface` module. This is also the case for other validation performed for mint and associated token accounts, such as token balance validation.

- All instructions of the `fund-ops` program, except the `AssertInvariants` instruction, perform extensive checks on the provided accounts inside their handlers. Their logic could be significantly simplified by using Anchor constraints instead.

Consider replacing the manual accounts validation and deserialization with Anchor's constraints and account wrappers whenever possible in order to improve the clarity and security of the codebase.

Update: *Partially Resolved at commits 0106383, 62cef9e, fedb1c4, and 0630b43. Now, anchor wrappers and constraints could also be used for mint and token accounts. The WisdomTree team stated:*

> *We have migrated a meaningful portion of manual validation to Anchor account wrappers + constraints (including PDA seed validation and cross-program PDA validation where applicable), reducing bespoke checks. Some mint/token-account validation remains manual (Token-2022 mint/token/ATA specifics and cases that flow through `remaining_accounts`). This aligns with OpenZeppelins's note that mint/token accounts could be further moved into wrappers. Where we keep manual checks, they are explicit and defensive (token program ownership checks, ATA/mint relationships, safe deserialization). These are bounded to specific handlers and are covered by negative tests for incorrect accounts.*

L-12 SBT Decimals Not Validated

The `MintSbt` instruction of the `sbt-ops` program invokes the `MintToChecked` instruction of the SPL Token-2022 program in order to mint SBT token to a given recipient. However, the documentation states that SBT tokens are always required to have 0 decimals, but this is not enforced by the `MintSbt` instruction, which always extracts decimals number from the `mint` account corresponding to the SBT token instead. As a result, this instruction allows for minting SBT tokens with any number of decimals. A similar issue is present in the `BatchMintSbt` instruction.

Consider requiring the decimal number for the provided SBT mint account to be equal to 0 in `MintSbt` and `BatchMintSbt` instructions.

Update: *Resolved at commit 7c230e6.*

L-13 Incorrect Roles Enforcement

The `roles-registry` program defines 4 roles with different privileges. The `DefaultAdmin` role enables granting and revoking any role, the `DelegatedAdmin` admin role enables granting and revoking the `Registrar` and `Issuer` roles, which in turn enable performing various operations on fund tokens and SBT tokens.

However, the access control for these last two roles is not enforced correctly, as an entity with the `Registrar` role can perform all token-related operations, whereas minting fund tokens should only be restricted to entities with the `Issuer` role. In fact, the `Issuer` role is not used anywhere outside the `roles-registry` program. Hence, it does not provide any privileges.

Consider clearly listing all privileges reserved for each role and updating the code to match this documentation. If the current code behavior is intended, consider fixing the comment.

Update: *Resolved at commits 5735e28 and 8658959.*

L-14 Frozen SBT Accounts Are Not Rejected by Transfer Hook

The `transfer-hook` program's `execute` function validates that the transacting parties hold a required Soulbound Token (SBT) by checking the balance of their SBT token accounts. This check is fundamental to the program's compliance enforcement mechanism. However, the internal `check_sbt_balance` function, used by the `execute` handler, only verifies the mint account and balance of the provided SBT accounts.

In particular, it does not check if the account is in a frozen state. This allows a user whose SBT account has been frozen, a potential method for temporarily freezing compliance status, to still pass the transfer hook check, undermining the ability to disable accounts via the SPL token standard's freeze mechanism.

Consider adding an SBT account state check to the `check_sbt_balance` function in order to ensure that the token account is not frozen. Alternatively, if there are no plans to ever freeze SBT accounts, consider stating this fact in the documentation.

Update: *Resolved at commits 5735e28 and 0b4a7a9.*

L-15 [Phase 3] Configuration Considerations

In order to avoid front-running attacks on the `BootstrapAdmin` instruction of the `roles-registry` program, it is required that either mint or freeze authority of a given mint signs the instruction call. For fund tokens, both authorities correspond to PDAs of the `fund-ops` program and for the SBT tokens, the mint authority corresponds to the `sbt-ops` program.

