CODESPECT

Canopy program

SECURITY ASSESSMENT REPORT

January 5, 2026

Prepared for:

canopy

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 Solana program of Canopy. Canopy is an investment platform that enables secure, decentralised management of investment opportunities (called "Plots"). The platform facilitates the complete investment lifecycle from opportunity creation to token distribution, with comprehensive security controls, role-based access management, and multi-token support.

This audit focuses on the first version of the Canopy program.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of smart contracts, execution testing.

CODESPECT found 47 points of attention, four classified as `High`, eight classified as `Medium`, twelve classified as `Low`, sixteen classified as `Informational`, and seven classified as `Best Practices`. All of the issues are summarised in Table 2.

Organisation of the document is as follows:

- **Section 5** summarizes the audit.
- **Section 6** describes the functionality of the code in scope.
- **Section 7** presents the issues.
- **Section 8** presents the issues discovered during fix review phase.
- **Section 9** discusses the documentation provided by the client for this audit.
- **Section 10** presents the compilation and tests.

Issues found:

Severity	Unresolved	Fixed	Acknowledged
High	0	4	0
Medium	0	8	0
Low	0	12	0
Informational	0	16	0
Best Practices	0	7	0
Total	**0**	**47**	**0**

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

Fix Review Issues found:

Severity	Unresolved	Fixed	Acknowledged
Medium	0	3	0
Low	0	7	0
Informational	0	2	0
Total	**0**	**12**	**0**

Table 3: Summary of Unresolved, Fixed, and Acknowledged Issues

5 Audit Summary

Audit Type	Security Review
Project Name	Canopy
Type of Project	Investment Platform
Duration of Engagement	19 Days
Duration of Fix Review Phase	3 Days
Draft Report	December 18, 2025
Final Report	January 5, 2026
Repository	canopy
Commit (Audit - Part 1)	d17092d2a2aab52b0f79634f90bd8890391111e5
Commit (Audit - Part 2)	234cd9ca0ca8348e466548513d72a2aba65e84e4
Commit (Final)	f264403f43335205b57cf047870f672266b97954
Documentation Assessment	Medium
Test Suite Assessment	Medium
Auditors	Shaflow, LordAlive

Table 4: Summary of the Audit

5.1 Scope - Audited Files

	File	LoC
1	utils/validation.rs	485
2	utils/safe_math.rs	248
3	utils/compute_units.rs	176
4	utils/time.rs	137
5	utils/logging.rs	127
6	utils/rate_limit.rs	112
7	utils/auth.rs	104
8	utils/emergency.rs	41
9	utils/mod.rs	16
10	instructions/tge/execute_airdrop.rs	407
11	instructions/tge/pull_vesting.rs	252
12	instructions/tge/claim_tokens.rs	225
13	instructions/tge/select_disbursement_method.rs	155
14	instructions/tge/setup_tge_distribution.rs	134
15	instructions/tge/update_tge_distribution.rs	93
16	instructions/tge/mark_tge_ready.rs	82
17	instructions/tge/mod.rs	14
18	instructions/watering/deposit.rs	354
19	instructions/watering/refund.rs	253
20	instructions/watering/batch_manage.rs	161
21	instructions/watering/manage.rs	102
22	instructions/watering/indicate.rs	50
23	instructions/watering/mod.rs	10
24	instructions/plot/create.rs	259
25	instructions/plot/conclude.rs	179
26	instructions/plot/update_status.rs	134
27	instructions/plot/update.rs	102
28	instructions/plot/accept.rs	88
29	instructions/plot/approve.rs	88
30	instructions/plot/update_investment_destination.rs	58
31	instructions/plot/mod.rs	14
32	instructions/time_lock/execute.rs	138
33	instructions/time_lock/propose.rs	124
34	instructions/time_lock/approve.rs	110
35	instructions/time_lock/cancel.rs	71
36	instructions/time_lock/mod.rs	8
37	instructions/platform/emergency_pause.rs	152
38	instructions/platform/initialize.rs	121
39	instructions/platform/update_fees_wallet.rs	47
40	instructions/platform/mod.rs	3
41	instructions/growth_cycle/update_metadata.rs	106
42	instructions/growth_cycle/create.rs	91
43	instructions/growth_cycle/mod.rs	4
44	instructions/admin/add.rs	71
45	instructions/admin/remove.rs	40
46	instructions/admin/disable.rs	37
47	instructions/admin/mod.rs	3
48	instructions/grove/create.rs	56
49	instructions/grove/pay_onboarding_fee.rs	55
50	instructions/grove/mod.rs	2
51	instructions/seedling/setup.rs	48
52	instructions/seedling/mod.rs	1
53	instructions/mod.rs	43
54	state/rate_limiter.rs	389
55	state/plot.rs	297
56	state/emergency_controls.rs	247
57	state/time_lock.rs	184
58	state/tge_distribution.rs	171
59	state/admin.rs	82
60	state/growth_cycle.rs	38

	File	LoC
61	state/grove.rs	36
62	state/fee_structure.rs	34
63	state/platform_collection.rs	32
64	state/watering.rs	31
65	state/mod.rs	30
66	state/receipt.rs	20
67	state/plot_escrow.rs	19
68	state/seedling.rs	17
69	state/platform_config.rs	9
70	error.rs	270
71	lib.rs	258
72	events.rs	215
	Total	**8370**

5.2 Findings Overview

	Finding	Severity	Update
1	Airdrop feature blocked due to missing airdrop_gas_reserve parameter in setup_tge_distribution(...) instruction	High	Fixed
2	Incorrect assignment of total_verified_deposits in select_disbursement_method(...) instruction causes distribution DoS	High	Fixed
3	Insufficient TokenAccount validation in the deposit(...) instruction	High	Fixed
4	The final_amount calculation is incorrect	High	Fixed
5	Incorrect authorization target check prevents valid admin actions in admin::remove(...) instruction	Medium	Fixed
6	Missing grove and seedling account check in the select_disbursement_method(...) instruction	Medium	Fixed
7	Missing growth_cycle account check in the deposit(...) instruction	Medium	Fixed
8	Missing grove-plot ownership verification in mark_tge_ready instruction	Medium	Fixed
9	The reserved gas is not refunded when changing the disbursement mode	Medium	Fixed
10	carry_fee may be lost	Medium	Fixed
11	time_lock_approval_threshold is not configured	Medium	Fixed
12	update_tge_distribution(...) instruction cannot modify VestingInfo in VestingIntegration mode	Medium	Fixed
13	Active status of platform_config is not being checked inside tge instructions	Low	Fixed
14	Active status of platform not checked during growth_cycle::update_metadata(...) Instruction	Low	Fixed
15	Improper access control in the update(...) instruction	Low	Fixed
16	Incorrect permission control in update_status(...) instruction	Low	Fixed
17	Insufficient holder_ata validation	Low	Fixed
18	Missing is_ready validation in execute_airdrop(...) and claim_tokens(...) instructions	Low	Fixed
19	Missing plot account check allows seeding_admin to approve mismatched proposals	Low	Fixed
20	Premature escrow validation misplaced in setup phase	Low	Fixed
21	The collection system has not been implemented	Low	Fixed
22	The security assumption of threshold-based approval proposals may fail	Low	Fixed
23	Unsafe use of try_deserialize	Low	Fixed
24	update_tge_distribution(...) permits unsafe updates after tge_distribution.is_ready	Low	Fixed
25	Inconsistent length validation in growth_cycle::create(...) instruction	Info	Fixed
26	Inconsistent maximum length constraints for the field	Info	Fixed
27	Inconsistent plot parameter validation	Info	Fixed
28	Incorrect groves.len() check	Info	Fixed
29	Incorrect seedling_admin check	Info	Fixed
30	Incorrect reduction of plot.collected_amount	Info	Fixed
31	Lacking an on-chain authorization check allows growth_cycle creation by anyone	Info	Fixed
32	Missing mut constraint on platform_collection	Info	Fixed
33	Redundant discriminator space allocation in platform::initialize(...) instruction for platform_collection account	Info	Fixed
34	Some fields in vesting_info may be inaccurate	Info	Fixed
35	The validation of grove_fees_destination is not implemented	Info	Fixed
36	The calculation of current_balance uses outdated data	Info	Fixed
37	The permission control comment does not match the implementation	Info	Fixed
38	emergency_pause(...) does not update platform active status	Info	Fixed
39	updated_at is not updated in the process_airdrop_batch(...) instruction	Info	Fixed
40	Unnecessary 64-byte validation for operation_data in time_lock::execute(...) instruction for OperationType::UpdateInvestmentDestination	Info	Fixed
41	Incorrect clearing of uri causes logging errors	Best Practices	Fixed
42	Perform the operation_data length check earlier	Best Practices	Fixed
43	Redundant code	Best Practices	Fixed
44	Redundant updates of updated_by and updated_at	Best Practices	Fixed
45	The time_lock status is not updated when it ready	Best Practices	Fixed
46	Unreachable state	Best Practices	Fixed
47	[Best Practise] Missing validation for default and duplicate values in platform::update_fees_wallet and grove::create	Best Practices	Fixed

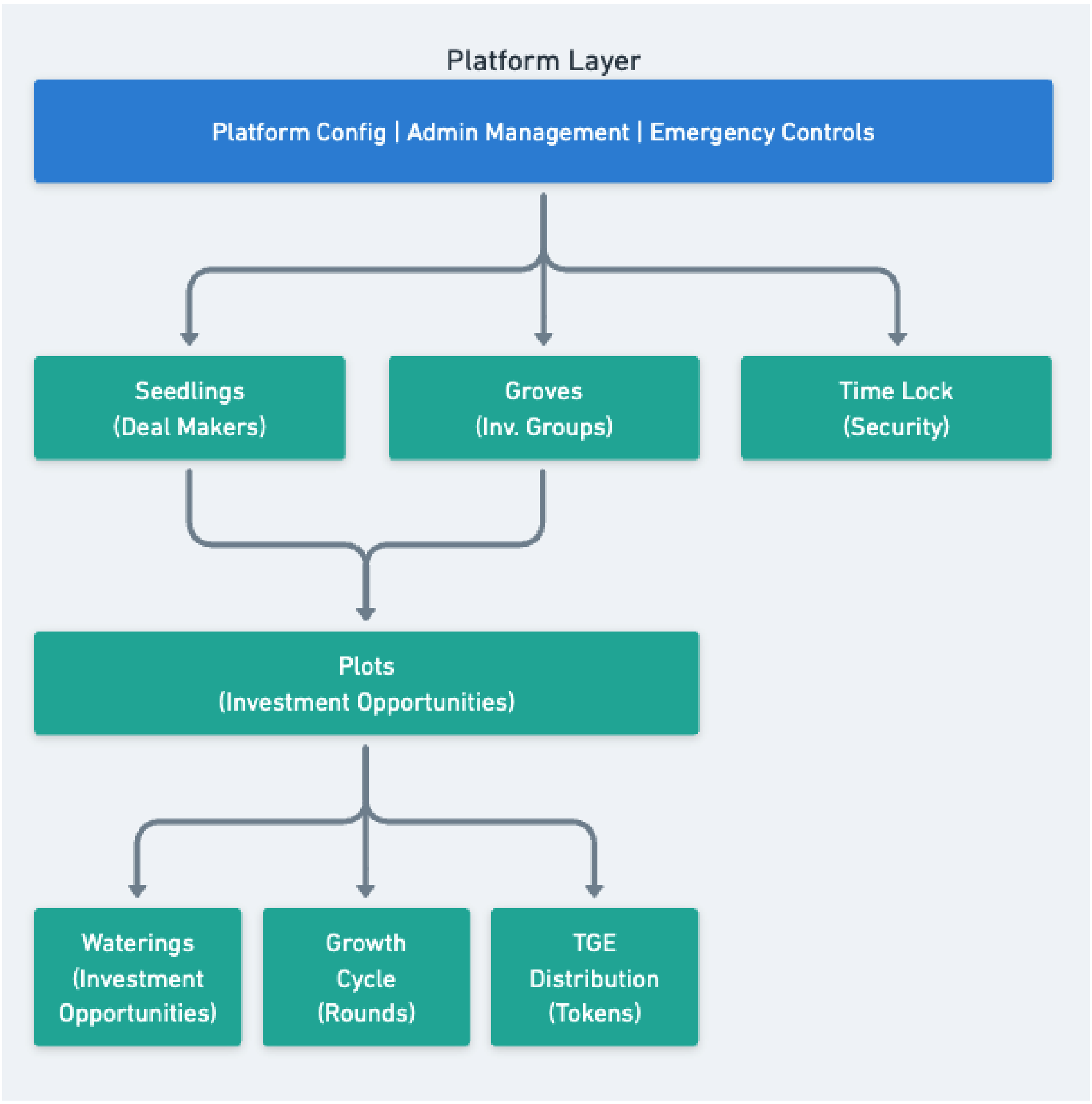
5.3 Findings Overview - Fix Review Issues

	Finding	Severity	Update
1	Incorrect `fee_wallet` check	Medium	Fixed
2	The `refund(...)` instruction cannot handle cases where the NFT has already been sold	Medium	Fixed
3	`carry_fee` may be lost	Medium	Fixed
4	Expired and unclaimed tokens cannot be withdrawn	Low	Fixed
5	Improper authorization in the `mark_tge_ready(...)` instruction	Low	Fixed
6	The `airdrop_gas_reserve` may not be refunded to the depositor	Low	Fixed
7	The `conclude(...)` instruction can be executed even when `investment_destination` is not set	Low	Fixed
8	The `remove(...)` instruction has insecure access control	Low	Fixed
9	Unsafe account creation	Low	Fixed
10	Updating the `royalty_wallet` does not take effect	Low	Fixed
11	The groves in `grove_list` may not be in an Active state	Info	Fixed
12	`select_disbursement_method(...)` does not restrict the plot state	Info	Fixed

6 System Overview

Canopy is a Solana investment platform that enables secure, decentralized management of investment opportunities (called "Plots"). The platform facilitates the complete investment lifecycle from opportunity creation to token distribution, with comprehensive security controls, role-based access management, and multi-token support.

