BLOCKAPEX

SMART CONTRACT SECURITY

V 1.0

Date : 08-09-2025

Prepared For : Open Game Protocol

About BlockApex

Founded in early 2021 BlockApex is a security-first blockchain consulting firm. We offer services in a wide range of areas including Audits for Smart Contracts, Blockchain Protocols, Tokenomics along with Invariant development (i.e., test-suite) and Decentralized Application Penetration Testing. With a dedicated team of over 40+ experts dispersed globally, BlockApex has contributed to enhancing the security of essential software components utilized by many users worldwide, including vital systems and technologies.

BlockApex has a focus on blockchain security, maintaining an expertise hub to navigate this dynamic field. We actively contribute to security research and openly share our findings with the community. Our work is available for review at our public repository, showcasing audit reports and insights into our innovative practices.

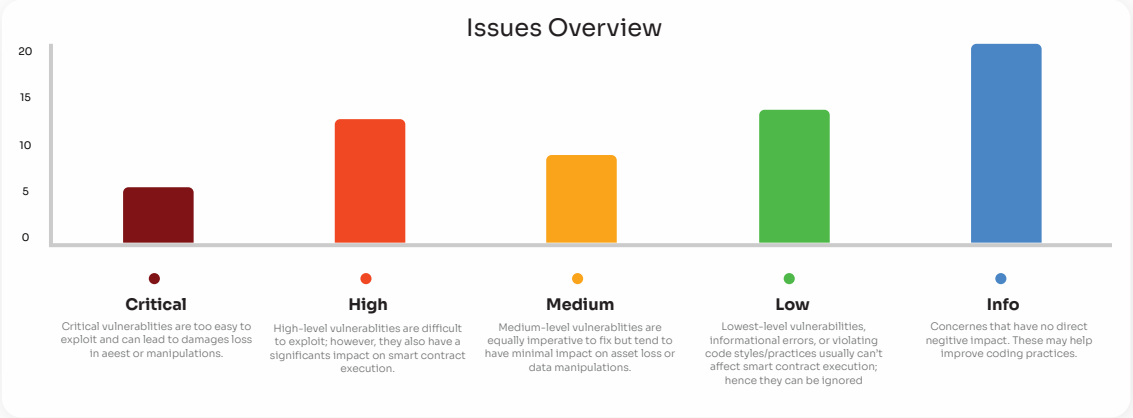
To stay informed about BlockApex's latest developments, breakthroughs, and services, we invite you to follow us on Twitter and explore our GitHub. For direct inquiries, partnership opportunities, or to learn more about how BlockApex can assist your organization in achieving its security objectives, please visit our Contact page at our website , or reach out to us via email at hello@blockapex.io.

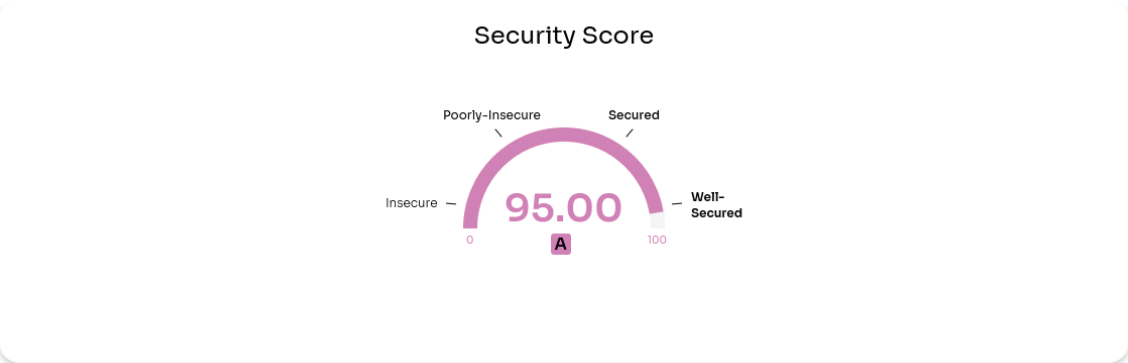
Contents

1 Executive Summary

Open Game Protocol (OGP) is a Solana-based framework designed to facilitate dynamic token economies using a bonding curve mechanism. Built with Rust and the Anchor framework, OGP enables automated price discovery, token minting, and liquidity management without relying on centralized intermediaries. By leveraging smart contracts, it ensures trustless and efficient token distribution, making it an ideal solution for gaming economies, NFT projects, and decentralized finance (DeFi) applications.

The protocol provides a flexible infrastructure where game developers and ecosystem participants can configure bonding curves, swap tokens, and optimize liquidity provisioning. With seamless integration across Solana's devnet and mainnet, OGP supports scalable, on-chain tokenomics tailored for interactive and evolving digital economies.




Security Score
Poorly-Insecure
Secured
Insecure
95.00
Well-
Secured
0
A
100

1.1 Scope

1.1.1 In Scope

Following Pull requests and Commit Hashes were Audited during the Course of Audit:

1. https://github.com/BlockApex/og-contracts
   - (cdff9372998c2d73ead48550620dd1047be7ecfb)
2. https://github.com/OpenGameProtocol/og-contracts/tree/brice/reward-dispatch
   - (32e72bcc619919ad6367cdec59cfdcf737d94260)
3. https://github.com/OpenGameProtocol/og-contracts/pull/38
4. https://github.com/OpenGameProtocol/og-contracts/pull/25
5. https://github.com/OpenGameProtocol/og-contracts/pull/28
6. https://github.com/OpenGameProtocol/og-contracts/pull/40
7. https://github.com/OpenGameProtocol/og-contracts/pull/77

1.1.2 Out of Scope

Anything not mentioned in the Scope.

1.2 Methodology

Each pull request was audited independently in an interative audit process. Initially during the recon phase the understanding ofthe codebase was developed which evolved during the course of audit as new functionalities were added. Auditors kept in constant communication with the devs to understanding functionalities and assumptions. The auditors perfomed manual audits line by line keeping in view the business logic ofthe protocol to find logical and security flaws in the codebase.

1.3 Status Descriptions

Acknowledged: The issue has been recognized and is under review. It indicates that the relevant team is aware ofthe problem and is actively considering the next steps or solutions.

Fixed: The issue has been addressed and resolved. Necessary actions or corrections have been implemented to eliminate the vulnerability or problem.

Closed: This status signifies that the issue has been thoroughly evaluated and acknowledged by the development team. While no immediate action is being taken.

1.4 Summary of Findings Identified

S.NO	SEVERITY	FINDINGS	STATUS
#1	CRITICAL	Donation Attack Causes DoS On Swapping Due to Balance Mismatch	FIXED
#2	CRITICAL	Unauthorized Configuration and Reinitialization of Bonding Curve	FIXED
#3	CRITICAL	Gaining Free gtokens via staking arbitrary mtoken_mint	FIXED
#4	CRITICAL	Lack of Claimed Amount Deduction Allows Unlimited Claims	CLOSED
#5	HIGH	Incorrect Handling of Token2022 Extensions Leads to Accounting Inconsistencies	FIXED
#6	MEDIUM	Vesting Phase DoS Due to Incompatible Token2022 mtoken Mint Handling	FIXED
#7	MEDIUM	Inconsistent Distributor Reuse Allows Multiple mtoken Launches for the Same gtoken	FIXED
#8	MEDIUM	Failure to Update Claim Timestamp Enables Premature Claims	ACKNOWLEDGED
#9	MEDIUM	Hardcoded MToken Balance Returns Incorrect Financial Data	FIXED
#10	MEDIUM	Lack of Validation Allows Multiple G-Token Pairs for the Same M-Token	CLOSED
#11	LOW	Missing Validation on Bonus and Vesting Parameters in StartLaunchProgram	FIXED
#12	LOW	Precision Loss in Referrer Bonus Calculation Can Cause Unstake Failure	FIXED