However, neither the `fund-ops` nor the `sbt-ops` program exposes a mechanism to call the `BootstrapAdmin` instruction through them. As a result, it is not possible to invoke `BootstrapAdmin` instruction with relevant PDA signatures, and it is necessary to initially configure mint accounts with governance signer as mint or freeze authority, so that this governance signer can then sign a call the `BootstrapAdmin` instruction. Afterwards, the authorities for the mint account can be configured to match relevant PDAs of the `fund-ops` and `sbt-ops` programs.

Consider documenting this requirement and following it during mint accounts configuration in order to allow providing a signature of a relevant authority account to the `BootstrapAdmin` instruction.

Update: *Resolved at commits 78e5da4 and 47f44cf.*

L-16 [Phase 3] `delegation_count` Could Be Re-Added to `RolePDA`

Removal of the functionality related to `DelegatedAdmin` role involved also removing the `delegation_count` field of the `RolePDA` struct. This field has been previously used to keep track of how many roles have been granted by each of delegated admins.

However, in case when the `DelegatedAdmin`-related functionality is re-introduced to the codebase, it will require updating all `RolePDA` accounts created so far to contain this member, which is a tedious and error-prone process, involving necessity of resizing accounts and paying extra rent for the extension.

Consider re-introducing the `delegation_count` field to the `RolePDA` struct in order to allow adding the `DelegatedAdmin`-related functionality without the necessity of modifying many already existing accounts.

Update: *Acknowledged, not resolved. The WisdomTree team does not have any plans to reintroduce the `DelegatedAdmin` role in the foreseeable future.*

L-17 [Phase 3] Enum Values Mismatch

The values representing role identifiers declared in the `roles-registry/src/constants.rs` file do not match the ones declared in the `state.rs` file.

Consider modifying these values so that they correspond to the values specified in the `Role` enum.

Update: *Resolved at commit 77fd599.*

Notes & Additional Information

N-01 `Option<Account<'info, T>>` Could Be Used for Optional Accounts

Throughout the codebase, multiple instances of instructions accepting optional accounts were identified (e.g., the `GrantRole` instruction). However, the code manually handles these accounts by verifying the ownership and other constraints and deserializing the data even though this could be done in a simpler way through the `Option` account wrapper in Anchor. Doing so would significantly simplify the code as there would be no need to manually reimplement each check.

Consider using the `Option` account wrapper in Anchor for optional accounts in the entire codebase. For each such account, consider applying relevant Anchor constraints so that each optional account is properly validated if provided.

Update: *Resolved at commits 6edc3f6, 1a1b799, and 0630b43.*

N-02 Redundant Accounts

Throughout the codebase, multiple instances of accounts being required but not used in instructions were identified:

- The `parent_wallet` account is unnecessary in the `InitializeChildren` instruction as the `_parent_wallet` argument could be used instead for PDA address generation.
- The `system_program` account is not needed in `MintSbt` and `BatchMintSbt` instructions of the `sbt-ops` program. It is also not needed in `MintFundTokens` and `BatchMintFundTokens` instructions of the `fund-ops` program.
- The recipient accounts passed to the `BatchMintSbt` instruction are not needed as their `Pubkeys` are already present in the `recipients` `Vec` passed to the instruction as an argument.

Consider removing unused accounts from the instructions in order to improve code clarity and efficiency.

Update: *Resolved at commits ba07c66 and 0f602c1.*

N-03 Non-Uniqueness of `Clock.unix_timestamp`

Throughout the codebase, multiple instructions, such as the `GrantRole` instruction, rely on the `unix_timestamp` field extracted from the `Clock` sysvar. However, the `unix_timestamp` gives the time in seconds, while each slot on Solana spans over 400ms. This means that it is possible for several consecutive blocks to share the same `unix_timestamp` value. Hence, the timestamp does not uniquely identify the block where a certain event happened.