The following diagram presents the high level architecture:


Platform Layer
Platform Config | Admin Management | Emergency Controls
Seedlings (Deal Makers)
Groves (Inv. Groups)
Time Lock (Security)
Plots (Investment Opportunities)
Waterings (Investment Opportunities)
Growth Cycle (Rounds)
TGE Distribution (Tokens)

The following points describe the relation model:

- **Platform** $\rightarrow$ **Admins:** 1:N (Platform has multiple admins)
- **Platform** $\rightarrow$ **Groves:** 1:N (Platform has multiple investment groups)
- **Platform** $\rightarrow$ **Seedlings:** 1:N (Platform has multiple deal sponsors)
- **Seedling** $\rightarrow$ **Growth Cycles:** 1:N (Each seedling runs multiple funding rounds)
- **Growth Cycle** $\rightarrow$ **Plots:** 1:N (Each round has multiple investment opportunities)
- **Plot** $\rightarrow$ **Waterings:** 1:N (Each plot has multiple investor participations)
- **Plot** $\rightarrow$ **Groves:** N:M (Plots can be offered to multiple groves)
- **Plot** $\rightarrow$ **TGE Distribution:** 1:1 (Each plot has one token distribution)
- **Watering** $\rightarrow$ **NFT Receipt:** 1:1 (Each investment has one transferable receipt)

6.1 Core Entities and State Management

6.1.1 Platform Configuration

The Platform Configuration serves as the global settings singleton for the entire system. It stores critical information including the fee wallet address (where platform fees are collected), the USDC mint address (primary token for investments), an emergency pause flag, and the time-lock approval threshold (defaulting to 3 admin approvals).

The emergency pause flag enables platform-wide shutdown capabilities, while the time-lock threshold controls how many admin approvals are needed for early execution of sensitive operations. All admin authorization checks reference this account, and the emergency control system validates the active status before allowing any operation to proceed.

6.1.2 Admin Management

The Admin system provides role-based access control with target-specific permissions. Each admin account contains the authority's wallet address, a list of up to five targets (Seedling, Grove, or Platform entities they can manage), timestamps for audit trails, and an active status flag that allows disabling without deletion.

The platform implements a three-tier authorization hierarchy. Platform Admins have full system access, can perform all operations, add or remove other admins, and approve time-locked operations. Their admin account contains the `platform_config` key in the targets list, granting them superuser status.

Seedling Admins can create and manage plots for their assigned seedlings, update growth cycles, but cannot modify other seedlings' data. Grove Admins can accept plots offered to their grove, approve or reject investor deposits, and manage plot status transitions for accepted plots. One admin can manage up to five entities simultaneously, and the system uses target-specific access validation where admins must have a target in their list to operate on it.

6.1.3 Grove

Groves represent investment groups or syndicates that participate in investment opportunities. Each Grove account stores a name, status, fee wallet address, onboarding fee amount, and platform fee structure.

Platform admins create new groves, which then must pay an onboarding fee to activate. Once active, grove admins can accept offered investment opportunities and collect carry fees from successful investments.

The fee structure includes platform fees (collected by the platform on all investments through this grove), grove carry fees (the grove's share of investment returns), and the one-time onboarding fee required to join the platform.

6.1.4 Seedling

Seedlings are deal sponsors who create investment opportunities on the platform. Each Seedling account contains a name, creation timestamp, creator address, and status.

The lifecycle follows the same progression as Groves. However, unlike Groves, Seedlings don't pay an onboarding fee and are managed exclusively by platform admins. Seedlings are responsible for creating growth cycles (funding rounds), creating plots (investment opportunities), concluding plots and distributing funds, and updating plot details throughout the investment lifecycle.

6.1.5 Growth Cycle

Growth Cycles represent funding rounds or investment campaigns managed by a seedling. Each cycle contains a reference to its parent seedling, a name, a metadata URI, creation timestamp and creator, and an active status flag.

These cycles serve as organizational units for grouping related plots. The metadata URI links to off-chain information. One seedling can run multiple growth cycles simultaneously, allowing for diverse investment campaigns.

6.1.6 Plot

The Plot entity is the core investment opportunity in the Canopy system. Each plot contains comprehensive financial data, escrow wallet references, and status tracking.

Core plot data includes the parent growth cycle reference, creator address, allocation amount (total to raise), minimum investment requirement, collected amount (running total), and total allocated amount (sum of all waterings). Timestamps track creation, updates, start date, and end date.

The plot uses three separate escrow wallets for fund isolation: the `investment_escrow` holds investor principal, the `platform_fees_escrow` accumulates platform fees, and the `grove_fees_escrow` accumulates grove carry fees. This separation ensures fee accounting, atomic fee distribution during conclusion, no mixing of investor funds with fees, and easier auditing and reconciliation. An optional investment destination wallet specifies where funds should be sent after collection.

Plots follow a strict state machine with eight possible statuses.

Authorization for status transitions is strictly controlled.

6.1.7 Watering

Waterings represent individual investor participation in a plot. Each watering tracks the parent plot, member (investor) address, requested allotment (amount desired), actual allotment (amount allocated by admin), invested amount (actually deposited), NFT receipt reference, status, and detailed fee tracking for refund calculations.

Key features include interest indication (investors express interest without depositing funds initially), admin allocation (admins can allocate amounts less than requested), deposit tracking (investor deposits allocated amount plus fees), NFT receipts (transferable investment positions that can be sold), detailed fee tracking (records platform and grove fees paid for accurate refund calculations), and comprehensive TGE tracking (monitors token distribution status).

6.1.8 TGE Distribution

The TGE Distribution entity manages token distribution to investors after investment conclusion. Each TGE Distribution contains the parent plot reference, token mint address, token vault (holding tokens for distribution), total token amount, disbursement method, readiness flags, distribution statistics, and carry rate configurations.

Three disbursement methods are supported. Manual Claim allows investors to call a claim instruction at their convenience, with gas costs paid by the claimant and proportional distribution based on investment amounts with automatic carry fee deduction. Airdrop enables admins to execute batch distributions, paying gas from a platform gas reserve, processing multiple investors per transaction, and automatically updating statuses. Vesting Integration uses Streamflow for time-locked distributions, creating individual vesting contracts per investor, with admins calling a `pull_vesting` instruction, and tokens locked in Streamflow contracts with predefined schedules. The last option is not fully implemented.

6.1.9 Time Lock

The Time Lock mechanism enforces a 24-hour delay on sensitive operations, preventing single-admin attacks and giving the community time to review and react to proposals. Each time lock contains the operation type, proposer identity, target entity, proposal timestamp, execution timestamp (typically 24 hours later), execution and cancellation flags, and a list of platform admin approvals.

Early execution is possible if the configured approval threshold is met (default: 3 approvals), allowing urgent legitimate changes while maintaining transparency since all proposals remain on-chain.

The security rationale is threefold: prevent single-admin attacks on critical functions, give the community time to identify and react to malicious proposals, and require multi-signature approval for bypassing delays, ensuring no single point of failure in governance.

6.1.10 Emergency Controls

Emergency Controls provide a platform-wide pause mechanism for security incidents. The control is embedded in the `PlatformConfig.is_active` flag, which is checked before any state-changing operation. When the platform is paused, all plot operations cease (creation, updates, status changes), investment operations halt (deposits and refunds), TGE distributions stop, and grove and seedling modifications are blocked.

Read operations continue to function during a pause, maintaining transparency. Emergency unpause operations can still be executed (through time-lock), and admin management remains available to restore access if needed.

Activation is asymmetric for safety: pause is immediate without time-lock delay (enabling rapid response to threats), while unpause requires time-lock (preventing premature restoration before issues are resolved). Only platform admins can pause, and unpause proposals must go through the standard time-lock approval process.

7 Issues

7.1 [High] Airdrop feature blocked due to missing `airdrop_gas_reserve` parameter in `setup_tge_distribution(...)` instruction

File(s): `setup_tge_distribution.rs`

Description: The `setup_tge_distribution` instruction does not accept or configure any parameters related to `airdrop_gas_reserve`. Because this value is never initialized or set during TGE setup, the subsequent `start_airdrop_handler` invoked within the `execute_airdrop` instruction cannot operate correctly. As a result, the airdrop flow becomes non-functional.

```
    pub fn handler(
     ctx: Context<SetupTgeDistribution>,
     _token_mint: Pubkey,
     distribution_mode: DistributionMode,
     carry_percentage: u16,
     token_symbol: String,
     claim_deadline: Option<u64>,
     vesting_info: Option<VestingInfo>,
) -> Result<()> {...}
```

Impact: The missing `airdrop_gas_reserve` configuration effectively blocks the entire airdrop feature. Any attempt to execute the airdrop will fail due to the absence of required state, preventing users from receiving their intended token allocations. This creates a critical functional gap in the **TGE** pipeline.

Recommendation(s): Introduce and validate an `airdrop_gas_reserve` parameter within the `setup_tge_distribution` instruction. Ensure that this value is properly initialized and stored in the distribution configuration so that `start_airdrop_handler` can access it during the airdrop execution process.

Status: Fixed

Update from CODESPECT: In the current system this issue cannot be overcomed through select_disbursement_method as it contains another major flaw which still allows a Denial of Service on execute_airdrop which is discussed inside issue 7.2.

Update from Canopy: `setup_tge_distribution` is no longer part of the codebase :)

7.2 [High] Incorrect assignment of `total_verified_deposits` in `select_disbursement_method(...)` instruction causes distribution DoS

File(s): `select_disbursement_method.rs`

Description: When executing the `select_disbursment(...)` instruction, the value of `tge_distribution.total_verified_deposits` is incorrectly set to `0` instead of being updated to `plot.collected_amount`. This misassignment corrupts the distribution state and prevents the system from recognizing the correct amount of verified deposits required for subsequent processing.

```
    //...

@>  tge_distribution.total_verified_deposits = 0;
    tge_distribution.airdrop_gas_reserve = estimated_gas_fee.unwrap_or(0);
    tge_distribution.price_oracle = None; // Can be set later for oracle-based carry

    //...
```

Here the `tge_distribution` of any distribution mode creates **Denial of Service**.

Impact: Setting `total_verified_deposits` to `0` can cause a complete denial-of-service across all distribution modes, as the protocol will treat the distribution as having no valid deposits. This blocks further execution of distribution-related flows and can halt token disbursement entirely.

Recommendation(s): Ensure that `tge_distribution.total_verified_deposits` is correctly assigned to `plot.collected_amount` within the `select_disbursment` instruction.