S.NO	SEVERITY	FINDINGS	STATUS
#13	LOW	Grant amplification (division-by-small-value) and issuance of unbacked launch-options leading to large, manipulable token claims	PARTIALLY FIXED
#14	LOW	Missing Validations on the mint accounts being used	CLOSED
#15	LOW	getLeftOverAmount Returns Last Observed Value Instead of Minimum	FIXED
#16	LOW	Participant Count Not Decreased After Full Unstake	ACKNOWLEDGED
#17	LOW	Potential Front-running attack on the global configuration initialization.	FIXED
#18	LOW	Batch the tx during launchBondingCurve	ACKNOWLEDGED
#19	INFO	Redundant Checks applied.	FIXED
#20	INFO	Use Anchors macro instead of Manual Size Calculation	FIXED
#21	INFO	Missing Validation for G-Token Metadata Fields	FIXED
#22	INFO	Unused and Unnecessary Parameters in mint_gtoken	PARTIALLY FIXED
#23	INFO	Missing Validation virtual_mtoken_reserves_final Should Be Greater Than virtual_mtoken_reserves_initial	FIXED
#24	CRITICAL	Usage of Unchecked Accounts can lead to funds loss	FIXED

S.NO	SEVERITY	FINDINGS	STATUS
#25	HIGH	Init can cause DOS with ATAs	FIXED
#26	MEDIUM	State Inconsistency in SwapCpAmm Leading to Data corruption	FIXED
#27	INFO	Claim_time being passed from outside in claim_fees	FIXED
#28	INFO	Admin and Oracle address should not be same	FIXED
#29	INFO	Arbitrary swap_time can be set during swapping	FIXED
#30	INFO	Redundant Initialization Check in Configure Distributor Function	FIXED
#31	INFO	Missing Validation on distributor_id during staking (ONCHAIN PROGRAM)	FIXED
#32	INFO	BigInt Conversion Failure with Decimal uiAmount Values	FIXED
#33	INFO	Inconsistent Blockhash Handling in Transaction Builders	ACKNOWLEDGED
#34	INFO	Unused Partner Parameter and User-Controlled Blockhash in Partner Metadata Creation	FIXED
#35	HIGH	Passing of Arbitrary Referrer	ACKNOWLEDGED
#36	MEDIUM	Precision loss on referrer bonus calculation	ACKNOWLEDGED

S.NO	SEVERITY	FINDINGS	STATUS
#37	LOW	Missing validation on vesting_speed_bonus	FIXED
#38	LOW	Overwriting of vesting_speed_bonus during vesting unitialization	FIXED
#39	LOW	Resetting of Participant_count during Vesting	FIXED
#40	LOW	Missing Validation on staking and referral allocation amount	ACKNOWLEDGED
#41	INFO	Unnecessary Initialization of Referrer during Unstake	FIXED
#42	INFO	Redundant distribution_pool Account in ProcessClaim Instruction	FIXED
#43	INFO	Wrong Error Emission	FIXED
#44	INFO	Use of Raw u8 for Phases Instead of Enum	FIXED
#45	INFO	Referrer Accounts Should Be Optional to Reduce Unnecessary Account Handling	ACKNOWLEDGED
#46	INFO	Reservations on protocol_fee_recipient	CLOSED
#47	INFO	No validation on the distributor_state reward_amount	CLOSED
#48	INFO	Comprehensive Lack of Input Parameter Validation	CLOSED

2 Findings and Risk Analysis

2.1 Donation Attack Causes DoS On Swapping Due to Balance Mismatch

Severity: Critical

Status: Fixed

Location:

- `solana/programs/bonding_curve/src/instructions/swap.rs`
- `@cdff9372998c2d73ead48550620dd1047be7ecfb`

Description: In the swap() functionality following check is applied , which essentially tries to check the VaultBalance and it requires that the `curve_gtoken_amount` should be equal to `virtual_gtoken_reserves + bonding_curve.dormant_gtoken_reserves`.

```
require!(
            self.curve_gtoken_account.amount == (self.bonding_curve.virtual_gtoken_reserves
                 + self.bonding_curve.dormant_gtoken_reserves),
            BondingCurveError::VaultBalanceMismatch
        );
```

This check opens room for donation attack where we can manipulate the `curve_gtoken_account.amount` by sending the gtoken to the `curve_gtoken_amount` , this would make the above check fail and the swapping functionality would fail essentially putting the curve in DOS state

Recommendation: To mitigate this issue it is advisable to perform the internal accounting via fields defined in a PDA as the accounts balance is manipulatable via donation attacks.

2.2 Unauthorized Configuration and Reinitialization of Bonding Curve

Severity: Critical

Status: Fixed

Location:

- `programs/bonding_curve/src/instructions/configure_curve.rs`
- `@cdff9372998c2d73ead48550620dd1047be7ecfb`

Description: The configureCurve function allows any signer to configure or reinitialize the bonding curve settings, regardless of whether they are the original creator ofthe gToken. There are no ownership or access control checks to ensure that only the original creator or an authorized entity can configure the bonding curve. As a result, an attacker can arbitrarily modify key parameters such as virtual_mtoken_reserves_initial, virtual_mtoken_reserves_final, and protocol_fee_recipient, potentially disrupting the bonding curves completion or migration process.

Recommendation: 1. Implement an ownership check to ensure only the original creator or an authorized admin can call configureCurve. 2. Introduce an access control mechanism, such as an authority field, that restricts modifications to authorized accounts. 3. Add a one-time initialization flag to prevent reconfiguration after the bonding curve is set.

2.3 Gaining Free gtokens via staking arbitrary mtoken_mint

Severity: Critical

Status: Fixed

Location:

solana/programs/claim_distribution/src/instructions/stake_launch_option.rs

Description: The `stake_launch_option` function accepts any arbitrary `mtoken_mint` without validating that it matches the legitimate basecoin (mtoken) that should be used for the specific launcher. This allows attackers to stake worthless tokens while earning launch options that can be exercised for valuable gTokens.

```
pub struct StakeLaunchOption&lt;&#x27;info&gt; {
..
..
pub mtoken_mint: Box&lt;InterfaceAccount&lt;&#x27;info, Mint&gt;&gt;,
..
..
}
```

Proof of Concept: TBC

Recommendation: It is recommended to put mtoken_mint in the Launcher struct and also validate it during the stake_launch_options.

2.4 Lack of Claimed Amount Deduction Allows Unlimited Claims

Severity: Critical

Status: Closed

Location:

- programs/claim_distribution/src/states/distributor.rs

Description: The process_claim function does not subtract the claimed amount from the user_claim account. As a result, users can repeatedly claim the same reward without limitation. This allows them to drain the entire distribution_pool_ata by making multiple claims, effectively claiming more tokens than allocated.

1. Record a claim with claim_amount = 100 and claim_timestamp = 10 seconds:

```
await program.methods.recordClaim(
    new anchor.BN(1),
    oracle_key.publicKey,
    new anchor.BN(100),  // User should be able to claim 100 tokens
    new anchor.BN(10)     // Claim allowed after 10 seconds
).accounts({
    claimant: claimant.publicKey,
    oracle: oracle_key.publicKey,
    tokenMint: mtokenMint,
}).signers([oracle_key, claimant]).rpc();
```

2. Claim the tokens at claim_timestamp = 11 seconds:

```
await program.methods.processClaim(
    new anchor.BN(1),
    oracle_key.publicKey,
    new anchor.BN(100),  // Claim 100 tokens
    new anchor.BN(11)     // Claim at 11 seconds
).accounts({
    tokenMint: mtokenMint,
    distributionPoolAta: distributor_config.distributorTokenAccount.toString(),
    oracle: oracle_key.publicKey,
    claimant: claimant.publicKey,
}).signers([oracle_key, claimant]).rpc();
```

3. Reclaim the same amount at claim_timestamp = 12 seconds:

```
await program.methods.processClaim(
    new anchor.BN(1),
    oracle_key.publicKey,
    new anchor.BN(100),  // Claim again
    new anchor.BN(12)     // Claim at 12 seconds
).accounts({
    tokenMint: mtokenMint,
    distributionPoolAta: distributor_config.distributorTokenAccount.toString(),
    oracle: oracle_key.publicKey,
    claimant: claimant.publicKey,
}).signers([oracle_key, claimant]).rpc();
```

Output

```
User claiming their claim amount: amount = 100
User reclaiming their claim amount: amount = 200
```

The user can continuously call process_claim every second, reclaiming the same amount repeatedly without any restriction, leading to infinite claims.

Impact

- Users can drain all tokens from distribution_pool_ata.
- No limit is enforced on how many times a user can claim their allocated amount
- This can result in unauthorized claims oftokens meant for other users.

Recommendation

- Deduct the claimed amount from user_claim.amount after a successful claim.
- Prevent claims if user_claim.amount is already zero.
- Implement tracking to ensure users cannot reclaim more than their allocated amount.