Consider using the `slot` field of the `Clock` sysvar instead of or in addition to `unix_timestamp`. Doing so would help store complete information about when certain changes had been made.

Update: *Acknowledged, not resolved. The WisdomTree team stated:*

> *We do not use `unix_timestamp` as a uniqueness primitive, nonce, authorization input, or correctness gate. It is metadata only. Canonical ordering/auditability is derived from transaction signatures and slot-based indexing off-chain. We can add `Clock.slot` alongside timestamps if/when we store time as an ordering/uniqueness requirement.*

N-04 Redundant Instruction Macro Argument

The `#[instruction(...)]` Anchor macro may be used in order to specify instructions' arguments that can be utilized in the Anchor constraints defined in `#[derive(Accounts)]` structs. However, the `target_admin` argument, which is specified in the `#[instruction(...)]` macro of the `BootstrapAdmin` instruction of the `roles-registry` program, is not used for any constraints in the `BootstrapAdmin` struct.

Consider removing the redundant `#[instruction]` macro arguments from the instruction context structs where they are not directly used.

Update: *Resolved at commit 4b9a17a.*

N-05 Redundant Arguments

Throughout the codebase, there are multiple instructions that receive arguments that could be easily extracted from the supplied accounts or are not used or are unneeded in these instructions to begin with:

- The `target_admin` argument value may be easily extracted from the `target_admin` account.
- The `_parent_wallet` argument is not used.
- The `require_two_signers` argument determines if 1 or 2 governance signers' signatures are required for the `BootstrapAdmin` instruction. However, according to the emitted message its value is false only during testing. As such, it should be removed in production and tests could easily be adjusted to always provide two signatures. This would not only remove unused argument but would also ensure that tests cover the execution paths of the real program.
- The `cascade` argument is only used in the `RevokeRole` handler in order to emit logs, where it does not provide any additional context as cascade operation is automatically performed whenever possible, despite this argument's value.

Consider removing unnecessary arguments and accounts from instructions in order to decrease space necessary to encode each instruction invocation in transactions.

Update: *Resolved at commits e248cfb, 99aa6ac, and 66a8ac2.*

N-06 Redundant Accounts' Fields

Throughout the codebase, multiple instances of accounts with fields that are not used anywhere were identified:

- The `MintConfigPDA` account contains the `initialized` member, which is set to `true` whenever a `BootstrapAdmin` instruction is invoked for a new `mint` account. However, there is no scenario where this field can be initialized to `false` and whenever the `mint_config` account has nonzero data size, it will always be initialized to `true`. Because of that, this field, along with the unused `is_initialized` helper function, is redundant and can be removed as a check of `mint_config` address combined with a non-zero data size check could be used instead to determine if `MintConfigPDA` account has been initialized.
- The `bump` member of `MintConfigPDA` is initialized, but not used anywhere. The same is true for the `bump` member of `RoleChildrenPDA` and the `bump` member of `RolePDA`.
- None of the fields of the `ExtraAccountMetaList` struct are used as this struct is not used at all in the codebase.

Consider removing the unnecessary fields of the accounts in order to reduce the rent cost of storing the accounts on-chain. Additionally, consider using all the necessary ones.

Update: *Resolved at commit 417e6d6.*

N-07 Unused Functions

Throughout the codebase, multiple instances of unused functions were identified:

- the `increment_delegations` function
- the `decrement_delegations` function
- the `is_initialized` function
- the `role_can_have_children` function
- the `get_pda` function

To improve the clarity and maintainability of the codebase, consider removing all the functions that are never intended to be used and utilizing the rest.