Status: Fixed

Update from Canopy: Fix commit: f92a1d12d36ce94e0ae38fc89e5ae4e25e307bd7 https://github.com/canopyfi/canopy/pull/75

Update from Canopy: https://github.com/canopyfi/canopy/commit/f92a1d12d36ce94e0ae38fc89e5ae4e25e307bd7

7.3 [High] Insufficient `TokenAccount` validation in the `deposit(...)` instruction

File(s): `deposit.rs`

Description: When creating a `plot`, the `investment_escrow`, `platform_fees_escrow`, and `grove_fees_escrow` accounts are initialized. These accounts are intended to receive the investment token (USDC) and fees. After the fundraising is completed, the `conclude(...)` instruction is called to transfer funds from these accounts.

```
#[derive(Accounts)]
pub struct Deposit<'info> {
    //...
    #[account(
        mut,
        constraint = investment_escrow.owner == escrow_wallet.key(),
        constraint = investment_escrow.mint == user_token_account.mint
    )]
    pub investment_escrow: Box<Account<'info, TokenAccount>>,

    // Platform fees escrow account
    #[account(
        mut,
        constraint = platform_fees_escrow.owner == escrow_wallet.key(),
        constraint = platform_fees_escrow.mint == user_token_account.mint
    )]
    pub platform_fees_escrow: Box<Account<'info, TokenAccount>>,

    // Grove fees escrow account
    #[account(
        mut,
        constraint = grove_fees_escrow.owner == escrow_wallet.key(),
        constraint = grove_fees_escrow.mint == user_token_account.mint
    )]
    pub grove_fees_escrow: Box<Account<'info, TokenAccount>>,
    //...
}
```

However, in the `deposit(...)` instruction, the `investment_escrow`, `platform_fees_escrow`, and `grove_fees_escrow` accounts are not verified to ensure they match the expected accounts in the plot.

Impact: An attacker could create new `TokenAccount` and pass them in when calling `deposit(...)`. This would cause the raised funds to be sent to unintended `TokenAccount`, leaving those funds stuck and unclaimable through the `conclude(...)` instruction.

Recommendation(s): Verify that the `investment_escrow`, `platform_fees_escrow`, and `grove_fees_escrow` accounts match the addresses in the corresponding fields of the `plot`.

Status: Fixed

Update from Canopy: Added 0d77137677bb65795476af0899fbece2445c598f https://github.com/canopyfi/canopy/pull/70

Update from Canopy: https://github.com/canopyfi/canopy/commit/0d77137677bb65795476af0899fbece2445c598f

Update from CODESPECT: Fixed

7.4 [High] The `final_amount` calculation is incorrect

File(s): `claim_tokens.rs`

Description: In the `claim_tokens(...)` instruction, the airdrop tokens to be distributed are calculated based on the proportion of the user's invested amount relative to the total invested token amount.

```
// Calculate total deposits from all investors for this plot
let total_deposits = plot.collected_amount;

require!(total_deposits > 0, ErrorCode::InvalidAmount);

// Calculate investor's proportional share (based on original deposit)
let investor_deposit = watering.deposit_amount;
let (investor_share, carry_fee, final_amount) = tge_distribution
    .calculate_final_amount(investor_deposit, total_deposits)?;
```

In the `calculate_final_amount(...)` calculation, it uses `watering.deposit_amount / plot.collected_amount`. However, `plot.collected_-amount` accumulates the `allocated_amount` of all users, while `watering.deposit_amount` is the sum of a single user's `allocated_amount` plus fees.

Impact: Because the numerator contains the additional fees while the denominator does not, the calculated `investor_share` for each user will be higher than expected. This may ultimately cause later claimants to fail in claiming due to insufficient tokens in the airdrop wallet.

Recommendation(s): It is recommended to use `receipt_metadata.investment_amount` as the numerator for the calculation.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/pull/74 with fix commited at b038c3a8ae0c5e87a1d0241e9bd0feb80fd593a2

Update from CODESPECT: Fixed

7.5 [Medium] Incorrect authorization target check prevents valid admin actions in `admin::remove(...)` instruction

File(s): `remove.rs`

Description: In the `remove(...)` instruction, an authorization check is performed to validate whether the caller has permission to `remove` target of the given admin. The first condition checks whether the calling admin has the target set to another admin's public key. However, by design, admin accounts do not hold other admin accounts as targets.

```
@> if !caller_admin.has_target(&admin.key())
        && !caller_admin.is_platform_admin(&platform_config.key())
   {
        return Err(error!(ErrorCode::Unauthorized));
   }
```

Targets are only defined for platform-level and domain-specific accounts such as `platform_config`, `growth_cycle`, or `seedling` etc. As a result, this authorization branch can never evaluate to true, making the logic ineffective and blocking otherwise valid admin operations.

Impact: This issue prevents legitimate target admins from successfully executing the `remove(...)` instruction, effectively breaking intended administrative functionality. It introduces an authorization deadlock where valid permissions cannot be exercised, resulting in operational disruption.

Recommendation(s): Update the authorization logic so that only admins who share the same valid target are permitted to execute the `remove(...)` instruction, or by the `platform_admin`.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/a0c25bc608d664dfa51dc5a7cf1848a68e99a84c

7.6 [Medium] Missing `grove` and `seedling` account check in the `select_disbursement_method(...)` instruction

File(s): `select_disbursement_method.rs`

Description: The `select_disbursement_method(...)` instruction is used to set the distribution method for the `tge`.

```
pub struct SelectDisbursementMethod<'info> {
    //...
    /// The grove that owns this plot (for authorization)
    pub grove: Account<'info, Grove>,

    /// The seedling that owns this plot (for authorization)
    pub seedling: Account<'info, Seedling>,
    //...
}
```

However, the `grove` and `seedling` accounts lack checks, so there is no guarantee that these two accounts match the fields in the `plot`.

Impact: These two accounts are used for permission verification. The lack of the above checks can lead to a permissions bypass, allowing `grove` and `seedling` accounts from other plots to call the `select_disbursement_method(...)` instruction.

Recommendation(s): For the `grove` account, use `plot.grove_list` for permission checks. For the `seedling` account, it is necessary to verify that it is associated with the `plot.growth_cycle` account.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/pull/72 8bb31895bc640d36a1af794201995b7d7d57fbb1

Update from CODESPECT: Fixed

7.7 [Medium] Missing `growth_cycle` account check in the `deposit(...)` instruction

File(s): `deposit.rs`

Description: The `deposit(...)` instruction is used to handle the deposit of watering accounts for a plot.

```
//...
 #[account(
        mut,
        constraint = plot.get_status() == PlotStatus::Collecting @ ErrorCode::PlotNotCollecting,
    )]
    pub plot: Box<Account<'info, Plot>>,

    /// The growth cycle this plot belongs to
    pub growth_cycle: Box<Account<'info, GrowthCycle>>,

    /// The seedling associated with the growth cycle
    pub seedling: Box<Account<'info, Seedling>>
//...
```

But, there is no check whether the `growth_cycle` account matches the corresponding `plot.growth_cycle`. So there is no guarantee that provided `growth_cycle` actually belongs to that `plot`.

Impact: The `growth_cycle` account is used in crucial places like `receipt_metadata` and Metaplex metadata uri. The lack of the above check can lead to minting NFT with wrong `growth_cycle` metadata. since the metadata URI, name, growth cycle name, and other display information are all taken from the unverified `growth_cycle` account. In the worst case, the attacker can attach their deposit to a more "premium" or high-value `growth_cycle`, making the NFT appear to represent a different investment. Further accounting based on `growth_cycle` can become completely corrupted.

Recommendation(s): Add the constraint where `growth_cycle` should match with the `plot.growth_cycle`

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/pull/71 merged at 892d0d68c0da5c9287cb6e351d24edb00a595f8e

Update from Canopy: https://github.com/canopyfi/canopy/commit/57836202e9b74bdbad8af975d01598bcc5f1c017

Update from CODESPECT: Fixed

7.8 [Medium] Missing grove-plot ownership verification in `mark_tge_ready` instruction

File(s): `mark_tge_ready.rs`

Description: The `mark_tge_ready` instruction in `mark_tge_ready.rs` fails to validate that the provided `grove` account actually belongs to the specified `plot` account. The instruction does not verify whether the grove's address exists in the `grove_list` of that plot, before allowing the operation to proceed.

```
/// The grove for authorization (must match plot's grove association)
pub grove: Account<'info, Grove>,



let is_platform_admin = AuthUtils::is_platform_admin(admin, &platform_config.key());
let is_grove_admin = admin.targets.contains(&grove.key());
let is_seedling_admin = admin.targets.contains(&seedling.key());
```

This means any arbitrary grove admin can call this instruction, even if the `grove` has no legitimate association with the given plot. The instruction will execute successfully as long as the grove account exists, regardless of whether it was actually associated with the specified plot.

Impact: Unauthorized grove admins can call `mark_tge_ready` for any given plot, bypassing intended access controls and violating the requirement that only plot-associated groves admins may modify TGE state. This allows unauthorized state transitions, compromises plot–grove data integrity, and breaks the core protocol invariant that only groves listed in `grove_list` of that plot are permitted to trigger plot-level state changes.

Recommendation(s): Add validation logic by taking in the `grove_list` of that plot to verify grove ownership before processing the instruction:

```
#[account(
    has_one = plot @ ErrorCode::GroveListDoesNotBelongToPlot
)]
pub grove_list: Box<Account<'info, PlotGroveList>>,


// Verify that the grove belongs to the grove_list of that plot
require!(
    ctx.accounts.grove_list.groves.contains(&ctx.accounts.grove.key()),
    ErrorCode::GroveNotInList
);
```

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/8769c59e05a7b13b818be17631294d96a7d32dcb

7.9 [Medium] The reserved gas is not refunded when changing the disbursement mode

File(s): select_disbursement_method.rs

Description: The `select_disbursement_method(...)` instruction uses the `init_if_needed` constraint on the `tge_distribution` account, which allows the account to be updated by calling this instruction even after it has been created.

```
pub struct SelectDisbursementMethod<'info> {
    //...
    /// The TGE distribution account to be created/updated
    /// Note: Tokens and mint will be set later during activation
    #[account(
        init_if_needed,
        payer = authority,
        space = TgeDistribution::SPACE,
        seeds = [b"tge_distribution", plot.key().as_ref()],
        bump
    )]
    pub tge_distribution: Account<'info, TgeDistribution>,
    //...
}
```

In the `select_disbursement_method(...)` instruction, if `ImmediateAirdrop` is selected, SOL is transferred to the `escrow_wallet` as gas compensation for the executor. However, when updating, it does not use the SOL already deposited previously but instead records the newly transferred SOL, and SOL already sent to the `escrow_wallet` is not refunded when switching the disbursement method.

Impact: When calling the `select_disbursement_method(...)` instruction to perform updates related to `ImmediateAirdrop`, SOL may become stuck in the `escrow_wallet` account.

Recommendation(s): It is recommended to refund the deposited SOL when switching away from `ImmediateAirdrop`, and to avoid overwriting `tge_distribution.airdrop_gas_reserve` when updating the configuration without changing the `ImmediateAirdrop` method.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/635b6215e472629512e93b8473b77d19da18e48a

7.10 [Medium] `carry_fee` may be lost

File(s): claim_tokens.rs

Description: In the `claim_token(...)` instruction, the `carry_fee` is calculated and sent to the plot or the platform.

```
#[derive(Accounts)]
pub struct ClaimTokens<'info> {
    //...
    /// CHECK: Platform carry fee account authority (from plot or platform config)
    pub platform_carry_account: AccountInfo<'info>,
    //...
```

However, the address that actually receives the `carry_fee` is not being verified.

Impact: The user can provide a TokenAccount they control as the `platform_carry_account`, causing the platform's `carry_fee` to be lost.

Recommendation(s): It is recommended to implement a check for the `platform_carry_account` to ensure that it is a TokenAccount under the platform's control.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/65ee81607de2045305721011dc869d703676aec6

Update from CODESPECT: Fixed but `platform_config.is_active` field is not getting checked here.

7.11 [Medium] `time_lock_approval_threshold` is not configured

File(s): `initialize.rs`

Description: The `time_lock_approval_threshold` field in `platform_config` is used to determine the number of `seeding_admin` approvals required for a proposal to be executed early.

```
pub fn handler(ctx: Context<ApproveTimeLock>) -> Result<bool> {
    //...
    let threshold = platform_config.time_lock_approval_threshold as usize;
    //...
    let threshold_reached = if time_lock.proposer_is_platform_admin {
        //...
    } else {
        // Non-platform admin proposer: need full threshold OR platform admin approval
        msg!("Non-platform proposer: need {} approvals OR platform admin approval", threshold);
        approval_count >= threshold || time_lock.platform_admin_approved
    };
```

However, this field is neither set during the initialization of `platform_config` nor can it be modified by any instruction.