2.5 Incorrect Handling of Token2022 Extensions Leads to Accounting Inconsistencies

Severity: High

Status: Fixed

Location:

- src/instruction/stake_launch_option.rs
- src/instruction/unstake_launch_option.rs

Description: The protocol accepts mtoken as either a standard SPL token or a Token2022 mint. However, the implementation does not validate or handle Token2022 extensions, such as the Fee-on-Transfer extension.

During staking, the program transfers mtoken_amount from the participant to the launcher account, but the recorded state is updated using the input parameter (mtoken_amount) rather than the actual amount received. Ifthe mint has a fee or other extension that reduces the transferred amount, the protocol's internal accounting will be inflated compared to the real tokens held.

This misalignment propagates to staking, referral, and reward calculations, and in the case of unstaking, it enables participants to withdraw more tokens than they effectively deposited.

Proof of Concept: Assume the mtoken mint is a Token2022 mint with a 2% fee-on-transfer extension enabled.

A participant calls the stake instruction with: mtoken_amount = 100

- Expected transfer: 100 tokens.
- Actual received by launcher_mtoken_ata: 98 tokens (after fee deduction).
- State now shows 100 tokens staked.
- Actual vault balance is only 98 tokens.

Later, the participant calls unstake.

- The program believes 100 tokens are available.
- Participant is able to withdraw 100 tokens, even though only 98 were deposited.
- Result: 2-token protocol loss.

The same miscalculation propagates to:

- Referral rewards (calculated on 100, not 98).
- Allocation logic, leading to inflated balances.

Recommendation: * Explicitly restrict supported mint types: Disallow Token2022 mints or only allow standard SPL tokens. * If Token2022 mints are supported, query and handle extensions explicitly,

ensuring transfer amounts are reconciled with actual received balances. * Update state variables based on the post-transfer received amount instead ofthe input parameter.

Developer Response Fixed by using mtoken balance before and after the transfer as the locked mtoken amount .

Auditor Response Using pre/post balance deltas addresses the fee-on-transfer case. However, Token2022 introduces several other extensions (e.g. Permanent Delegate , interest-bearing, transfer hooks e.t.c) that may still affect . Unless all relevant extensions are explicitly handled, inconsistencies could persist. I recommend performing comprehensive extensions checks when Token2022 mints are allowed.

Developer Response Fixed by making an explicit list of allowed extensions along with comments explaining why they can be allowed. We generally ensured that extensions that do not make tokens untradeable are compatible with our program.

2.6 Vesting Phase DoS Due to Incompatible Token2022 mtoken Mint Handling

Severity: Medium

Status: Fixed

Location:

src/instruction/exercise_launch_options.rs

Description: In the ExerciseLaunchOptions instruction context, the mtoken_mint account is defined as:

```
pub mtoken_mint: Box&lt;Account&lt;&#x27;info, Mint&gt;&gt;
```

This enforces the mint to be a legacy SPL Token mint owned by the SPL Token program. However, the protocol is designed to support both legacy SPL tokens and Token2022 mints.

Because ofthis restriction, any mtoken that is a Token2022 mint will not be accepted, causing transactions for launches and vesting to fail. This results in the entire vesting phase of gtoken launches using Token2022 mints being blocked.

Proof of Concept: TBC

Recommendation: Update the instruction context to use:

```
pub mtoken_mint: InterfaceAccount&lt;&#x27;info, Mint&gt;, //owner = SPL-Token or Token-2022
```

This ensures compatibility with both SPL Token and Token2022 programs, aligning with the protocol's intended design.

2.7 Inconsistent Distributor Reuse Allows Multiple mtoken Launches for the Same gtoken

Severity: Medium

Status: Fixed

Location:

src/instruction/start_launch_program.rs

Description: In the StartLaunchProgram instruction, the distributor account is derived only from the gtoken_mint, admin_key, and distributor_id:

```
#[account(
mut,
seeds = [
DISTRIBUTOR_SEED_PREFIX.as_bytes(),
gtoken_mint.as_ref(),
admin_key.as_ref(),
&amp;distributor_id.to_le_bytes()
],
bump
)]
pub distributor: AccountInfo&lt;&#x27;info&gt;,
```

The associated launcher is then derived using this distributor together with the mtoken_mint.

This design does not explicitly enforce that a distributor account can only be tied to a single mtoken. As a result, an admin may mistakenly create multiple launchers for the same gtoken but with different mtoken mints using the same distributor account.

During the staking phase, both launchers work as expected, but in later phases (e.g., config_distribution and vesting), the protocol only supports one launcher tied to a distributor. This inconsistency causes failures when multiple launchers exist under the same distributor, effectively breaking vesting for the additional launchers.

Proof of Concept: TBC

Recommendation: * Enforce a one-to-one relationship between a distributor and an mtoken mint in StartLaunchProgram. * Validate that a distributor ID has not already been used for a given gtoken with a different mtoken before allowing a new launcher to be initialized. * Alternatively, derive the distributor account seeds to also include the mtoken_mint, ensuring uniqueness at the account level.

2.8 Failure to Update Claim Timestamp Enables Premature Claims

Severity: Medium

Status: Acknowledged

Location:

- programs/claim_distribution/src/states/distributor.rs

Description: The record_claim function does not update user_claim.timestamp with the claim_timestamp provided in the parameter. This omission allows users to claim tokens at any time since user_claim.timestamp remains zero. Additionally, the check that claim_timestamp must be strictly greater than user_claim.timestamp is ineffective because user_claim.timestamp is never updated.

1. Call recordClaim with a claim_timestamp of 10 seconds:

```
await program.methods.recordClaim(
    new anchor.BN(1),
    oracle_key.publicKey,
    new anchor.BN(100),
    new anchor.BN(10) // Claim allowed after 10 seconds
).accounts({
    claimant: claimant.publicKey,
    oracle: oracle_key.publicKey,
    tokenMint: mtokenMint,
}).signers([oracle_key, claimant]).rpc();
```

2. Call processClaim at 1 second, bypassing the intended delay:

```
await program.methods.processClaim(
    new anchor.BN(1),
    oracle_key.publicKey,
    new anchor.BN(100),
    new anchor.BN(1) // Only 1 second has passed, but claim is still processed
).accounts({
    tokenMint: mtokenMint,
    distributionPoolAta: distributor_config.distributorTokenAccount.toString(),
    oracle: oracle_key.publicKey,
    claimant: claimant.publicKey,
}).signers([oracle_key, claimant]).rpc();
```

Output: amount: 100

Impact * Users can claim tokens before the intended claim time. * The system does not enforce the delay period, allowing claims earlier than expected.

Recommendation

- Update user_claim.timestamp = claim_timestamp in the record_claim function.

- Ensure claim_timestamp is greater than current blockchain time (Clock::get()?.unix_timestamp) before processing the claim.

2.9 Hardcoded MToken Balance Returns Incorrect Financial Data

Severity: Medium

Status: Fixed

Location:

- solana/src/api/getDistributorStatus.ts
- #L-107@cb0123e5b0a0677e9a98689e307d2a3bf36478b9a

Description: The mtokenBalance is hardcoded to BigInt(0) instead of querying the actual token account balance. This causes the getDistributorStatus API to return incorrect financial information about MToken holdings, which could mislead users about their actual token balances and affect decision-making in the protocol.

```
const mtokenBalance = BigInt(0);
```

Recommendation: Replace the hardcoded value with proper token account balance query:

2.10 Lack of Validation Allows Multiple G-Token Pairs for the Same M-Token

Severity: Medium

Status: Closed

Location:

programs/bonding_curve/src/instructions/mint_gtoken.rs

Description: The mint_gtoken function does not enforce the invariant that a single M-Token (meme token) should only be paired with one unique G-Token. Currently, the function allows the same mtoken_mint to be reused with different gtoken_mint values, enabling multiple G-Tokens to be paired with the same M-Token. This violates the expected uniqueness constraint and could lead to inconsistencies in the bonding curve mechanism.