Update: *Resolved at commit 66a8ac2.*

N-08 Redundant Checks

Throughout the codebase, multiple instances of redundant checks were identified:

- This check verifies that the `parent_role` account's `role` member is of `DelegatedAdmin` type. However, this check is redundant as it is already ensured by the derivation of the `parent_role` account, where the role is used as one of the seeds, and it is not possible for an account derived from a certain role to store another role value. Similarly, these checks [1, 2] verifying the `role` field to equal `Registrar` or `DefaultAdmin` are redundant.
- The checks verifying the `granter_role` PDA are unnecessary as the PDA itself has been derived from the values being compared.

Consider removing the redundant checks in order to improve the clarity and maintainability of the codebase.

Update: *Resolved at commit b0d2d02. The checks against the `Registrar` and `DefaultAdmin` roles present in the `fund-ops/state.rs` file have been deliberately retained.*

N-09 Misleading Comments

Throughout the codebase, multiple instances of misleading comments were identified:

- This comment suggests that the `BootstrapAdmin` instruction may be called only once, which is ensured by the `init` constraint. However, the `init` constraint in this case only ensures that the `BootstrapAdmin` instruction is called only once per `mint` account, not once in total.
- This comment states that the space allocated for `RolePDA` accounts equals 150 bytes, but in reality, 168 bytes are allocated.
- This comment claims that the current code approach for managing roles relies on off-chain cycle detection and avoidance. However, in the current implementation, no cycles in the roles graph are possible as the same role cannot be granted twice to one account at the same time and additionally, accounts with the `DelegatedAdmin` role cannot grant the `DelegatedAdmin` or `DefaultAdmin` role.
- These comments [1, 2] suggest that 4 different checks are performed for the `Registrar` and `DefaultAdmin` roles. However, in the actual code, point 2 from this list is split into two, hence the numbers from the subsequent comments do not match the ones listed at the beginning of the handler.

- This comment states that the `mint` account's owner is validated in the `InitializeConfig` instruction, but it is not the case in the code.

Consider rephrasing the comments mentioned above in order to improve the clarity and maintainability of the codebase.

Update: *Resolved at commit 636adf8.*

N-10 Naming Suggestion

The `admin_count` account passed to the `RevokeRole` instruction corresponds to the `MintConfigPDA` account, and inside the `RevokeRole` handler, this account is referred to as `mint_config_info`. As such, it could be renamed to `mint_config`, which would better describe the type of data it stores.

Consider renaming the `admin_count` account the `RevokeRole` instruction in order to improve the clarity of the codebase.

Update: *Resolved at commit 03ae0d8.*

N-11 Usage of `AccountInfo`

Throughout the codebase, there are multiple instances where the `AccountInfo` wrapper is used, for example for the `admin_count` account in the `RevokeRole` instruction. However, the usage of `AccountInfo` is discouraged and `UncheckedAccount` should be used instead where necessary.

Consider using `UncheckedAccount` wrapper instead of `AccountInfo` in case where higher-level account wrappers are not suitable.

Update: *Resolved at commits 06869e2 and 403f83a.*

N-12 Redundant `owner` Constraint

The `GrantRole` instruction of the `roles-registry` program expects to receive several PDA accounts belonging to that program. However, these accounts specify the `owner = crate::ID` constraint, which is redundant in this context, as `crate::ID` is the default value for `owner` when the `init` constraint is specified. In the case when it is not present, it is

determined to be the program which defined the underlying account's data type, which is equal to the `roles-registry` program in this case.

Consider removing redundant constraints from accounts in order to improve the clarity of the codebase.