Impact: The `platform_admin` will not be able to control the number of approvers required for early proposal execution. Two `seeding_admins` would be able to execute the proposal immediately.

Recommendation(s): It is recommended to assign a value to this field during the initialization of `platform_config` and allow it to be modified through the timelock.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/d33525e8908e770681735d111bcb740c8c1e7e11

7.12 [Medium] `update_tge_distribution(...)` instruction cannot modify `VestingInfo` in `VestingIntegration` mode

File(s): `update_tge_distribution.rs`

Description: The `update_tge_distribution(...)` instruction does not provide any mechanism to update `VestingInfo` when the distribution is operating in `VestingIntegration` mode. Although the instruction updates other fields of the `tge_distribution` account, the vesting-specific configuration remains unchanged. This prevents administrators from modifying or correcting vesting parameters once the distribution mode is set to vesting.

```
pub fn handler(
    ctx: Context<UpdateTgeDistribution>,
    new_token_symbol: Option<String>,
    new_carry_percentage: Option<u16>,
    new_claim_deadline: Option<Option<u64>>,
) -> Result<()>
```

As a result, vesting-related configuration becomes locked and cannot be corrected or adjusted after initial setup.

Impact: Inability to update `VestingInfo` results in a rigid and potentially faulty vesting configuration. If vesting parameters were set incorrectly during the initial setup, the system offers no way to adjust them, leading to long-term inconsistencies or incorrect release schedules. This restricts protocol flexibility and can adversely affect users relying on accurate vesting logic.

Recommendation(s): Extend the `update_tge_distribution(...)` instruction to support updating `VestingInfo` whenever `tge_distribution.distribution_mode` is `VestingIntegration`.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/18795a9e110792ba69a8acaba543c12e16d97260

Update from CODESPECT: Fixed

7.13 [Low] Active status of `platform_config` is not being checked inside `tge` instructions

File(s): `setup_tge_distribution.rs` , `update_tge_distribution.rs` , `mark_tge_ready.rs` , `select_disbursement_method.rs`

Description: The platform's configuration account is not validated for its `is_active` status before executing `setup_tge_distribution`, `update_tge_distribution`, `mark_tge_ready` and `select_disbursement_method` Instructions, This allows those instructions to run even when the platform is intended to be inactive. Although these operations are admin-only, the absence of this check opens the possibility for misuse if an admin becomes malicious or compromised.

```
//...

/// Platform config for authorization
pub platform_config: Account<'info, PlatformConfig>,

//...
```

Impact: The missing `is_active` validation leads to weakened access control. Administrative actions meant to be disabled during inactive phases can still be performed, allowing unintended state changes or disruptive operations. In the worst case, a malicious admins could alter critical protocol behavior while the platform is supposed to be inactive.

Recommendation(s): Add a mandatory check for the platform's `is_active` flag at the start of all relevant administrative instructions. The instruction should halt with an appropriate error when the platform is inactive, ensuring that inactive mode reliably restricts all sensitive operations.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/b86c97fdbf6191e3cb0dd8506def2202d9f235a5

Update from CODESPECT: Fixed

7.14 [Low] Active status of platform not checked during `growth_cycle::update_metadata(...)` Instruction

File(s): `update_metadata.rs`

Description: In the `update_metadata(...)` instruction, the program does not verify whether the platform is currently Active. Although the instruction correctly loads the `platform_config` using its PDA seeds, it fails to enforce a check on the platform's active/inactive status before allowing metadata updates. This means adminis (or any authorized actors) can update metadata even when the platform is globally inactive.

```
/// Platform configuration for admin checks
#[account(
    seeds = [b"platform_config"],
    bump,
)]
pub platform_config: Account<'info, PlatformConfig>,
```

Impact: Allowing metadata updates while the platform is inactive bypasses intended global state controls and leads to inconsistent administrative behavior. This reduces operational clarity, may cause unexpected state transitions during system downtime, and weakens the platform's configuration integrity.

Recommendation(s): Add a mandatory check for the platform's `is_active` flag at the start of all relevant administrative instructions. The instruction should halt with an appropriate error when the platform is inactive, ensuring that inactive mode reliably restricts all sensitive operations

Status: Fixed

Update from CODESPECT: The `is_active` field in `platform_config` is used for global pause. The instructions such as `claim_tokens`, `execute_airdrop`, `pull_vesting`, and `indicate` also cannot be paused when platform_config.is_active is set to false. So `is_active` field in `platform_config` should be checked in all the above instructions.

Update from Canopy: https://github.com/canopyfi/canopy/commit/f058770cff6c1e438bd7cba68209e9f64a2d10a5

7.15 [Low] Improper access control in the `update(...)` instruction

File(s): `update.rs`

Description: The `update(...)` instruction can modify parameters related to `seeding_admin` and `grove_admin`.

```
pub struct Update<'info> {
    //...
    #[account(
        mut,
        seeds = [
            b"plot",
            plot.growth_cycle.as_ref(),
            name.as_bytes(),
        ],
        bump,
        constraint = plot.created_by == authority.key()
            || admin.is_platform_admin(&platform_config.key())
            @ ErrorCode::Unauthorized,
    )]
    pub plot: Box<Account<'info, Plot>>,
```

However, this instruction allows the plot creator to call it and modify the above parameters, resulting in improper access control.

Impact: This would allow the plot creator to modify parameters related to the counterparty. For example, if the creator is `grove_admin`, after `seeding_admin` calls `approve(...)` to modify `allocation` and `minimum_investment`, `grove_admin` can call update to adjust them again.

Recommendation(s): It is recommended to allow only `platform_admin` to call this instruction

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/1391075e27cc0e3a438f55b2fc164b6f40c3bf9e

Update from CODESPECT: Fixed

7.16 [Low] Incorrect permission control in `update_status(...)` instruction

File(s): `update_status.rs`

Description: During the plot's lifecycle, if a plot is created by the `seedling_admin`, it requires approval from the `grove_admin` and a call to the `accept(...)` instruction to change the status from `Offer` to `PlotStatus::InterestGathering`. Additionally, if the `grove_admin` agrees with the `grove_fees` parameter provided by the `seedling_admin`, they can directly call the `update_status(...)` instruction to perform the status transition.

```
pub fn handler(ctx: Context<UpdateStatus>, new_status: PlotStatus, _name: String) -> Result<()> {
    //...
    // Check if the user is authorized for this specific transition
    let is_authorized = match (&current_status, &new_status) {
        // Platform admins can perform all transitions
        // Grove admins can move from Offered to InterestGathering
        (PlotStatus::Offered, PlotStatus::InterestGathering) => {
            is_platform_admin || is_seedling_admin
        }
    //...
}
```

However, the `update_status(...)` instruction incorrectly grants the permission to transition the status from `Offer` to `PlotStatus::InterestGathering` to `seedling_admin`.

Impact: This allows the `seedling_admin` to unilaterally advance the plot's status to `InterestGathering` without the counterparty (`grove_-admin`)'s consent.

Recommendation(s): Grant the permission to transition the status from `Offer` to `PlotStatus::InterestGathering` to `grove_admin`.

```
    let is_authorized = match (&current_status, &new_status) {
        // Platform admins can perform all transitions
        // Grove admins can move from Offered to InterestGathering
        (PlotStatus::Offered, PlotStatus::InterestGathering) => {
-           is_platform_admin || is_seedling_admin
+           is_platform_admin || is_grove_admin
        }
```

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/pull/76 merged at 57836202e9b74bdbad8af975d01598bcc5f1c017

Update from CODESPECT: Fixed

7.17 [Low] Insufficient `holder_ata` validation

File(s): `execute_airdrop.rs`

Description: The `process_airdrop_batch(...)` instruction is used by the admin to execute airdrops to investors.

```
require!(
    holder_ata_info.owner == &anchor_spl::token::ID,
    ErrorCode::InvalidTokenAccount
);
```

However, the validation for the `holder_ata_info` passed in the `remaining_accounts` is insufficient. It only checks whether `holder_ata_info` belongs to the expected token program, but it does not verify that the account owner is the current holder of the NFT.

Impact: The missing owner check would allow the executor to distribute tokens to a TokenAccount that is not controlled by the NFT holder. And the NFT cannot claim again afterwards.

Recommendation(s): It is recommended to verify that the `holder_ata` owner is the current NFT holder.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/1f1361c5bd86e3f7bb50c6489e0a337ab43c8247

Update from CODESPECT: Fixed

7.18 [Low] Missing `is_ready` validation in `execute_airdrop(...)` and `claim_tokens(...)` instructions

File(s): `claim_tokens.rs` , `execute_airdrop.rs`

Description: The `start_airdrop(...)` and `process_airdrop_batch(...)` instructions do not verify whether the associated `tge_distribution` is in a ready state. Additionally, the `claim_token(...)` instruction also fails to check the `tge_distribution.is_ready` flag before allowing claims. Without these validations, airdrop processing and token claims can occur even when the distribution has not been properly activated where TGE distribution after tokens have been deposited into escrow.

```
pub fn handler(
    ctx: Context<UpdateTgeDistribution>,
    new_token_symbol: Option<String>,
    new_carry_percentage: Option<u16>,
    new_claim_deadline: Option<Option<u64>>,
) -> Result<()> {...}
```

Impact: Skipping the `is_ready` validation allows premature or invalid airdrop execution and token claiming even before the escrow receives tokens for the respective distribution.

Recommendation(s): Add mandatory checks for `tge_distribution.is_ready` in `start_airdrop(...)`, `process_airdrop_batch(...)`, and `claim_token(...)`. Each instruction should halt with an appropriate error if the distribution has not been marked as ready. This ensures that all airdrop and claim operations only occur after the distribution is fully configured.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/939c57a4ff7eff9e6e622e3834c2357fed6e7a9d

Update from CODESPECT: Fixed `tge_distribution.is_ready` is getting checked two times in `execute_airdrop.rs`.

7.19 [Low] Missing `plot` account check allows `seeding_admin` to approve mismatched proposals

File(s): `approve.rs`

Description: The `seedling_admin` can approve proposals in the Pending state, and once the number of approvers reaches the threshold, the proposal can be executed early.

```
pub fn handler(ctx: Context<ApproveTimeLock>) -> Result<bool> {
    //...
    let has_permission = match time_lock.operation_type {
        crate::state::OperationType::UpdateInvestmentDestination => {
            // For plot operations, check if admin has permission on the plot
            let plot = ctx.accounts.plot.as_ref()
                .ok_or(ErrorCode::InvalidInput)?;

            // Check if admin is seedling admin (has the growth_cycle as a target)
            let is_seedling_admin = admin.has_target(&plot.growth_cycle);

            // Platform admin OR seedling admin
            approver_is_platform_admin || is_seedling_admin
        },
        //...
    };
```

However, when performing the authorization check for the `approve(...)` instruction, there is no verification that the `plot` account matches the proposal's `target` field.

Impact: A `seedling_admin` can approve proposals that do not belong to their own seedling, allowing the proposal to reach the threshold early and be executed ahead of schedule.

Recommendation(s): It is recommended to verify that the `plot` account matches the proposal's `target` field.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/c7a29876d1952883dbb6683cbb20ac8e28ae547f Also added ability to set grove fee wallet because that was missing, this also has time lock and should validate that admins have permissions for that grove

7.20 [Low] Premature escrow validation misplaced in setup phase

File(s): `setup_tge_distribution.rs`

Description: The `setup_tge_distribution(...)` instruction currently performs a balance check on the `escrow_token_account`, even though this responsibility is intended for the `mark_tge_distribution(...)` instruction. According to the inline comments in `mark_tge_distribution`,

```
/// Activates a TGE distribution after tokens have been deposited into escrow
/// This is the second phase of TGE setup - called when tokens arrive
#[derive(Accounts)]
pub struct MarkTgeReady<'info>
```

this second-phase instruction is designed to validate the escrow balance after all setup parameters are finalized. Performing the balance check prematurely in `setup_tge_distribution(...)` breaks the intended two-phase flow and causes validation logic to occur at an incorrect stage.

```
//...
    // Validate we have tokens to distribute
    require!(escrow_token_account.amount > 0, ErrorCode::InsufficientTokens);
//...
```

Impact: By checking the escrow balance during the initial setup phase, the system may reject valid setups or force the admin to lock funds earlier than required. This disrupts the designed initialization flow, reduces flexibility, and may cause configuration failures if funds are not yet deposited at the time of setup. It also fragments the validation logic, making the system harder to maintain and increasing the chance of inconsistent behavior between the two phases.