Proof of Concept: * A user successfully mints a G-Token (gtokenMint) with an M-Token (mtokenMint):

```
await program.methods
 .mintGtoken(&amp;quot;g-pepe-token&amp;quot;, &amp;quot;g-pepe&amp;quot;, &amp;quot;&amp;quot;, new anchor.BN(1),
      1)
 .accounts({
   creator: user1.publicKey,
   gtokenMint: gtokenMint1.publicKey,
   mtokenMint: mtokenMint,  // First pairing of mtokenMint
   gtokenMetadataAccount: pda,
 })
 .signers([user1, gtokenMint1])
 .rpc();
```

- The same mtokenMint is then used to mint a different G-Token (new_gtoken_mint), which should not be allowed:

```
await program.methods
 .mintGtoken(&amp;quot;g-pepe-token&amp;quot;, &amp;quot;g-pepe&amp;quot;, &amp;quot;&amp;quot;, new anchor.BN(1),
      1)
 .accounts({
   creator: user2.publicKey,
   gtokenMint: new_gtoken_mint2.publicKey,
   mtokenMint: mtokenMint, // Same M-Token used again with a different G-Token
   gtokenMetadataAccount: pda,
 })
 .signers([user2, new_gtoken_mint2])
 .rpc();
```

Recommendation: * Implement a check in mint_gtoken to ensure that an M-Token can only be associated with one G-Token. * Store a mapping of mtokenMint to its associated gtokenMint, and prevent minting a new G-Token if an entry already exists for the given M-Token. * Enforce constraints at the account level to prevent multiple G-Tokens from being created for the same M-Token.

2.11 Missing Validation on Bonus and Vesting Parameters in StartLaunchProgram

Severity: Low

Status: Fixed

Location:

src/instruction/start_launch_program.rs

Description: In the StartLaunchProgram implementation, certain input parameters are written directly to state without validation.

- referrer_bonus and referred_bonus are intended to represent percentages between 0–100, but no range checks are enforced.
- vesting_duration is accepted without validation and could be set to 0, which may result in an invalid or nonsensical vesting configuration.
- Additionally, no check ensures that the gtoken_mint and mtoken_mint are different, allowing accidental misconfiguration of launches.

Proof of Concept: TBC

Recommendation: Enforce explicit checks in process() to validate input parameters:

- Require 0 <= referrer_bonus <= 100.
- Require 0 <= referred_bonus <= 100.
- Require vesting_duration > 0.

Add a validation to ensure gtoken_mint != mtoken_mint.

2.12 Precision Loss in Referrer Bonus Calculation Can Cause Unstake Failure

Severity: Low

Status: Fixed

Location:

src/instruction/exercise_launch_options.rs

Description: The referral bonus calculation in both StakeLaunchOption and UnstakeLaunchOption instructions uses integer division, which introduces rounding errors. Over multiple staking and unstaking operations, these small precision mismatches accumulate.

When a participant fully unstakes, the computed referrer_bonus may exceed the recorded referrer_launch_option_amount, causing an underflow and transaction failure. This prevents the affected participant from unstaking their entire amount.

Proof of Concept: (with 35% referral bonus)

- First Stake

Stake: 103111 (9 decimals) Bonus = 103111 * 35 / 100 = 36,088.85 → 36,088 (rounded down) Referrer bonus recorded = 36,088

- Second Stake

Stake: 1 (9 decimals) Bonus = 1 * 35 / 100 = 0.35 → 0 (rounded down) Total stake = 103112 Bonus recomputed = 36,089.2 → 36,089 Referrer bonus is now higher than expected by +1.

- Unstake

When the user tries to fully unstakes, protocol expects to subtract 36,089 from the referrer's recorded 36,088. This causes an underflow (-1) → transaction fails.

Recommendation: Use consistent rounding strategy across stake/unstake (always floor/ceil).

2.13 Grant amplification (division-by-small-value) and issuance of unbacked launch-options leading to large, manipulable token claims

Severity: Low

Status: Partially Fixed

Location:

src/instruction/configure_distributor.rs src/instruction/exercise_launch_options.rs

Description: The ConfigureDistributor::process code computes a “newly granted” staking launch-option amount for a launch partner using a formula that divides by current_total_staking. When current_total_staking is small (or can be made small by an adversary), the computed grant can explode far beyond the intended slack. The code then unconditionally increases both the launcher’s global staking LO total and the launch partner’s escrow LO fields without ensuring those additional LO are backed by locked mTokens.

This combination allows an attacker or colluding party to:

- Receive massive, unbacked LO by manipulating current_total_staking (e.g., by coordinated unstake / front-run), then
- Convert those LO to gTokens after vesting (or otherwise liquidate them), diluting and stealing value from honest participants and the protocol.

protocol funds can be extracted (or honest participants heavily diluted), and core invariants (LO <= backing) are broken.

```
let slack = staking_launch_option_target.saturating_sub(current_total_staking);
let total_launch_option_amount = launcher.total_staking_launch_option_amount
.checked_add(launcher.total_referrer_launch_option_amount)?
.checked_add(launcher.total_referred_launch_option_amount)?;
let newly_granted_to_lp = if current_total_staking == 0 {
staking_launch_option_target
} else {
(slack * total_launch_option_amount) / current_total_staking
};
```

A: Grant formula is unstable / self-referential and divides by a possibly tiny denominator

- The implemented formula is derived in comments as solving a self-referential equation for x:

```
x = (slack / target) * (total_launch_option_amount + x)
```

and then the code uses the rearranged expression:

```
x = slack * total_launch_option_amount / current_total_staking
```

- where current_total_staking represents the current staked LO (denominator).

- Problem: current_total_staking can be small (or 0). If small, x grows large because you divide by a tiny number. The code tries to guard current_total_staking == 0 via a branch, but for tiny but non-zero denominators the result can still be unreasonably large.
- Additionally the code silently clamps grant_u128 to u64::MAX, which would mask extreme error conditions instead offailing safely.

B: Newly granted LO are not collateralized

- After calculating newly_granted_to_lp, the code directly updates launcher.total_staking_launch_option_amount and lp_escrow.staking_launch_option_amount without requiring additional mTokens to be locked in launcher_mtoken_ata (or any other escrow). No check enforces locked_mtoken_amount >= staking_launch_option_amount or any similar invariant.
- This means LO are effectively minted/assigned on the protocol state without real mToken collateral — they are “unbacked”.

C: Front-running / manipulation makes attack easy

- current_total_staking is read at the call time. An attacker can (and will) front-run or coordinate an unstake transaction that reduces the denominator right before the ConfigureDistributor transaction gets executed (or initiate the ConfigureDistributor when current_total_staking is small), producing an outsized grant.
- Admin or privileged parties that call ConfigureDistributor (or can set staking_launch_option_target) might also be complicit or misconfigure values, allowing excessive grants.

D: Secondary effects: dilution of honest participants during vesting/exercise

- LO are later converted to gTokens pro rata against total_launch_option_amount. Granting excessive LO to an LP increases the denominator used during gToken allocation, thereby reducing honest users’ share. This is an immediate economic dilution.

Proof of Concept: #####Concrete numeric examples (PoC / walk-through) Normal-expected case

- staking_launch_option_target = 1500
- total_launch_option_amount = 1300 (current LO in system)
- current_total_staking = 1000 (mtokens staked → staking LO)

Compute:

- slack = 1500 - 1000 = 500
- newly_granted_to_lp = 500 * 1300 / 1000 = 650
- After grant, total LO = 1300 + 650 = 1950 — plausible.

Attack with front-run / unstake before ConfigureDistributor executes Attacker unstakes 700 just before ConfigureDistributor call lands:

- current_total_staking = 1000 - 700 = 300
- total_launch_option_amount may fall (if attacker removed their own LO) to 600
- slack = 1500 - 300 = 1200
- newly_granted_to_lp = 1200 * 600 / 300 = 2400

Result:

- Grant = 2400 LO to LP instead of expected 650.
- New total LO = 600 + 2400 = 3000.
- LP controls 80%+ of LO while having no extra mToken collateral, enabling later substantial gToken claims — direct economic theft/dilution.

Vesting dilution example Suppose allocation_gtoken_amount = 1000 to distribute to non-team pool. A legitimate user has 500 LO.

- Before extra LP grant: user share = 500 * 1000 / 1300 = 384 gTokens.
- After normal grant of 650: total LO=1950, user share = 500 * 1000 / 1950 = 256 gTokens.
- After exploit grant of 2400: total LO=3000, user share = 500 * 1000 / 3000 = 166 gTokens. User goes from 384 → 166 gTokens due to attack (huge dilution).