Update: *Resolved at commit 62810b7.*

N-13 Unreachable Code

Throughout the codebase, multiple instances of unreachable code were identified:

- The `RevokeRole` instruction's handler contains code that allows for breaking a loop over child accounts early in case the current index is greater or equal to `target_children.child_count`. However, assuming that `child_count` is always updated correctly when adding and removing children, this should never happen as it will always correspond to the `len` of the `children` `Vec`.
- The `break` statement will never be executed as `contexts_len - 1`, which bounds the value of `i` in the loop is guaranteed to be lower than `MAX_CONTEXTS`.
- The `BatchFreeze` instruction of the `fund-ops` program contains the conditional logic related to scenarios when the `freeze_account` builder returns an error and when the external call fails. However, an error can be returned from the `freeze_account` helper function only when the specified token program is incorrect, but the correct token address is specified for each call. Moreover, whenever a program called through CPI reverts or returns an error on Solana, the entire transaction unconditionally reverts, so the code will never execute. As such, the `success_count` variable is redundant as it will always be equal to the batch size if the transaction succeeds. The same problems are present in the `BatchThaw` instruction, where additionally, an error in `require` statement will never be reported as it is already guaranteed that at least one CPI call will be made.
- The `if` statement will always evaluate to `true`, as whenever the `recipient_has_sbt` variable is set to `true`, the loop is terminated, so it is not possible for another branch of the code to be executed.

Consider removing all instances unreachable code in order to improve the clarity and maintainability of the codebase.

Update: *Resolved at commit 295c0d8.*

N-14 Unused Errors and Events

The `errors.rs` file and the `events.rs` file of the `roles-registry` program contain declarations of many errors and events related to compliance and role management. However, a lot of these errors and events, such as `RoleNotFound` or `AccountThawed`, are not used anywhere in the codebase.

Consider removing all instances of unused errors and events that are not intended to be used and using all the remaining ones in order to improve clarity of the codebase.

Update: *Acknowledged, not resolved. The WisdomTree team stated:*

> *This is a code-hygiene/clarity item with no runtime/security impact. We are intentionally deferring this cleanup to avoid churn in the IDL/client surface until integration finalizes.*

N-15 Confusing Error Numbers

The `ComplianceError` enum contains definitions of multiple errors used in the codebase. These errors are indexed from 5000. This value is likely chosen to avoid conflicts with internal Anchor errors. However, whenever Anchor parses error definitions, it avoids conflicts anyway by adding 6000 to each user-space error offset which was defined using `#[error_code]` macro without the `offset` parameter. Hence, the defined error codes start with 11000, which is confusing. A similar issue also exists for other error-related enums in the codebase, specifically for `fund-ops` and `transfer-hook` programs.

Consider using smaller errors' offsets in order to improve the clarity of the codebase.

Update: *Acknowledged, not resolved. The WisdomTree team stated:*

> *Renumbering is compatibility-breaking for monitoring/log parsing and current operational runbooks keyed on numeric codes. We mitigate this by decoding symbolically (error name via IDL/tooling) and documenting the effective mapping. We can revisit smaller offsets in a coordinated major revision window.*

N-16 `BatchMintSbt` Instruction Is Not Scalable

The `BatchMintSbt` instruction of the `sbt-ops` program allows minting for SBT tokens in batch. The maximum amount of tokens which can be minted at the same time is 20. However, if all accounts are passed without using Address Lookup Tables (ALTs), this number of requested mints would result in exceeding the transaction's available space on Solana equal to

1232 bytes, as each element of the `recipients` `Vec` requires 32 bytes for each requested mint, and for each such mint, two additional accounts are required, resulting in extra 64 byte overhead.

This means that if ALTs are not used, the maximum possible recipients number does not exceed 10 and the unused transaction space used is not significantly bigger than if multiple `MintSbt` instructions were included in a single transaction. Furthermore, even though ALTs can be used to increase number of recipients in batch mint and thus to achieve the number of 20, this does not actually increase efficiency as each such operation would require additional transactions to create and fill ALTs with required accounts. This is because mint recipients and their ATAs will be different each time. On top of that, rent will need to be deposited for each ALT being used.

Finally, since recipients and their SBT accounts are passed to the `BatchMintSbt` instruction using the `remaining_accounts`, it is not possible to use Anchor constraints for them and the entire validation of these accounts needs to be done manually, increasing error surface and making the code less readable.