Recommendation(s): Remove the escrow balance check from `setup_tge_distribution(...)` and ensure that validation occurs exclusively within `mark_tge_distribution(...)`, as originally intended. This restores the correct two-phase initialization flow and keeps balance validation aligned with the proper stage of distribution activation.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/db182e73f8595682da9f997f49461a1b51fa84bf

7.21 [Low] The collection system has not been implemented

File(s): initialize.rs, deposit.rs

Description: The Metadata program provides a collection feature that allows a group of NFTs to be verified on-chain, consisting of a Collection NFT that represents the collection and regular NFTs that belong to it.

```
pub struct Initialize<'info> {
    //...
    /// The collection mint (master NFT)
    #[account(
        init,
        payer = authority,
        mint::decimals = 0,
        mint::authority = collection_authority,
        mint::freeze_authority = collection_authority,
        mint::token_program = token_2022_program,
        seeds = [b"collection_mint"],
        bump
    )]
    pub collection_mint: InterfaceAccount<'info, Mint>,
    //...
}
```

In this program, a Collection NFT is created in the `initialize(...)` instruction. However, no Metadata account is created for this mint, nor is the `CollectionDetails` field populated. This is precisely what distinguishes a Collection NFT from a regular NFT. Additionally, when creating metadata for newly deposited NFTs in the `deposit(...)` instruction, they are not linked to the Collection NFT.

Impact: The on-chain collection verification feature provided by the Metadata program is not functional.

Recommendation(s): It is recommended to create metadata for the `collection_mint` and populate the `CollectionDetails` field, and to associate newly minted NFTs with this collection in the `deposit(...)` instruction.

Status: Fixed

Update from Canopy: We ended up moving to the Metaplex Core standard rather than the legacy Token Metadata program. We moved to this standard specifically to store investment data on-chain via the Attributes plugin, enabling DeFi integrations. With that move I think this issue goes away :)

7.22 [Low] The security assumption of threshold-based approval proposals may fail

File(s): add.rs

Description: For seedling proposals, when a sufficient number of `seedling_admins` approve them, the proposal is considered safe and can be executed immediately without waiting.

```
pub fn handler(ctx: Context<Add>, new_admin_authority: Pubkey, target: Pubkey) -> Result<()> {
    //...
    let has_target_access = AuthUtils::has_target_access(caller_admin, &target);
    let is_platform_admin = AuthUtils::is_platform_admin(caller_admin, &platform_config.key());

    require!(
        has_target_access || is_platform_admin,
        ErrorCode::Unauthorized
    );
```

However, the current permission account management model may cause this security assumption to fail, because if an admin has authority over the target account, they can grant the target account's permissions to a new admin. This means `seedling_admins` could assign additional `seedling_admins`.

Impact: This can break the security assumption of threshold-based proposals. A malicious `seedling_admin` could add more `seedling_-admins`, allowing a proposal to reach the approval threshold and execute a malicious proposal immediately.

Recommendation(s): It is recommended to enforce stricter role access control. For example, add role manager to handle the addition of new admins.

Status: Fixed

Update from Canopy: Updated so that new admins cannot approve or propose time lock changes. There will be a notification of the addition of admins, so we can ensure projects can verify added admins before they are eligible to approve time locks
https://github.com/canopyfi/canopy/commit/c492a2a45184fc08a5d1ed64b9c1a69acdb5f15d

7.23 [Low] Unsafe use of `try_deserialize`

File(s): `execute_airdrop.rs`

Description: In the `process_airdrop_batch(...)` instruction, the airdrop distribution-related accounts are read from `remain_accounts`, and then `try_deserialize` is used to deserialize their data. However, the use of `try_deserialize` here is unsafe because the deserialization process only checks whether the account's discriminator matches the expected one, but does not verify whether the account's owner is the expected program.

Impact: A malicious distributor can pass in accounts from a malicious program to distribute the airdrop.

Recommendation(s): It is recommended to verify that the accounts belong to the expected program before performing `try_deserialize` deserialization.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/928720352e7ecae2864eb6e48f67ad87a36f6934

Update from CODESPECT: Fixed but checking for token2022 is sufficient here.

7.24 [Low] `update_tge_distribution(...)` permits unsafe updates after `tge_distribution.is_ready`

File(s): `update_tge_distribution.rs`

Description: The `update_tge_distribution(...)` instruction currently allows modifying the TGE distribution configuration even after it has already been marked as active (`tge_distribution.is_ready = true`). Updating distribution parameters at this stage creates inconsistency between users who have already claimed using the previous token mint and those who will claim after the update. This results in mismatched token allocations and general disruption among depositors.

```
//...
    /// TGE Distribution account (to be updated)
    #[account(
        mut,
        seeds = [b"tge_distribution", plot.key().as_ref()],
        bump,
        constraint = tge_distribution.plot == plot.key()
    )]
    pub tge_distribution: Account<'info, TgeDistribution>,
//...
```

Impact: Allowing updates after activation can cause serious inconsistency in claim outcomes. Users who claimed earlier may receive tokens from one mint or configuration, while later users receive a different version. This can cause confusion, loss of trust, and potential disputes among participants. Functionally, it breaks the expected immutability of the distribution parameters once claims have begun. If abused, this behavior may destabilize the entire TGE distribution flow.

Recommendation(s): Restrict the `update_tge_distribution(...)` instruction from modifying any distribution parameters once `tge_distribution.is_ready` is set to `true`. Enforce a strict validation that prevents updates after activation to ensure consistent token distribution and protect the integrity of all depositor claims.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/784c02c2c5e60d6afc92b3b3efc86b7deae2a81c

Update from CODESPECT: Fixed

7.25 [Info] Inconsistent length validation in `growth_cycle::create(...)` instruction

File(s): `create.rs`

Description: The `growth_cycle::create(...)` instruction performs input length validation for the `name` field, but the enforced limit does not match the length constraint defined in the `GrowthCycle` account structure. While the on-chain account specifies a maximum length of 50 bytes, the instruction incorrectly checking for names up to 100 bytes. Inside `create(...)`:

```
// Validate input lengths
//@audit-info name.len() should be less than 50 not 100
require!(name.len() <= 100, ErrorCode::NameTooLong);
require!(token_warrant.len() <= 200, ErrorCode::NameTooLong);
```

Inside `growth_cycle`:

```
pub struct GrowthCycle {
    #[max_len(50)]
    pub name: String,
    pub seedling: Pubkey,
//...
```

Impact: The inconsistency between the instruction's validation logic and the account's declared maximum size can cause unexpected runtime errors or data truncation when storing values that exceed the actual allowed limit.

Recommendation(s): Align the length validation in the instruction with the `GrowthCycle` account definition by enforcing a maximum of 50 bytes for the `name` field.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/69cdbac12e4880332e04a3b5a5cd4553810cf279

7.26 [Info] Inconsistent maximum length constraints for the field

File(s): `receipt.rs`, `growth_cycle.rs`

Description: After a user performs `deposit(...)`, a `ReceiptMetadata` account is created. The `growth_cycle_name` field of this account comes from `growth_cycle` and is limited to a maximum length of 64.

```
#[account]
#[derive(InitSpace)]
pub struct ReceiptMetadata {
    //...
    #[max_len(64)]
    pub growth_cycle_name: String, // Display name (e.g., "Series A", "Seed Round")
    //...
```

However, in the `GrowthCycle` account, the maximum length of the `name` field is only 50.

```
#[account]
#[derive(InitSpace)]
pub struct GrowthCycle {
    #[max_len(50)]
    pub name: String,
    //...
```

Impact: The `ReceiptMetadata` account is over-allocated by 14 bytes of unused space, resulting in wasted rent.

Recommendation(s): It is recommended to enforce a consistent size constraint on the `name` field.

Status: Fixed

Update from Canopy: Removed ReceiptMetadata as we move to Metaplex Core

7.27 [Info] Inconsistent plot parameter validation

File(s): `accept.rs`, `approve.rs`, `update.rs`

Description: There is inconsistent range validation for relevant parameters throughout the plot lifecycle.

a. The `accept(...)` instruction does not call `validate_fee_structure` when modifying `grove_fees`;
b. The `approve(...)` instruction does not call `validate_amounts` when modifying `new_allocation` and `new_minimum_investment`, thus missing the `max_safe_allocation` check;
c. The `update(...)` instruction lacks `validate_dates`, `validate_fee_structure`, and `validate_amounts` checks when modifying parameters;

Impact: The range checks for plot parameters during updates differ from those at plot creation, allowing parameters to be set out of bounds during updates.

Recommendation(s): It is recommended to enforce the same checks during parameter updates as those applied at plot creation.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/84d8e5b0188487cf1a3d2bc2d3475527f1509e8e

Update from CODESPECT: Fixed

7.28 [Info] Incorrect `groves.len()` check

File(s): `validation.rs`

Description: In the `create(...)` instruction, the `validate_plot_creation(...)` function is called to perform parameter validation.

```rust
/// Comprehensive validation for plot creation parameters
pub fn validate_plot_creation(...) -> Result<()> {
    //...
    require!(groves.len() <= 10, ErrorCode::CollectionsTooMany); // Reasonable limit

    Ok(())
}
```

The `validate_plot_creation` checks that `groves.len()` is less than 10, but the `PlotGroveList` only allocates space for 5 `Pubkey` entries. Therefore, when `groves.len()` exceeds 5, the transaction will fail due to insufficient account space.

```rust
pub struct CreatePlot<'info> {
    //...
    #[account(
        init,
        payer = authority,
        space = PlotGroveList::INITIAL_SPACE,
        seeds = [b"plot_groves", plot.key().as_ref()],
        bump
    )]
    pub grove_list: Box<Account<'info, PlotGroveList>>,
    //...
}
#[account]
pub struct PlotGroveList {
    pub plot: Pubkey,                    // 32 bytes - reference to parent plot
    pub groves: Vec<Pubkey>,             // 4 + (32 * count) bytes
}

impl PlotGroveList {
    // Base size + room for initial groves
    pub const INITIAL_SPACE: usize = 8 + 32 + 4 + (32 * 5); // Space for 5 groves initially
```

Impact: When `groves.len()` exceeds 5, the transaction will fail, but it will not throw the expected error.

Recommendation(s): It is recommended to check that `groves.len()` does not exceed 5, or allocate more space for `PlotGroveList`.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/62580858fbd3f6d20b2605125d476137e58ab55e

Update from CODESPECT: Fixed

7.29 [Info] Incorrect `seedling_admin` check

File(s): `update_investment_destination.rs`, `approve.rs`, `execute.rs`

Description: The `seedling_admin` check is performed in the `update_investment_destination(...)`, `approve(...)`, and `execute(...)` instructions.

```
let is_seedling_admin = admin.has_target(&plot.growth_cycle);
```

However, this check does not verify that the admin account includes the `seedling` account but instead checks the `growth_cycle` account, which does not match the intended implementation.

Impact: To obtain privileges, the platform_admin must additionally configure a `growth_cycle` for the admin account.

Recommendation(s): It is recommended to check the `seedling` account instead of the `growth_cycle` account.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/44a66c6514f509d9a76eb16d05dcdb5a1094c6ec

7.30 [Info] Incorrect reduction of `plot.collected_amount`

File(s): `refund.rs`

Description: In the `deposit(...)` instruction, `plot.collected_amount` records the sum of `allocated_amount`. If the fundraising is not completed, `refund(...)` can be called to process the refund.

```
pub fn refund_watering(ctx: Context<RefundWatering>) -> Result<()> {
    //...
    let investment_amount = ctx.accounts.watering.allotment;
    let platform_fee_amount = ctx.accounts.watering.platform_fees_paid;
    let grove_fee_amount = ctx.accounts.watering.grove_fees_paid;
    let total_refund = investment_amount + platform_fee_amount + grove_fee_amount;
    //...
    plot.collected_amount = plot.collected_amount.saturating_sub(total_refund);
    //...
}
```

However, during the `refund(...)` process, `plot.collected_amount` is reduced by both the `allocated_amount` and the fees.

Impact: `plot.collected_amount` is excessively reduced on each call, causing the remaining amount to be understated. However, since `saturating_sub` is used, this does not lead to a DoS.