Recommendation: 1. #####Immutable or Pre-Committed Targets The staking_launch_option_target should be set at the time of launcher creation, before staking begins, and remain immutable throughout the staking phase. This ensures participants know the target in advance and cannot be surprised by mid-staking changes.

2. #####Enforce Bounds and Caps Set a maximum limit on staking_launch_option_target (fixed by governance or as a fraction oftotal supply). Ensure that any grant cannot exceed slack or, at most, a small multiple of it (e.g., 2×). Do not silently clamp values to u64::MAX; instead, fail with an explicit error when the limit is exceeded.
3. #####Require Collateralization Before Crediting Launch Options Every launch option (LO) must always be backed by underlying mTokens. Before increasing LO balances for either the LP or the launcher, the protocol should verify sufficient collateralization.

This can be enforced at the LP level by checking that the LP's locked mTokens cover the LO balance after the grant, or at the global level by checking that total locked mTokens in the launcher cover all outstanding LO.

Developer Response Fixed partly by scaling all the launch options when a launch partner is present. The concern about awarding gtokens for launch options not backed by any staked mtokens is by design.

In reality we believe that the other concerns raised should have been mathematically impossible due to the staking/referral logic implemented in the staking/unstaking/exercising instructions. Newly implemented logic should make it clearer while producing the same end result for the user.

2.14 Missing Validations on the mint accounts being used

Severity: Low

Status: Closed

Location:

solana/programs/claim_distribution/src/instructions/claim_fees.rs#L100-L101 solana/programs/claim_distribution/sr
L92

Description: In the ClaimFees and Swap_cp_amm gtoken_mint, qtoken_mint is passed in the accounts struct. Currently there is no validation applied on the accounts being passed hence it is possible to provide any mint account not aligning with the mint accounts defined in the distributor state.

Proof of Concept: TBC

Recommendation: It is recommended to validate the mint accounts from the distributor_state as while configuring the distributor these mints are set there.

2.15 getLeftOverAmount Returns Last Observed Value Instead of Minimum

Severity: Low

Status: Fixed

Location:

- og-contracts/solana/src
- @cb0123e5b0a0677e9a98689e307d2a3bf36478b9a

Description: The getLeftOverAmount function in getBondingCurveStatus.ts contains a **logic error** where the fallback mechanism returns the **last observed leftover value** instead ofthe **minimum observed leftover value** as documented in the code comment.

```
1 // Fallback: return the minimum observed leftover (safest lower-bound).
2 // @audit as per the comment above , this will not take the minimum observed leftover
      rather it would take the last observed leftover.
3 const fallback = previousSnapshot.leftOver;
4 return fallback &gt; BigInt(0) ? fallback : BigInt(0);
```

The function takes up to 3 snapshots of vault balance and fees, but when consecutive snapshots differ, it returns the **last observed value** rather than tracking and returning the **minimum observed value**.

Recommendation: Implement proper minimum tracking in the fallback mechanism as per the comment.

2.16 Participant Count Not Decreased After Full Unstake

Severity: Low

Status: Acknowledged

Location:

src/instruction/unstake_launch_option.rs

Description: In the UnstakeLaunchOption instruction, when a participant fully unstakes their mtoken, the participant_count in the Launcher account (self.launcher.participant_count) is not reduced.

This means participants who have completely exited before the vesting phase (phase 3) are still counted as active. It can lead to inaccurate participant tracking and misleading metrics.

Proof of Concept: TBC

Recommendation: Update the UnstakeLaunchOption logic to decrease self.launcher.participant_count when a participant fully unstakes before the vesting phase begins. This ensures that participant tracking reflects only currently staked users.

Developer Response Will leave as such. This is by design, using loose definition of "participant" where anyone who interacted with the launcher is counted as having participated.

2.17 Potential Front-running attack on the global configuration initialization.

Severity: Low

Status: Fixed

Location:

- solana/programs/bonding_curve/src/instructions/configure.rs
- @cdff9372998c2d73ead48550620dd1047be7ecfb

Description: When the program is deployed by the admin, the first function which he needs to call is the configure() function and pass the new_config to set the global config. In the current implementation a require check is used for admin checks.

```
require!(
            new_config.authority.eq(&amp;self.admin.key()),
            BondingCurveError::NotAuthorized
        );
```

This check just says ifthe authority field of new_config being passed is equal to the signers keys , go ahead. This check is insufficient as an attacker can frontrun and pass a new_config and with authority == signer and it will go ahead.
This issue allows the attacker to frontrun the legitimate admin and set the global configuration.

Recommendation: To mitigate this issue, it is advised to embed a Default admin as a constant and then check the process() function call against that admin. This way only the defined admin is able to set the global configurations.

2.18 Batch the tx during launchBondingCurve

Severity: Low

Status: Acknowledged

Location:

- og-contracts/solana/src
- @cb0123e5b0a0677e9a98689e307d2a3bf36478b9a

Description: The launchBondingCurve function in getLaunchBondingCurveTxs.ts executes transactions sequentially rather than atomically, creating a critical window for front-running attacks. The current implementation sends transactions in the following order:

```
// Current vulnerable implementation
if (launchBondingCurveTxs.createConfigTx) {
  await provider.sendAndConfirm!(launchBondingCurveTxs.createConfigTx, [], {
    commitment: &#x27;finalized&#x27;,
    skipPreflight: true,
  });
}
await provider.sendAndConfirm!(launchBondingCurveTxs.createPoolTx, [], {
  commitment: &#x27;finalized&#x27;,
  skipPreflight: true,
});
if (launchBondingCurveTxs.swapBuyTx) {
  await provider.sendAndConfirm!(launchBondingCurveTxs.swapBuyTx, [], {
    commitment: &#x27;finalized&#x27;,
    skipPreflight: true,
  });
}
```

The Problem: After createPoolTx is confirmed but before swapBuyTx executes, there's a critical time window where:

- The bonding curve pool is live and tradeable
- The admin's intended first swap hasn't executed yet
- MEV bots and front-runners can exploit this window

Recommendation: Implement Atomic Transaction Batching: Combine createPoolTx and swapBuyTx into a single atomic transaction

2.19 Redundant Checks applied.

Severity: Info

Status: Fixed

Location:

- solana/programs/bonding_curve/src/instructions/swap.rs
- @cdff9372998c2d73ead48550620dd1047be7ecfb
- solana/programs/bonding_curve/src/states/bonding_curve.rs
- @cdff9372998c2d73ead48550620dd1047be7ecfb

Description: In the swap function multiple redundant checks are found , these checks can be performed in the Struct{} using Anchor macros. Redundant checks affect the code readability.

Swap.rs

```
// Verify that the protocol fee recipient passed in the instruction
        // equals the one stored in the bonding curve.
        // @audit not required , we can use the protocol_fee_recipient in the bonding curve account.
        require!(
            self.bonding_curve.protocol_fee_recipient == *self.protocol_fee_recipient.key,
            BondingCurveError::InvalidFeeRecipient
        );

        // Validate the user&#x27;s OG token account
        // @audit redundant check as constraint associated_token::authority = user is already applied.
        require_keys_eq!(
            self.user_mtoken_account.owner,
            self.user.key(),
            BondingCurveError::InvalidTokenAccountOwner
        );

        // @audit redundant check as constraint associated_token::mint = mtoken_mint is already applied.
        require_keys_eq!(
            self.user_mtoken_account.mint,
            self.mtoken_mint.key(),
            BondingCurveError::InvalidTokenAccountMint
        );

        // Validate the user&#x27;s g-token account
        // @audit redundant check as constraint associated_token::authority = user is already applied.
        require_keys_eq!(
            self.user_gtoken_account.owner,
            self.user.key(),
            BondingCurveError::InvalidTokenAccountOwner
        );

        // @audit redundant check as constraint associated_token::mint = gtoken_mint is already applied.
        require_keys_eq!(
            self.user_gtoken_account.mint,
            self.gtoken_mint.key(),
```

```
            BondingCurveError::InvalidTokenAccountMint
        );
```

bonding_curve.rs

```
let curve_id_bytes = &amp;self.curve_id.to_le_bytes(); // convert the unique curve id to bytes
        let (expected_curve_pda, expected_bump) = Pubkey::find_program_address(
            &amp;[
                BondingCurve::SEED_PREFIX.as_bytes(),
                self.gtoken_mint.as_ref(),
                self.mtoken_mint.as_ref(),
                curve_id_bytes,
            ],
            &amp;crate::ID,
        );

        require!(
            curve_bump == expected_bump,
            BondingCurveError::InvalidCurveBump
        );

        // Validate the bonding curve&#x27;s g-token associated token account (ATA)
        let expected_curve_gtoken_ata =
            anchor_spl::associated_token::get_associated_token_address(
                &amp;expected_curve_pda,
                &amp;self.gtoken_mint,
            );
        require!(
            *curve_gtoken_ata.key == expected_curve_gtoken_ata,
            BondingCurveError::InvalidGtokenAccount
        );

        // Validate the bonding curve&#x27;s mtoken associated token account (ATA)
        let expected_curve_mtoken_ata =
            anchor_spl::associated_token::get_associated_token_address(
                &amp;expected_curve_pda,
                &amp;self.mtoken_mint,
            );
        require!(
            *curve_mtoken_ata.key == expected_curve_mtoken_ata,
            BondingCurveError::InvalidMtokenAccount
        );

        require!(
            *token_program_info.key == anchor_spl::token::ID,
            BondingCurveError::InvalidMtokenProgram
        );
```

Recommendation: It is recommended to remove the redundant check for better code clarity & readability as anchor perform most ofthe validation which can be omitted by the developer.