Consider removing the `BatchMintSbt` instruction from the `sbt-ops` program and batch multiple `MintSbt` instructions instead (if needed) in order to simplify the code and making it less error-prone.

Update: *Resolved at commit fedb1c4.*

N-17 Helper Function Could Be Used for Instruction's Payload Creation

The `RotateAuthority` instruction of the `fund-ops` program allows for changing the authority for mint and freeze operations for a fund token. This is done by calling the `SetAuthority` instruction of the Token 2022 program. However, the input for this instruction is constructed manually, which is an error-prone process, despite there being a dedicated helper function available for that purpose.

Consider using the `set_authority` helper function for creating an input for the `SetAuthority` instruction in order to simplify the code and reduce the potential error surface.

Update: *Resolved at commit 456e5bf.*

N-18 Programs Structure Inconsistency

The codebase contains the implementation of four different programs. In the case of the `sbt-ops` and `transfer-hook` programs, all handlers and instruction-related structs are contained in the `lib.rs` files. However, in the case of the `fund-ops` program, the handlers for all the instructions are defined in separate files contained in the `handlers/` directory. On the other hand, in the case of the `roles-registry` program, the instruction-related structs are additionally separated and stored in the `instructions/` directory.

Consider unifying the convention for defining instruction handlers and inputs throughout the codebase. In order to increase readability of the codebase, the approach from the `roles-registry` program, with separate directories for handlers and instruction inputs, could be implemented in the remaining programs.

Update: *Acknowledged, not resolved.*

N-19 Logs Emitted When Values Do Not Change

The `UpdateConfig` instruction of the `transfer-hook` program allows the caller to change the `primary_sbt_mint` and `enforce_sbt_on_admin_ops` fields of the `ComplianceConfigPDA` account. However, it is not checked if the new values are different from the old ones and the logs are always emitted whenever new values are not `None`, which could be confusing for off-chain listeners.

Consider only emitting relevant logs when the new values are different from the old ones in the `UpdateConfig` instruction.

Update: *Resolved at commit 62810b7.*

N-20 [Phase 3] No Ability to Freeze SBT Tokens Through `sbt-ops`

The `sbt-ops` program allows to mint SBT tokens by accounts with the `Registrar` role. This is possible as each SBT token will have its mint authority set to a PDA of this program.

However, the `sbt-ops` program does not currently have an ability to freeze SBT tokens, which is an expected use case for temporarily blocking transfers from a given account. While this can still be possible by registering another entity as freeze authority, it should be clearly documented if this is an intended design.

Consider introducing a way to freeze SBT tokens through the `sbt-ops` program, in a similar way as this is handled in the `fund-ops` program. Alternatively, if freeze authority is expected to not be an `sbt-ops` program's PDA, consider documenting this design choice and configure mint account appropriately.

Update: *Resolved at commit 78e5da4.*

N-21 [Phase 3] Misleading Error

The `RevokeRole` instruction of the `roles-registry` program reverts with the `Unauthorized` error in case when an attempt is made to revoke the last default admin.

However, this error does not sufficiently describe the revert reason as the same error happens in case when an attempt is made to revoke a role without having enough privileges.

Consider reverting with more informative error on attempts to revoke the last default admin.

Update: *Resolved at commit 10c931d.*

Client Reported

CR-01 `ComplianceConfigPDA` Deserialization Failure

The `MintFundTokens` instruction of the `fund-ops` program attempts to deserialize the config account's data by using the `try_from_slice` method. However, this deserialization method does not account for the 8-bytes discriminator, which is appended by Anchor for each account at the beginning of its data. As a consequence, such deserialization attempts will result in an error, reverting the entire transaction.

Consider importing the `ComplianceConfigPDA` struct from the `transfer-hook` program and using it inside a relevant Anchor account wrapper for automatic deserialization of the `config` account's data in the `MintFundTokens` instruction.