Recommendation(s): It is recommended that in `refund(...)`, `plot.collected_amount` should only be reduced by `investment_amount` and not by the fees.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/fd3ee7c1bf3c35976a2b250fb00889f0cf0cbedc

7.31 [Info] Lacking an on-chain authorization check allows `growth_cycle` creation by anyone

File(s): `create.rs`

Description: For the creation of `growth_cycle` the comments mention that authorization is off-chain and only Platform admins and Seedling admins are allowed to call the `create(...)` instruction.

```
pub fn handler(...) -> Result<()> {
    //...
    // NOTE: Admin authorization check is handled at the application layer.
    // Only Platform admins and Seedling admins should be allowed to call this instruction.
    // The authority signer ensures proper authentication, and additional role checks
    // can be enforced in the frontend/backend before submitting transactions.
```

However, the lack of on-chain authorization allows anyone to directly send a transaction to create a `growth_cycle` account.

Impact: An unauthorized `growth_cycle` account may be created under a `Seedling`.

Recommendation(s): It is recommended to add on-chain authorization checks to the `create(...)` instruction.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/c3329986c880a1da3cbd880fd7f3344d4c54a20d

7.32 [Info] Missing `mut` constraint on `platform_collection`

File(s): `deposit.rs`

Description: In the `deposit(...)` instruction, the `platform_collection` account is passed in. This account is expected to be modified in the instruction, specifically the `total_minted` and `updated_at` fields.

```
{
    let platform_collection = &mut ctx.accounts.platform_collection;
    platform_collection.total_minted += 1;
    platform_collection.updated_at = current_time;
}
```

However, in `ctx`, this account is not marked as `mut`.

Impact: Not marking the account as `mut` but modifying its data will cause the changes not to be persisted on-chain. The instruction will execute successfully, but the `total_minted` and `updated_at` fields will not be updated.

Recommendation(s): Mark `platform_collection` as `mut`.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/pull/82 merged at c2b9023f0acf1d96d9237f917bf6b714bcee3a2e

Update from CODESPECT: Fixed

7.33 [Info] Redundant discriminator space allocation in `platform::initialize(...)` instruction for `platform_collection` account

File(s): `initialize .rs`

Description: In the `platform::initialize(...)` instruction, the `PlatformCollection` account is initialized with extra space allocated for the account discriminator. The `space` parameter explicitly adds an additional 8 bytes, while the `PlatformCollection::SPACE` constant already includes these 8 bytes as part of its size calculation. This results in redundant space allocation when creating the account, leading to an over-provisioned account size.

```
/// Platform collection account for NFT receipts
   #[account(
       init,
       payer = authority,
       space = 8 + PlatformCollection::SPACE,
       seeds = [b"platform_collection"],
       bump
   )]
   pub platform_collection: Account<'info, PlatformCollection>,
```

Impact: This issue has no functional or security impact on the protocol. It only results in a small and unnecessary allocation of 8 extra bytes, slightly increasing account size and rent requirements without providing any benefit.

Recommendation(s): Remove the additional 8-byte discriminator from the `space` parameter or adjust the `PlatformCollection::SPACE` constant to exclude it, ensuring the discriminator is counted exactly once.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/be10e2b1f721723511dc68cfe6ba0749ee3fdda6

7.34 [Info] Some fields in `vesting_info` may be inaccurate

File(s): `pull_vesting.rs`

Description: In the `pull_vesting(...)` instruction, locked tokens are transferred from Streamflow to the plot's `escrow_wallet`.The instruction will modify the fields `total_withdrawn`, `last_withdrawal_at`, and `withdrawal_count`.

```
withdraw_from_streamflow(
    &ctx.accounts.vesting_program.to_account_info(),
    &ctx.accounts.vesting_contract.to_account_info(),
    &ctx.accounts.escrow_wallet.to_account_info(),
    &ctx.accounts.escrow_token_account.to_account_info(),
    &ctx.accounts.vesting_escrow_tokens.to_account_info(),
    &ctx.accounts.vesting_treasury.to_account_info(),
    &ctx.accounts.vesting_treasury_tokens.to_account_info(),
    &ctx.accounts.token_mint.to_account_info(),
    &ctx.accounts.token_program.to_account_info(),
    &ctx.accounts.authority.to_account_info(),
)?;
```

In the CPI call, the recipient is `escrow_wallet`, but the authority is `authority`(not recipient). This means that the `automatic_withdrawal` in `vesting_contract` is true, so anyone can call the `withdraw` instruction to release already unlocked tokens directly to the receipt.

Impact: Anyone can bypass the `pull_vesting(...)` instruction and directly call the `withdraw(...)` function in Streamflow to release tokens to the `escrow_wallet`. Therefore, the fields `total_withdrawn`, `last_withdrawal_at`, and `withdrawal_count` updated in `pull_-vesting(...)` may be inaccurate.

Recommendation(s): It is recommended to ensure `automatic_withdrawal` is set to false, which requires that the authority during withdrawal must be the receiver (`escrow_wallet`).

Status: Fixed

Update from Canopy: Looked into it and since we won't be able to control the streamflow vesting setup (yet, might offer that as a part of the platform once we start marketing to founders seeking money). So for this issue seems like the most important thing that we'd lose integrity on would be the total_withdrawn. So when pull vesting happens we check streamflows records for that and use them as source of truth https://github.com/canopyfi/canopy/commit/914e6c88acc01b174e8ecc5e70dbf04cf7823b1b

7.35 [Info] The validation of `grove_fees_destination` is not implemented

File(s): `conclude.rs`

Description: In the `conclude(...)` instruction, the `grove_fees_destination` account is passed to receive the grove fee.

```
pub struct ConcludePlot<'info> {
    //...
    /// Grove fees destination (first grove's fee wallet for simplicity)
    /// CHECK: Passed by caller, should be validated by business logic
    #[account(mut)]
    pub grove_fees_destination: Box<Account<'info, TokenAccount>>,
```

This account is expected to be checked and must be the first grove's fee wallet, but this check is not actually implemented.

Impact: Grove fees may be sent to an unintended address.

Recommendation(s): It is recommended to implement a check for the `grove_fees_destination` account.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/950e8e187f470ab2c01bf280fd46e2aa4ffa5f56

Update from CODESPECT: Fixed

7.36 [Info] The calculation of `current_balance` uses outdated data

File(s): `deposit.rs`

Description: In the `deposit(...)` instruction, after the user pays the investment amount and fees, the `current_balance` is calculated using the balances of the current three TokenAccounts.

```
// Update total value locked in escrow wallet
let current_balance = ctx.accounts.investment_escrow.amount
    + ctx.accounts.platform_fees_escrow.amount
    + ctx.accounts.grove_fees_escrow.amount;

escrow_wallet.total_value_locked = current_balance;
escrow_wallet.updated_at = Clock::get()?.unix_timestamp as u64;
```

However, this calculation occurs while the three TokenAccounts are being used as token recipient accounts in the CPI transfers. And the three accounts are not reloaded to update the memory data, causing `current_balance` to be calculated using outdated balances.

Impact: The `current_balance` uses outdated balances and is assigned to `escrow_wallet.total_value_locked`, which may cause off-chain systems to retrieve incorrect data.

Recommendation(s): It is recommended to reload the three TokenAccount accounts before calculating `current_balance`.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/pull/79 merged at c31373994f5ae7d1e7afa591624177778df6c577

Update from Canopy: https://github.com/canopyfi/canopy/commit/c31373994f5ae7d1e7afa591624177778df6c577

Update from CODESPECT: Fixed

7.37 [Info] The permission control comment does not match the implementation

File(s): execute_airdrop.rs

Description: In the setup_tge_distribution , start_airdrop_handler and update_tge_distribution instructions, the permission control comment states:

```
//...
       // Authorization check - must be platform admin or seedling admin
    let is_authorized = AuthUtils::is_platform_admin(admin, &platform_config.key()) ||
                        AuthUtils::is_growth_cycle_admin(admin, &plot.growth_cycle);
//...
```

However, the implementation checks for a growth_cycle admin instead of seedling admin

Impact: This results in weakened access control within the system, allowing unprivileged accounts to access functionality that should be restricted.

Recommendation(s): Implement what mentioned in the comment or modify the comment if this is the intended functionality

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/9657d0055c0b318fc68dbfc70ca5143287bb6c25

Update from CODESPECT: Fixed

7.38 [Info] `emergency_pause(...)` does not update platform active status

File(s): emergency_pause.rs

Description: The emergency_pause(...) instruction is intended to pause all platform operations during emergency situations. However, the instruction does not update the active/inactive status in the platform_config account. Instead, it only updates the global_rate_-limiter account, which is not referenced by any instruction when checking whether the platform is active. As a result, even after invoking the emergency pause, normal platform operations continue to function as if no pause was applied.

Impact: The emergency pause mechanism is ineffective, as it does not actually stop or restrict platform operations. This undermines the purpose of having an emergency control and may prevent timely mitigation during critical incidents.

Recommendation(s): Update the emergency_pause(...) instruction to explicitly modify the platform's active status within the platform_-config account.

Status: Fixed

Update from Canopy: Removed emergency pause since it was partially implemented and needs to be rethought

7.39 [Info] `updated_at` is not updated in the `process_airdrop_batch(...)` instruction

File(s): update_tge_distribution.rs

Description: The tge_distribution.updated_at field is updated with every TGE interaction.

```
// Check if airdrop is complete
if is_complete {
    //...
    tge_distribution.total_claimed = SafeMath::add(
        tge_distribution.total_claimed,
        airdrop_batch.total_distributed,
    )?;
    tge_distribution.total_carry_collected = SafeMath::add(
        tge_distribution.total_carry_collected,
        airdrop_batch.total_carry_collected,
    )?;
```

However, in process_airdrop_batch(...), parameters in tge_distribution are modified, but updated_at is not updated.

Impact: The missing update of updated_at can cause the last TGE interaction time to be inconsistent with the records.

Recommendation(s): Update tge_distribution.updated_at in the process_airdrop_batch(...) instruction.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/fa506c30bc1d26dcf88d3be3a77f684ad09ab699

7.40 [info] Unnecessary 64-byte validation for operation_data in `time_lock::execute(...)` instruction for `OperationType::UpdateInvestmentDestination`

File(s): `execute.rs`

Description: In the `UpdateInvestmentDestination` operation, the instruction enforces that `operation_data` must be exactly 64 bytes long. However, this requirement is unnecessary because the instruction does not actually use the first 32-bytes of the `operation_data` to derive or decode the previous destination address. Instead, the old destination is obtained directly from `plot.investment_destination_wallet`, making the strict length check redundant and unnecessary.

```
require!(operation_data.len() == 64, ErrorCode::InvalidInput);

  //...
  //...

 let old_destination = plot.investment_destination_wallet;
```

Impact: he incorrect 64-byte requirement results in an unnecessary waste of 32 bytes of input data, providing no functional or security benefit. This adds minor overhead and contributes to inefficient instruction design without affecting system behavior.

Recommendation(s): Remove or relax the 64-byte requirement for `operation_data` when handling `UpdateInvestmentDestination`, as it is not needed for the operation's logic.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/d9db31ab94ce21509ce89fdca30b6dd2f347549f

Update from CODESPECT: In `propose`, `OperationType::UpdateInvestmentDestination` does not allow `new_destination` to be default. But in `execute`, when `new_destination` is default, it is treated as None. This is an inconsistency.

Update from Canopy: https://github.com/canopyfi/canopy/commit/d00269a5489d69ff85471a5b810b85b3e1bd1172

7.41 [Best Practice] Incorrect clearing of `uri` causes logging errors

File(s): `update_metadata.rs`

Description: The `update_metadata(...)` instruction is used to update the `url` of a `growth_cycle`.

```
if uri.is_empty() {
    // Empty string means clear the URI
    growth_cycle.metadata_uri = Some(String::new());
} else {
```

However, when clearing the `url`, `growth_cycle.metadata_uri` is incorrectly set to an empty `String` instead of `None`.

```
match (&old_metadata_uri, &new_metadata_uri) {
    (Some(old), Some(new)) if old != new => {
        msg!("Updated growth cycle '{}' metadata URI from '{}' to '{}'",
             growth_cycle.name, old, new);
    }
    (Some(old), None) => {
        msg!("Removed growth cycle '{}' metadata URI (was: '{}')",
             growth_cycle.name, old);
    }
```

This causes subsequent log printing to enter the incorrect `Some(), Some()` branch instead of the `Some(), None` branch.

Impact: The program may log incorrect information, leading to ambiguity.