2.20 Use Anchors macro instead of Manual Size Calculation

Severity: Info

Status: Fixed

Location:

- solana/programs/bonding_curve/src/states/config.rs
- @cdff9372998c2d73ead48550620dd1047be7ecfb
- solana/programs/bonding_curve/src/states/bonding_curve.rs
- @cdff9372998c2d73ead48550620dd1047be7ecfb

Description: Anchor provides two convenient ways to calculate the struct size

1. Size_of::
2. INIT_SPACE macro

Currently the size of struct is manually calculated which can be error prone.

```
Impl Config{
pub const SEED_PREFIX: &amp;&#x27;static str = &quot;global-config-v2&quot;;
 // pub const LEN: usize = 8 + size_of::&lt;Config&gt;();

   pub const LEN: usize = 32 + 8 * 7;
}
```

```
impl&lt;&#x27;info&gt; BondingCurve {
   pub const SEED_PREFIX: &amp;&#x27;static str = &quot;bonding_curve&quot;;
   /// Rough estimate: 8 (discriminator) + 233 (fields without padding) + 31 (padding) =
       272 bytes.
   pub const LEN: usize = 272;
. . . . . ..}
```

Recommendation: It is advisable to use the INIT_SPACE macro for calculation ofthe struct size.

Reference: https://www.anchor-lang.com/docs/references/space

2.21 Missing Validation for G-Token Metadata Fields

Severity: Info

Status: Fixed

Location:

- `programs/bonding_curve/src/instructions/mint_gtoken.rs`
- `@cdff9372998c2d73ead48550620dd1047be7ecfb`

Description: The mint_gtoken function does not enforce validation checks on the name, symbol, and uri parameters. As a result, users can mint a G-Token with empty or invalid metadata fields, leading to incomplete or misleading token information.

Recommendation: - Implement validation checks to ensure that name, symbol, and uri are non-empty strings before proceeding with token minting. - Reject transactions where any ofthese fields are empty to enforce correct metadata usage.

2.22 Unused and Unnecessary Parameters in mint_gtoken

Severity: Info

Status: Partially Fixed

Location:

- programs/bonding_curve/src/instructions/mint_gtoken.rs
- @cdff9372998c2d73ead48550620dd1047be7ecfb

Description: The mint_gtoken function includes parameters that are either unused or unnecessary: 1. Unused Parameter (metadata_bump): The metadata_bump parameter is declared but never used within the function. Keeping it serves no purpose and adds unnecessary complexity. 2. Unnecessary Parameter (curve_id): The curve_id is included in the PDA derivation for the bonding_curve account, but it does not impact the uniqueness ofthe PDA. Since gtoken_mint and mtoken_mint already ensure a unique PDA for each bonding curve, curve_id is redundant.If curve_id is required for internal logic, it should be derived within the contract rather than being provided by the user.

Recommendation: - Remove metadata_bump since it is not used. - Remove curve_id from PDA derivation unless there is a specific reason to keep it. If needed, it should be derived internally instead of being passed as user input.

2.23 Missing Validation virtual_mtoken_reserves_final Should Be Greater Than virtual_mtoken_reserves_initial

Severity: Info

Status: Fixed

Location:

- programs/bonding_curve/src/instructions/configure_curve.rs
- @cdff9372998c2d73ead48550620dd1047be7ecfb

Description: The configure_curve function allows setting up a bonding curve with initial and final virtual mToken reserves. However, it does not enforce a check ensuring that virtual_mtoken_reserves_final is greater than virtual_mtoken_reserves_initial. This missing validation can lead to an improperly configured curve where the final reserve value is lower than the initial reserve, potentially causing unexpected behavior in the bonding curve mechanics.

Recommendation: 1. Add a validation check in configure_curve to ensure that virtual_mtoken_reserves_final is strictly greater than virtual_mtoken_reserves_initial. 2. Return an error ifthe condition is not met to prevent invalid curve configurations.

```
if virtual_mtoken_reserves_final &lt;= virtual_mtoken_reserves_initial {
        return Err(ProgramError::InvalidArgument.into());
}
```

2.24 Usage of Unchecked Accounts can lead to funds loss

Severity: Critical

Status: Fixed

Location:

- /solana/programs/claim_distribution/src/instructions/swap_cp_amm.rs
- #L96-L111@cb0123e5b0a0677e9a98689e307d2a3bf36478b9
- solana/programs/claim_distribution/src/instructions/swap_cp_amm.rs
- #L121-L121@cb0123e5b0a0677e9a98689e307d2a3bf36478b9

Description: Both ClaimFees and SwapCpAmm instructions use multiple UncheckedAccount types that are passed directly to CPI calls without validation. The most critical issue is cp_amm_program which can be any program, allowing arbitrary code execution with the distributor PDA as signer.

unchecked accounts: - `cp_amm_program`: Can execute arbitrary code - `pool`: Can claim from wrong pools - `position`: Can claim from wrong LP positions - `*_vault`: Can deposit/withdraw from wrong vaults - `pool_authority`: Can use wrong authority

Recommendation: It is recommended to add validation for all accounts.

```
    self.cp_amm_program.key() == EXPECTED_CP_AMM_PROGRAM_ID,
    TokenDistributionError::InvalidCpAmmProgram
);
```

2.25 Init can cause DOS with ATAs

Severity: High

Status: Fixed

Location:

- solana/programs/claim_distribution/src/instructions/configure_distributor.rs
- #L66-L92@cb0123e5b0a0677e9a98689e307d2a3bf36478b9

Description: In the configure_distributor , 3 different ata's are created which are

1. distributor_gtoken_ata
2. distributor_mtoken_ata
3. distributor_qtoken_ata

Currently init is being used to initialize these ata's but using init makes it prone to DOS scenarios. As anyone can initiate the ata and set its authority to the distributor. When the distributor will try to configure the distributor , init will fail causing the DOS scenario

Recommendation: It is recommended to use the `init_if_needed` as ifthe ata is already created it will pass safely.

2.26 State Inconsistency in SwapCpAmm Leading to Data corruption

Severity: Medium

Status: Fixed

Location:

- solana/programs/claim_distribution/src/instructions/swap_cp_amm.rs
- #L222-L228@cb0123e5b0a0677e9a98689e307d2a3bf36478b9

Description: The SwapCpAmm instruction has two independent parameters that control different aspects ofthe swap but lack validation:

`base_token_mint`: determines where tokens are deposited `swap_gtoken`: determines which state counter gets incremented

A User can pass `base_token_mint=gtoken_mint` with `swap_gtoken=false`, causing gtokens to be received while the mtoken counter is incremented. This corrupts the `total_gtoken_swapped_in` and `total_mtoken_swapped_in` state variables.

Recommendation:

```
// Validate swap_gtoken matches base_token_mint
if swap_gtoken {
    require!(
        self.base_token_mint.key() == self.distributor_state.gtoken_mint,
        TokenDistributionError::InvalidBaseTokenForGtokenSwap
    );
} else {
    require!(
        self.base_token_mint.key() == self.distributor_state.mtoken_mint,
        TokenDistributionError::InvalidBaseTokenForMtokenSwap
    );
}
```

2.27 Claim_time being passed from outside in claim_fees

Severity: Info

Status: Fixed

Location:

- solana/programs/claim_distribution/src/instructions/claim_fees.rs
- #L152@cb0123e5b0a0677e9a98689e307d2a3bf36478b9

Description: In the ClaimFees function claim_time is being passed as a parameter to the function. There is currently no validation on the claim_time as it can be in back date too or an invalid one.