Update: *Resolved at commit cd58566.*

CR-02 Double Account's Data Borrow

The `MintFundTokens` instruction of the `fund-ops` program invokes the `MintToChecked` instruction of the Token-2022 program in order to mint fund tokens to the specified recipient. However, before that call, it borrows the `mint` account's data and, since this account is passed as mutable to the `MintToChecked` instruction, an automatic check is performed to ensure that this account's data may be mutably borrowed. At this point, since the data is already borrowed, the check fails, resulting in the entire transaction reverting.

Consider dropping the reference to the `mint` account's data before performing the CPI call to the Token-2022 program.

Update: *Resolved at commit cd58566.*

Conclusion

The WisdomTree Solana tokenization protocol demonstrates a thoughtful adaptation of institutional fund token requirements to Solana's programming model. By leveraging Token-2022 extensions for core functionality and confining custom logic to tightly scoped programs (`roles-registry`, `fund-ops`, `transfer-hook`, and `sbt-ops`), the architecture reduces attack surface while preserving operational flexibility for regulated issuance, compliance enforcement, and administrative control.

The audit identified opportunities to further strengthen the codebase, particularly around account validation. During the course of the audit, many of these enhancements were implemented, and test coverage was meaningfully improved. Continued refinement through improved test coverage, with particular attention to edge cases involving incorrect account inputs and integration tests, would greatly enhance clarity, reduce operational complexity, and support long-term maintainability as the system evolves. Restructuring the codebase to fully leverage the Anchor framework's built-in validation capabilities would provide similar benefits.

The WisdomTree team is appreciated for their active engagement throughout the audit, their responsiveness to the audit findings, and for providing exceptional documentation that gave clarity regarding business requirements and expected code behavior. This collaborative approach has resulted in meaningful security improvements that have enhanced the protocol's readiness for institutional and/or retail use.

Appendix

Issue Classification

OpenZeppelin classifies smart contract vulnerabilities on a 5-level scale:

- Critical
- High
- Medium
- Low
- Note/Information

Critical Severity

This classification is applied when the issue's impact is catastrophic, threatening extensive damage to the client's reputation and/or causing severe financial loss to the client or users. The likelihood of exploitation can be high, warranting a swift response. Critical issues typically involve significant risks such as the permanent loss or locking of a large volume of users' sensitive assets or the failure of core system functionalities without viable mitigations. These issues demand immediate attention due to their potential to compromise system integrity or user trust significantly.

High Severity

These issues are characterized by the potential to substantially impact the client's reputation and/or result in considerable financial losses. The likelihood of exploitation is significant, warranting a swift response. Such issues might include temporary loss or locking of a significant number of users' sensitive assets or disruptions to critical system functionalities, albeit with potential, yet limited, mitigations available. The emphasis is on the significant but not always catastrophic effects on system operation or asset security, necessitating prompt and effective remediation.

Medium Severity

Issues classified as being of medium severity can lead to a noticeable negative impact on the client's reputation and/or moderate financial losses. Such issues, if left unattended, have a moderate likelihood of being exploited or may cause unwanted side effects in the system.

These issues are typically confined to a smaller subset of users' sensitive assets or might involve deviations from the specified system design that, while not directly financial in nature, compromise system integrity or user experience. The focus here is on issues that pose a real but contained risk, warranting timely attention to prevent escalation.

Low Severity

Low-severity issues are those that have a low impact on the client's operations and/or reputation. These issues may represent minor risks or inefficiencies to the client's specific business model. They are identified as areas for improvement that, while not urgent, could enhance the security and quality of the codebase if addressed.

Notes & Additional Information Severity

This category is reserved for issues that, despite having a minimal impact, are still important to resolve. Addressing these issues contributes to the overall security posture and code quality improvement but does not require immediate action. It reflects a commitment to maintaining high standards and continuous improvement, even in areas that do not pose immediate risks.