Recommendation(s): It is recommended to set `growth_cycle.metadata_uri` to `None` when clearing the `url`.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/597c4dc08695c5a7fc4a119eea16c5394c2dba91

7.42 [Best Practice] Missing validation for default and duplicate values in `platform::update_fees_wallet` and `grove::create`

File(s): `update_fees_wallet.rs`, `create.rs`

Description: In the `platform::update_fees_wallet` instruction, there is no validation to ensure that the `new_fee_wallet` is different from the previously configured fee wallet or that it is not set to the default `Pubkey`. This allows redundant or invalid updates that do not meaningfully change platform configuration. Similarly, in the `grove::create` instruction, several critical fields such as `name`, `fee_wallet`, `onboarding_fee`, and `platform_fees` are not validated against default or empty values. The absence of these checks permits the creation of grove accounts with null or unintended configurations.

Impact: These missing validations do not introduce a direct security vulnerability but can lead to misconfigured platform and grove states.

Recommendation(s): Add explicit validation to ensure that `new_fee_wallet` is neither the default public key nor identical to the existing fee wallet before applying updates. Additionally, enforce checks in `grove::create` to prevent default or empty values for critical fields, ensuring all newly created groves start in a valid and intentional configuration state.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/c52b0975b43ad028edc585235f3e0d6c460cd94e

7.43 [Best Practice] Perform the `operation_data` length check earlier

File(s): `execute.rs`

Description: In the `time_lock` feature, the length of `operation_data` is only checked when the proposal is executed.

```
match time_lock.operation_type {
    crate::state::OperationType::UpdateFeesWallet => {
        //...
        require!(operation_data.len() == 32, ErrorCode::InvalidInput);
        //...
    crate::state::OperationType::UpdateCollectionRoyalty => {
        //...
        require!(operation_data.len() == 32, ErrorCode::InvalidInput);
        //...
    crate::state::OperationType::UpdateInvestmentDestination => {
        //...
        require!(operation_data.len() == 64, ErrorCode::InvalidInput);
        //...
}
```

In fact, this check could be performed earlier, when the proposal is submitted, to prevent a `TimeLock` account with an incorrect `operation_data` length from being created.

Impact: A proposal may fail to execute after it becomes executable due to an `operation_data` length mismatch.

Recommendation(s): It is recommended to move the `operation_data` length check to the `propose(...)` instruction.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/d36d5d0c9aad8aad9cb82abc18d6aff1a5194b80

7.44 [Best Practice] Redundant code

File(s): `validation.rs`

Description: In `validation.rs`, `SafeMath` and its implementation are redundantly defined, as they have already been implemented in `safe_math.rs`.

```
pub struct SafeMath;

impl SafeMath {
    //...
}
```

Impact: Redundant code increases readability difficulty, enlarges the program size, and raises deployment costs.

Recommendation(s): It is recommended to remove the redundant `SafeMath` definition.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/0c6a28273e5c6b553afd6ed62bfa915675759e98

7.45 [Best Practice] Redundant updates of `updated_by` and `updated_at`

File(s): `remove.rs`, `initialize.rs`

Description: The `remove(...)` instruction is used to remove a target from the admin while also updating `updated_by` and `updated_at`.

```rust
pub fn handler(ctx: Context<Remove>, target: Pubkey) -> Result<()> {
    //...
    admin.remove_target(&target, caller_admin.key());
    admin.updated_at = Clock::get()?.unix_timestamp as u64;
    admin.updated_by = caller_admin.key();

    Ok(())
}
```

In the `remove_target(...)` function, `updated_by` and `updated_at` are already updated, but after calling `remove_target(...)`, `updated_by` and `updated_at` are updated again redundantly.

```rust
pub fn handler(...) -> Result<()> {
    //...
    platform_admin.updated_at = current_time;
    platform_admin.updated_by = authority.key();
    //...
    platform_admin.add_target(platform_config.key(), authority.key());
    //...
}
```

The same issue also occurs in the `initialize(...)` instruction. Calling the `add_target(...)` function updates `updated_by` and `updated_-at`, but they have already been updated beforehand.

Impact: Redundant updates of `updated_by` and `updated_at` result in wasted compute units.

Recommendation(s): It is recommended not to redundantly update `updated_by` and `updated_at` before and after calling the `add_-target(...)` and `remove_target(...)` functions.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/c1246e17e812ca471a03218ae7637fd17b5f37b7

7.46 [Best Practice] The `time_lock` status is not updated when it ready

File(s): `approve.rs`

Description: When the number of proposal approvers reaches the threshold, the proposal can be executed early, and `early_execution_-approved` is set to true.

```rust
pub fn handler(ctx: Context<ApproveTimeLock>) -> Result<bool> {
    //...
    if threshold_reached {
        // Set ready_at to current time to allow immediate execution
        let current_time = Clock::get()?.unix_timestamp;
        time_lock.ready_at = current_time;
        time_lock.early_execution_approved = true;
        //...
    }
```

The `time_lock` account defines a `TimeLockStatus::Ready` state, but when a proposal can be executed early, the `time_lock` account is not set to this state.

Impact: `TimeLockStatus::Ready` is defined but not used, which may cause ambiguity when querying the `time_lock` account data on-chain.

Recommendation(s): It is recommended to either remove the `TimeLockStatus::Ready` state or set the `time_lock` account to `TimeLockStatus::Ready` when a proposal reaches the threshold, and account for the `TimeLockStatus::Ready` state in `can_execute(...)`.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/cd1be34c258497ed9a03fe5b179837e0263c9aba

7.47 [Best Practice] Unreachable state

File(s): `grove.rs`, `growth_cycle.rs`, `seedling.rs`

Description: In the program, some accounts have enum-defined states, but certain defined states can never be reached. * `Grove` has no method to reach the `Inactive` state. * `GrowthCycle` has no method to reach the `Raising`, `Closed`, or `Completed` states. * `Seedling` does not use the `Approved` state and has no method to reach the `Inactive` state.

Impact: These unreachable states cause code redundancy and make reading and maintenance inconvenient.

Recommendation(s): It is recommended to either remove the unreachable states or add state transition instructions for them.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/01d520846f1f231e3975f7d08fa28234d4a20bad Removed some of the states, added instructions for GrowthCycle::Complete so that can be updated.

8 Fix Review Issues

8.1 [Medium] Incorrect `fee_wallet` check

File(s): `pay_onboarding_fee.rs`

Description: When the platform admin adds a Grove, if an `onboarding_fee` is set, the Grove needs to call the `pay_onboarding_fee(...)` instruction to pay the `onboarding_fee`.

```
pub fn handler(ctx: Context<PayOnboardingFee>, grove_name: String) -> Result<()> {
    //...
    // Verify the fee wallet matches the Grove's fee wallet
    require!(
        fee_wallet.key() == grove.fee_wallet,
        crate::error::ErrorCode::InvalidFeeWallet
    );
```

However, the `fee_wallet` check here is incorrect. It restricts the wallet to `grove.fee_wallet`, which effectively means they are paying fees to themselves.

Impact: The platform will lose the `onboarding_fee`.

Recommendation(s): The fee check should use the platform's fee address instead of the Grove's fee address.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/96af3bad0ad01446c4d17ffc4f538a0b6c7e8d13

8.2 [Medium] The `refund(...)` instruction cannot handle cases where the NFT has already been sold

File(s): `refund.rs`

Description: During the `Collecting` phase, users can `deposit` USDC to obtain NFTs that may be received in the future `TGE`. During this period, users can choose to sell the NFTs. However, after the `Collecting` phase, a `plot` may be canceled if it did not receive enough funds. In this case, users need to call the `refund(...)` instruction to get a refund.

```
/// Watering to refund
#[account(
    mut,
    seeds = [
        b"watering",
        plot.key().as_ref(),
        authority.key().as_ref(),
    ],
    bump,
    constraint = watering.plot == plot.key() @ ErrorCode::WateringPlotMismatch,
    constraint = watering.member == authority.key() @ ErrorCode::UnauthorizedMember,
    constraint = watering.status == WateringStatus::Deposited @ ErrorCode::WateringNotDeposited,
    constraint = watering.nft_receipt.is_some() @ ErrorCode::NoNftReceipt,
)]
pub watering: Box<Account<'info, Watering>>,

/// The Metaplex Core NFT asset to burn
#[account(
    mut,
    constraint = receipt_asset.key() == watering.nft_receipt.unwrap() @ ErrorCode::InvalidNftReceipt,
    constraint = receipt_asset.owner == authority.key() @ ErrorCode::InvalidTokenAccountOwner,
)]
pub receipt_asset: Account<'info, BaseAssetV1>,
```

However, the `refund(...)` instruction requires the caller to be the original `depositor` and to hold the NFT. If the user has already sold their NFT, the current NFT holder cannot call the refund to receive USDC.

Impact: The `refund(...)` instruction cannot be executed when the NFT has been sold, and the NFT buyer will lose their USDC.

Recommendation(s): It is recommended that the `refund(...)` instruction be called by the current NFT holder and that the refund be issued to the current NFT holder.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/0e70b79a0bd14ff11a97538f4e4c40f5963dcda6

8.3 [Medium] carry_fee may be lost

File(s): execute_airdrop.rs

Description: In the execute_airdrop(...) instruction, the target address for carry_fee is not checked.

```
/// Platform carry fee token account
#[account(
    mut,
    constraint = platform_carry_token_account.mint == token_mint.key()
)]
pub platform_carry_token_account: Box<Account<'info, TokenAccount>>,

/// Grove carry fee token account
#[account(
    mut,
    constraint = grove_carry_token_account.mint == token_mint.key()
)]
pub grove_carry_token_account: Box<Account<'info, TokenAccount>>,
```

The caller can send platform fees to any specified TokenAccount.

Impact: The carry_fee could be stolen.

Recommendation(s): It is recommended to implement a check for the platform_carry_account and grove_carry_token_account.

Status: Fixed

Update from CODESPECT: dup of 7.10

Update from Canopy: https://github.com/canopyfi/canopy/commit/1577f71441d659b472802831517e839c513452c5

Update from CODESPECT: It seems to be inconsistent with the implementation in claim_tokens. In that instruction, grove_carry_fee is sent to the Grove in the first grove_list. Should this behavior be kept consistent here as well?

Update from Canopy: https://github.com/canopyfi/canopy/commit/a2172eab707ab6f83c70ef108523d7f04b78a4d0

8.4 [Low] Expired and unclaimed tokens cannot be withdrawn

File(s): tge_distribution.rs

Description: The seedling_admin can set a claim deadline for user tokens, and any claims made before the deadline will be rejected.

```
/// Check if distribution is still valid (not expired)
/// Returns Result to handle Clock errors properly
pub fn is_valid(&self) -> Result<bool> {
    if !self.is_active {
        return Ok(false);
    }

    if let Some(deadline) = self.claim_deadline {
        let current_time = Clock::get()?.unix_timestamp as u64;
        Ok(current_time <= deadline)
    } else {
        Ok(true)
    }
}
```

However, if there are unclaimed tokens, this portion of tokens will be stuck and cannot be withdrawn by anyone.

Impact: Expired and unclaimed tokens will be stuck.

Recommendation(s): It is recommended to add an instruction that allows withdrawing undistributed tokens when the TGE expires.

Status: Fixed

Update from Canopy: We should not do this currently, as it will need some more thought going forward. I don't want an admin to be able to withdraw the claim to get the money, whether it is platform or seedling admin they shouldn't be able to remove everyone's wins from them. I'm going to remove the deadline so if its a claim they have as long as they want to claim it doesn't matter to us and only would open another attack vector to be able to end a claim and allow someone to pull those tokens

Update from Canopy: https://github.com/canopyfi/canopy/commit/67d4493194a848a558e6b56aea705ce06bb34192

8.5 [Low] Improper authorization in the `mark_tge_ready(...)` instruction

File(s): `mark_tge_ready`

Description: The `mark_tge_ready(...)` instruction allows a `grove_admin` to call it.

```
let is_platform_admin = AuthUtils::is_platform_admin(admin, &platform_config.key());
let is_grove_admin = admin.targets.contains(&grove.key());
let is_seedling_admin = admin.targets.contains(&seedling.key());

require!(
    is_platform_admin || is_grove_admin || is_seedling_admin,
    ErrorCode::Unauthorized
);
```

However, this instruction sets some parameters such as `token_mint` that are normally set by the `update_tge_distribution(...)` instruction and should only be allowed to be set by the `seedling_admin`.

Impact: A `grove_admin` could call the `mark_tge_ready(...)` instruction to start the TGE before the `seedling_admin` calls `update_tge_distribution(...)`, potentially modifying some parameters set by the `seedling_admin`.

Recommendation(s): It is recommended not to allow a `grove_admin` to call `mark_tge_ready(...)`.