Recommendation: It is recommended to use the on-chain type via Solana Clock as its more robust and accurate than passing the claim_time as parameter from backend.

2.28 Admin and Oracle address should not be same

Severity: Info

Status: Fixed

Location:

- solana/programs/claim_distribution/src/instructions/setup_admin_config.rs
- #L35-L35@cb0123e5b0a0677e9a98689e307d2a3bf36478b9

Description: During the `SetupAdminConfig` we can set the oracle and feehandler. It's important that the oracle and admin should be different so we have the intented 2 step functionality. Currently there is no check which gurratees that the admin and Oracle are different.

Recommendation: It is recommended to put a check that admin.key and admin_config.oracle are not the same.

2.29 Arbitrary swap_time can be set during swapping

Severity: Info

Status: Fixed

Location:

- solana/programs/claim_distribution/src/instructions/swap_cp_amm.rs
- #L158-L158@cb0123e5b0a0677e9a98689e307d2a3bf36478b9

Description: During the swap process swap_time is being passed as a parameter to the process() function. This means arbitrary swap_time can be passed which would not reflect the actual swap time and leads to corrupted distributor_state.

Recommendation: It is recommended to use the Solana Clock for accurate time at which the swap is being done.

2.30 Redundant Initialization Check in Configure Distributor Function

Severity: Info

Status: Fixed

Location:

- programs/claim_distribution/src/instructions/configure_distributor.rs
- 108-110@cb0123e5b0a0677e9a98689e307d2a3bf36478b9

Description: The configure_distributor function contains an unreachable initialization check that cannot serve its intended purpose due to the account constraints. The distributor_state account is defined with the init constraint, which creates a new account. However, the function subsequently checks if distributor_state.initialized is true and returns an error if so.

```
if distributor_state.initialized {
            return Err(TokenDistributionError::DistributorAlreadyInitialized.into());
        }
```

This check is logically unreachable because: - The init constraint ensures a new account is created. - If the PDA already exists, the init constraint fails before the function logic executes. - New accounts have all fields initialized to their default values (false for booleans).

Recommendation: Remove the redundant initialization check entirely, as the init constraint already provides the necessary protection against duplicate account creation

2.31 Missing Validation on distributor_id during staking (ONCHAIN PROGRAM)

Severity: Info

Status: Fixed

Location:

- /claim_distribution/src/instructions/stake_gtokens.rs
- @cb0123e5b0a0677e9a98689e307d2a3bf36478b9a

Description: During the audit of sdk we found that the distributor_id being used in the system is arbitrary and can be anything which will be passed by the backend, Meaning it can be or not be sequential its totally upto the admin how he implements it.

In the stake_gtoken instruction *_distributor_id* is passed as parameter , In the stake_gtoken there is no check inplaced which ensures that the distributor against this *_distributor_id* is initialized or not , hence it is possible to pass wrong *_distributor_id* and tokens will be staked on it.

Recommendation: We would recommend following: 1. Implement a *_distributor_id* counter on-chain which can increment as the admin creates more distributor 2. Write proper documentation for implementation on the backend side so the developer have understanding to how *_distributor_id* should be handeled.

2.32 BigInt Conversion Failure with Decimal uiAmount Values

Severity: Info

Status: Fixed

Location:

- solana/src/utils/fetchCirculatingSupply.ts
- @cb0123e5b0a0677e9a98689e307d2a3bf36478b9a

Description: The fetchCirculatingSupply function in fetchCirculatingSupply.ts contains a issue that causes runtime failures when processing decimal uiAmount values. The function attempts to convert decimal numbers directly to BigInt, which only accepts integers, resulting in a RangeError.

```
1     if (supplyResponse.value.uiAmount !== null &amp;&amp; supplyResponse.value.uiAmount
          !== undefined) {
2       // @audit this will fail if uiAmount is in decimals as BigInt expects an integer not
            decimal.
3       console.log(&#x27;supplyResponse.value.uiAmount=&#x27;, supplyResponse.value.
            uiAmount);
4       return BigInt(supplyResponse.value.uiAmount);
5     }
```

Since the uiAmount can be in decimal format if we try to Convert it to BigInt, it will result in runtime error.

Recommendation: 1. Fix decimal handling by using Math.floor() before converting to BigInt, which normalizes the value and prevents runtime errors 2. Use uiAmountString for precision. The supplyResponse provides uiAmountString, which represents the token amount as a string and properly accounts for decimals.

2.33 Inconsistent Blockhash Handling in Transaction Builders

Severity: Info

Status: Acknowledged

Location:

- solana/src/utils/buildUnstakeTx.ts
- @cb0123e5b0a0677e9a98689e307d2a3bf36478b9a

Description: The codebase shows inconsistent approaches to handling blockhashes in transaction builders, with some functions using optional parameters while others directly fetch the latest blockhash. This inconsistency creates potential issues with transaction reliability and performance optimization.

```
const blockhash =
  params.blockhash || (await connection.getLatestBlockhash()).blockhash;
transaction.recentBlockhash = blockhash;
```

Recommendation: We recommend standardizing blockhash handling by always using the latest blockhash rather than accepting it from users. This approach ensures transactions aren't processed with stale or invalid blockhashes.

2.34 Unused Partner Parameter and User-Controlled Blockhash in Partner Metadata Creation

Severity: Info

Status: Fixed

Location:

- og-contracts/ solana/src/api/getCreatePartnerMetadataTx.ts
- @cb0123e5b0a0677e9a98689e307d2a3bf36478b9a

Description: Two issues identified in the PartnerMetadataParams interface:

1. **Unused Parameter:** The partner parameter is accepted but never utilized in the function logic, potentially causing partner metadata creation to fail or create invalid records
2. **User-Controlled Blockhash**: The optional blockhash parameter allows users to specify transaction blockhash instead offetching the latest, which can lead to transaction failures or manipulation

```
export interface PartnerMetadataParams { payer: Uint8Array; partner: Uint8Array; name: string;
 website: string; logo: string; feeClaimer: string; blockhash?: string; }
```

Recommendation: 1. **Fix Partner Parameter Usage:** Either utilize the partner parameter in the function implementation or remove it from the interface if not needed 2. **Auto-fetch Blockhash:** Remove the blockhash parameter and automatically fetch the latest blockhash within the function using

2.35 Passing of Arbitrary Referrer

Severity: High

Status: Acknowledged

Location:

- solana/programs/claim_distribution/src/instructions/stake_launch_option.rs
- @c7f4d979f36ccf6090b9f3bee78776fa82f597e0

Description: In the protocol during staking , the staker can provide a referrer which gets some percentage allocation. Currently the protocol blocks self referrer but the protocol does not checks ifthe referrer itself is a registered staker on the platform. Logically the users stakes to become stakers and then referrer other users to earn more.

Here Arbitrary user can be passed and he can become the referrer.

Recommendation: Since referrer_launch_option_escrow is desealized as LaunchOptionEscrow , there is has_staked field , which is set to true once the user has staked. Validate the referrer using this field.

2.36 Precision loss on referrer bonus calculation

Severity: Medium

Status: Acknowledged

Location:

- solana/programs/claim_distribution/src/instructions/stake_launch_option.rs
- @c7f4d979f36ccf6090b9f3bee78776fa82f597e0
- solana/programs/claim_distribution/src/instructions/unstake_launch_option.rs
- @c7f4d979f36ccf6090b9f3bee78776fa82f597e0

Description: The referrer bonus calculation uses integer division that truncates fractional parts, causing systematic precision loss where referrers receive fewer tokens than the intended percentage. The calculation (mtoken_amount * referrer_bonus) / 100 drops any remainder, resulting in referrers consistently receiving less than the specified 20% bonus

```
let referrer_bonus_u128 = (mtoken_amount as u128).checked_mul(self.launcher.referrer_bonus
     as u128).ok_or(TokenDistributionError::OverflowOrUnderflowOccurred)?
.checked_div(100) //  INTEGER DIVISION TRUNCATES REMAINDER
.ok_or(TokenDistributionError::OverflowOrUnderflowOccurred)?;
```

Example 1: Perfect Division - User stakes: 20 tokens
- Expected referrer bonus: 20 20 / 100 = 4.0 tokens - Actual referrer bonus: 4 tokens (no loss)

Example 2: Precision Loss - User stakes: 19 tokens - Expected referrer bonus: 19 20 / 100 = 3.8 tokens - Actual referrer bonus: 3 tokens (0.8 tokens lost)

Example 3: Maximum Loss Per Transaction - User stakes: 99 tokens - Expected referrer bonus: 99 20 / 100 = 19.8 tokens - Actual referrer bonus: 19 tokens (0.8 tokens lost)

Recommendation: Replace percentage-based calculation with basis points for higher precision.