Status: Fixed

Update from Canopy: Want grove admins to be able to do this as well. I added check to ensure that TGE is properly configured before mark_tge_ready() https://github.com/canopyfi/canopy/commit/c4b19349aef89c68477324a479cb2ed7474ad1a6

Update from CODESPECT: If that is the case, we suggest removing this instruction's ability to reconfigure `token_mint` and `total_tokens`. The `escrow_token_account` may not be the expected account (the system does not enforce it to be an ATA), which could result in an incorrect `total_tokens` amount and `token_mint`.

Update from Canopy: https://github.com/canopyfi/canopy/commit/f264403f43335205b57cf047870f672266b97954 Simplifies `mark_tge_ready` by removing the ability to change token configuration at activation time - those parameters must now be configured beforehand via `update_tge_distribution`

8.6 [Low] The `airdrop_gas_reserve` may not be refunded to the depositor

File(s): `select_disbursement_method.rs`

Description: When changing `distribution_mode` from `DistributionMode::ImmediateAirdrop` to another distribution method, a refund will be issued for the already transferred `airdrop_gas_reserve`.

```
pub fn handler(...) -> Result<()> {
    //...
    if switching_away_from_airdrop && previous_gas_reserve > 0 {
        // Refund the gas reserve back to the authority
        let escrow_wallet_info = escrow_wallet.to_account_info();
        let authority_info = ctx.accounts.authority.to_account_info();

        **escrow_wallet_info.try_borrow_mut_lamports()? = escrow_wallet_info
            .lamports()
            .checked_sub(previous_gas_reserve)
            .ok_or(ErrorCode::ArithmeticOverflow)?;
        **authority_info.try_borrow_mut_lamports()? = authority_info
            .lamports()
            .checked_add(previous_gas_reserve)
            .ok_or(ErrorCode::ArithmeticOverflow)?;

        msg!(
            "Refunded {} lamports gas reserve to authority (switching from ImmediateAirdrop)",
            previous_gas_reserve
        );
    }
```

The refund target address is the authority of the current instruction call. however, this address may not be the one that originally transferred the gas.

Impact: The refund may not be returned to the address that originally made the deposit.

Recommendation(s): It is recommended to transfer the refund to the platform admin for distribution.

Status: Fixed

Update from Canopy: Stored the depositor's information on the deposit, and validated that the funds went back to them. Ignored anyone who managed to add more gas since they are on the same team and have been added as admins so just fall back to first, we are not responsible for that https://github.com/canopyfi/canopy/commit/a93dda9b83df879e61a3bb64cdc7441f1a8605ee

8.7 [Low] The `conclude(...)` instruction can be executed even when `investment_destination` is not set

File(s): `conclude.rs`

Description: When a `plot` is not created by a `grove`, the `investment_destination` field will not be set and will be configured later through `time_lock`.

```
// Verify the investment destination token account owner matches the plot's setting
if let Some(destination_wallet) = plot.investment_destination_wallet {
    require!(
        ctx.accounts.investment_destination.owner == destination_wallet,
        ErrorCode::InvalidInvestmentDestination
    );
}
```

However, the `investment_destination` field is the only field checked by the `conclude` instruction to determine where the funding should be sent. If this field is not set, the check will be skipped, and funds may be sent to an unintended address.

Impact: The funds of a Seedling could be sent to any arbitrary address.

Recommendation(s): It is recommended to prohibit calling the `conclude(...)` instruction when `investment_destination` is not set, rather than skipping the check.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/9c9c18bb4522ae1c0c068aeef77ab5db10fd9674

8.8 [Low] The `remove(...)` instruction has insecure access control

File(s): `remove.rs`

Description: In the `remove(...)` instruction, an admin holding the `target` address permission can remove the permissions of any other admin who also holds the `target` address permission.

```
pub fn handler(ctx: Context<Remove>, target: Pubkey) -> Result<()> {
    //...
    if !caller_admin.has_target(&target) && !caller_admin.is_platform_admin(&platform_config.key())
    {
        return Err(error!(ErrorCode::Unauthorized));
    }
    //...
}
```

Impact: This permission design pattern is unsafe. If there is a malicious admin in the system holding the `target` permission, they can remove other admins' control over the `target` permission.

Recommendation(s): It is recommended that, except for the platform admin, other admins are only allowed to use `remove(...)` to remove their own `target` permission and not the `target` permission of other admins.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/698b2f7292584a8aaa138885da5b95ad1d3033ac only platform admin can remove, don't need people removing their own

Update from CODESPECT: Sir, the implementation of this commit appears to be the opposite of your explanation. We suggest retaining the platform admin's removal permission for security control, and also allowing users to remove themselves.

Update from Canopy: https://github.com/canopyfi/canopy/commit/2888e691c24f5c341116d5be4f489b2ecf4d06f3

8.9 [Low] Unsafe account creation

File(s): `initialize.rs`

Description: In the `initialize(...)` instruction, `Admin` accounts will be created for the `founding_admins`.

```
anchor_lang::solana_program::program::invoke_signed(
    &anchor_lang::solana_program::system_instruction::create_account(
        &authority_key,
        &admin_account_info.key(),
        admin_lamports,
        admin_space as u64,
        ctx.program_id,
    ),
    &[
        ctx.accounts.authority.to_account_info(),
        admin_account_info.clone(),
        ctx.accounts.system_program.to_account_info(),
    ],
    &[signer_seeds],
)?;
```

However, the instruction directly calls `system_instruction::create_account` to create the accounts. This requires the accounts being created to have a rent of 0. Otherwise, the accounts will be considered already in use and the creation will fail. An attacker could preemptively send 1 lamport to the account to be created, causing the `initialize(...)` instruction to fail due to a CPI call failure.

Impact: An attacker could potentially DOS the `initialize(...)` instruction by front-running and sending 1 lamport to the account to be created.

Recommendation(s): When checking whether an account needs to be created, verify not only whether lamports exist but also whether the account data size matches the expected size. When creating the account, call `create_account` only if lamports are 0; if not, top up the missing rent and then use `system_instruction::allocate` and `system_instruction::assign` to create the account.

Status: Fixed

Update from Canopy: I have a PR open for this, but it seems like it's not necessary. When I deploy this program I immediately run the initialize for the platform for the only time. Since that closes the opportunity for this DOS because it should only be run once https://github.com/canopyfi/canopy/pull/135

8.10 [Low] Updating the `royalty_wallet` does not take effect

File(s): `execute.rs`

Description: The platform admin can update the `royalty_wallet` in the `platform_collection` through a time lock.

```
crate::state::OperationType::UpdateCollectionRoyalty => {
    //...
    let old_royalty_wallet = platform_collection.royalty_wallet;

    // Update the royalty wallet
    platform_collection.royalty_wallet = new_royalty_wallet;
    platform_collection.updated_at = TimeUtils::get_current_timestamp()?;
    //...
}
```

However, this update may not take effect because the royalty distribution address is actually recorded in the royalty plugin of `BaseCollectionV1`, and the address stored there is not updated.

Impact: Replacing the `royalty_wallet` may not take effect, and royalties will still be sent to the old address.

Recommendation(s): When updating the `platform_collection.royalty_wallet` field, also update the address in the `BaseCollectionV1` royalty plugin.

Status: Fixed

Update from Canopy: This instruction just seems dumb, we are having it set to the multisig, if we ever need to change it we'll add it. Right now it adds unnecessary complexity and another thing to monitor. ebd2ce0930e1379a82eefb2ad134d90270f77f7d

8.11 [Info] The groves in `grove_list` may not be in an Active state

File(s): `create.rs`

Description: Currently, in the `grove_list` check, most validations directly check whether the admin exists in the `grove_list` for the target, without passing in the actual Grove account for verification.

```
#[inline(always)]
pub fn is_grove_admin_for_list(admin: &Admin, groves: &[Pubkey]) -> bool {
    groves.iter().any(|grove| admin.has_target(grove))
}
```

Therefore, it cannot be determined whether the Grove account is in an Active state or has already been frozen.

Impact: A Grove account that is not in an Active state can still interact.

Recommendation(s): It is recommended to check the actual Grove account state during grove verification.

Status: Fixed

Update from Canopy: https://github.com/canopyfi/canopy/commit/f2728dc1db10c0cc4706ae4d66163eb8c421813d

Update from CODESPECT: In the current modification, if one of the groves is frozen, the call will directly revert instead of continuing the checks. I recommend continuing the loop when a grove is frozen or not in the ACTIVE state, rather than reverting. utils/auth.rsL65

Update from CODESPECT: mark_tge_ready.rsL54 The `mark_tge_ready` instruction does not seem to perform an update check on the grove.

Update from Canopy: https://github.com/canopyfi/canopy/commit/b9a379fe4dad0b11b6165a6a52b068bc0a2ddae5

8.12 [Info] `select_disbursement_method(...)` does not restrict the plot state

File(s): `select_disbursement_method.rs`

Description: When creating a TGE in the `select_disbursement_method(...)` instruction, the plot state is not restricted.

```rust
/// The plot to configure TGE disbursement for
/// Note: Can be configured at any time, no status requirement
#[account(
    has_one = growth_cycle @ ErrorCode::InvalidGrowthCycle,
)]
pub plot: Account<'info, Plot>,
```

A TGE should be created only after all water funds have been collected, at which point the number of recipients and `plot.collected_-amount` will be finalized.

Impact: Creating the TGE too early can cause `tge_distribution.total_verified_deposits` to not equal the final `plot.collected_amount`, which will cause token claim checks to fail.

Recommendation(s): It is recommended to allow TGE creation only in the Collected or Completed phase.

Status: Fixed

Update from Canopy: Needs to be `completed` because that's when the company gets the money from the raise. which should 100
https://github.com/canopyfi/canopy/commit/efa9d1e867e82c3a22f8c66b71684c7f3a5386cc

9 Evaluation of Provided Documentation

The **Canopy** documentation was provided in the form of a README file and NatSpec comments:

- **NatSpec:** The in-code NatSpec comments were generally sufficient and very helpful in explaining specific flows and code branches. In several cases, they also clarified assumptions underlying the implementation, providing context for why certain approaches were taken and the intended behavior of the system.
- **README:** The provided README offered a solid overview of the protocol's functionality. However, in some instances, there were discrepancies between the README and the actual code.

Overall, the documentation was adequate and sufficient for the scope of this audit. Additionally, the Canopy team remained consistently available and responsive, promptly addressing all questions and concerns raised by **CODESPECT** throughout the audit process.

10 Test Suite Evaluation

10.1 Compilation Output

```
> npm run build

[...]

    Finished `test` profile [unoptimized + debuginfo] target(s) in 16.36s
     Running unittests src/lib.rs (canopy/target/debug/deps/canopy-a00bd83e90bfe779)
```

10.2 Tests Output

```
> npm test

[...]

  541 passing (2s)
  3 pending
```

10.3 Notes on the Test Suite

The Canopy test suite demonstrates exceptional coverage across all core protocol functionality, achieving 100% coverage for all 33 existing instruction files through a comprehensive set of 57 test files organized across 12 distinct test directories. The testing infrastructure effectively validates the protocol's critical workflows, including time-locked governance operations, multi-admin approval mechanisms, investment raise flows, token generation events with vesting integration, and emergency pause controls. The test organization is particularly strong, with clear separation between unit tests, integration tests, security tests, and edge case scenarios, making the codebase highly maintainable and the test intentions transparent.

The suite excels in several key areas: authorization and permission validation are thoroughly tested across all modules, arithmetic overflow protection is verified through dedicated security tests, and the rate limiting mechanisms are validated indirectly through instruction-level tests. Integration tests provide valuable end-to-end validation of complex workflows like full raise cycles and cancellation-refund sequences, while specialized test categories for database synchronization, client utilities, and worker processes demonstrate attention to the full application stack beyond just the on-chain program logic.

However, there are notable areas for improvement. The missing `setup_tge_distribution.rs` instruction file indicates an incomplete implementation or documentation gap that should be addressed to ensure the TGE module is fully functional. More significantly, several integration tests require PostgreSQL infrastructure, creating a barrier to entry for contributors who want to run the complete test suite locally without additional setup. The dependency on external database infrastructure also means that some critical integration paths may not be regularly validated in lightweight CI environments. Expanding the test suite to include more standalone integration tests that don't require external dependencies, adding explicit documentation about which tests require database setup, and investigating whether database-dependent tests could be supplemented with mock-based alternatives would significantly improve the accessibility and reliability of the testing framework.