2.37 Missing validation on vesting_speed_bonus

Severity: Low

Status: Fixed

Location:

- solana/programs/claim_distribution/src/instructions/set_vesting_speed.rs
- @c7f4d979f36ccf6090b9f3bee78776fa82f597e0

Description: The Set_vesting_speed function can be used to set the `vesting_speed_bonus` which essentially speedup the vesting duration. Currently in the function there are no bounds defined. Hence its possible to set it to any value which can be also greater then the launcher.vesting_duration as this can create issue during exercise_launch_option where `effective_duration` would become 0

```
let effective_duration = self.launcher.vesting_duration.saturating_sub(self.launcher.
    vesting_speed_bonus as u16) as u64;
``
```

```
pub fn process(
        &amp;mut self,
        _admin_key: Pubkey,
        _distributor_id: u64,
        vesting_speed_bonus: u8,
    ) -&gt; Result&lt;()&gt; {
        // @audit no upper bound for vesting_speed_bonus. The vesting speed bonus should
            not be greater than the launcher.vesting_duration
        require!(self.launcher.vesting_speed_bonus &lt; vesting_speed_bonus,
            TokenDistributionError::NewVestingSpeedTooLow);
        self.launcher.vesting_speed_bonus = vesting_speed_bonus;
        Ok(())
    }
```

Recommendation: It is recommended to put a check that the `launcher.vesting_speed_bonus` is not greater than `launcher.vesting_duration`

2.38 Overwriting of vesting_speed_bonus during vesting unitialization

Severity: Low

Status: Fixed

Location:

- `solana/programs/claim_distribution/src/instructions/initiate_vesting.rs`
- `@c7f4d979f36ccf6090b9f3bee78776fa82f597e0`

Description: When `InitiateVesting` is called by the launcher , the function sets the launcher.`vesting_speed_bonus` to zero. The `set_vesting_speed` can be called by launcher before vesting , hence ifthe `vesting_speed_bonus` is already set then it would be overwriten.

Recommendation: It is recommended that a check to be performed that ifthe vesting_speed_bonus is declared already then it should not be overwritten.

2.39 Resetting of Participant_count during Vesting

Severity: Low

Status: Fixed

Location:

- `solana/programs/claim_distribution/src/instructions/initiate_vesting.rs`
- `@c7f4d979f36ccf6090b9f3bee78776fa82f597e0`

Description: `launcher.participant_count` contains the number of users who staked their mtoken. When the Initiate_vesting is called then it's reset to 0. which means ifthe frontend is utilizing this field to show number of staked users it would show 0

Recommendation: It is recommended to not reset it to 0 or maintain the vested_users field which can show the number of users in vesting phase.

2.40 Missing Validation on staking and referral allocation amount

Severity: Low

Status: Acknowledged

Location:

- `solana/programs/claim_distribution/src/instructions/initiate_vesting.rs`
- `@c7f4d979f36ccf6090b9f3bee78776fa82f597e0`

Description: During InitiateVesting staking_allocation_gtoken_amount and referral_allocation_gtoken_amount are passed which are transfered to `launcher_gtoken_ata`

There is currently no check in placed which make sures that the `referral_allocation_gtoken_amount` is >= sum of (`launcher.total_referrer_launch_option_amount` & `launcher.total_referred_launch_option_amount`) and the `staking_allocation_gtoken_amount` be greater= `launcher.total_staking_launch_option_amount`

Recommendation: It is recommended to add the validation on the allocation amounts being passed in the function.

2.41 Unnecessary Initialization of Referrer during Unstake

Severity: Info

Status: Fixed

Location:

- `solana/programs/claim_distribution/src/instructions/unstake_launch_option.rs`
- `@c7f4d979f36ccf6090b9f3bee78776fa82f597e0`

Description: When the `unstake_launch_options` is called `referrer_launch_option_escrow` is also passed in the accounts but is it initialized via `init_if_needed`. Logically it should only be initialzed via the staking is done , upon unstaking it should already exist.

```
#[account(
        init_if_needed,
        payer = participant,
        space = 8 + LaunchOptionEscrow::LEN,
        seeds = [
            LaunchOptionEscrow::SEED_PREFIX.as_bytes(),
            distributor.key().as_ref(),
            referrer.key().as_ref()
        ],
        bump
    )]
```

Recommendation: It is recommended to derive the `referrer_launch_option_escrow` via the seeds and do not initialized it in the `unstake_launch_option`.

2.42 Redundant distribution_pool Account in ProcessClaim Instruction

Severity: Info

Status: Fixed

Location:

- `programs/claim_distribution/src/instruction/process_claim.rs`

Description: The ProcessClaim instruction includes a distribution_pool account, which is redundant because the distributor account already represents the same PDA. The PDA derivation of distributor and distribution_pool uses identical seeds, making distribution_pool unnecessary. Removing distribution_pool does not affect functionality, as distributor can be used in its place.

Recommendation Remove the distribution_pool account from ProcessClaim and use distributor directly wherever distribution_pool is referenced.

2.43 Wrong Error Emission

Severity: Info

Status: Fixed

Location:

- programs/claim_distribution/src/states/distributor.rs

Description: When the distribute_claim is called and user_claim.amount is passed as zero, the claimAlreadyProcessed error is emitted.

```
if user_claim.amount == 0 {
    return Err(TokenDistributionError::ClaimAlreadyProcessed.into());
}
```

Whereas InvalidClaimAmount should be emitted here. One thing to note is this error can be considered valid if in the distribute_claim the user_claim.amount was set to 0 after distribution, which is currently not being done

2.44 Use of Raw u8 for Phases Instead of Enum

Severity: Info

Status: Fixed

Location:

src/state/launcher.rs

Description: The Launcher account defines the current protocol phase as a raw u8 with hardcoded numeric values:

```
// phases:
// 0 =&gt; uninitialized
// 1 =&gt; staking phase
// 2 =&gt; launched
// 3 =&gt; vesting phase
pub phase: u8;
```

Proof of Concept: TBC

Recommendation: Replace the raw u8 with a Rust enum that explicitly defines valid phases. For example:

```
#[derive(AnchorSerialize, AnchorDeserialize, Clone, Copy, PartialEq, Eq)]
pub enum Phase {
    Uninitialized,
    Staking,
    Launched,
    Vesting,
}
pub struct Launcher {
```

This enforces type safety, improves readability, and prevents invalid values from being set.

2.45 Referrer Accounts Should Be Optional to Reduce Unnecessary Account Handling

Severity: Info

Status: Acknowledged

Location:

src/instruction/stake_launch_option.rs src/instruction/unstake_launch_option.rs

Description: In both StakeLaunchOption and UnstakeLaunchOption instruction contexts, the referrer and its associated referrer_launch_option_escrow accounts are always required, even when a participant does not have a referrer.

```
/// CHECK: validated below
pub referrer: UncheckedAccount&lt;&#x27;info&gt;,
#[account(
mut,
seeds = [
LaunchOptionEscrow::SEED_PREFIX.as_bytes(),
distributor.key().as_ref(),
mtoken_mint.key().as_ref(),
referrer.key().as_ref()
],
bump
)]
pub referrer_launch_option_escrow: Account&lt;&#x27;info, LaunchOptionEscrow&gt;,
```

Ifthe participant's referrer is set to the default public key, these accounts serve no purpose and should not be required. Keeping them mandatory introduces unnecessary account creation and loading overhead in the Solana runtime.

Proof of Concept: TBC

Recommendation: Make the referrer and referrer_launch_option_escrow accounts optional in both StakeLaunchOption and UnstakeLaunchOption .

- If no referrer is provided (default pubkey), skip creation/validation.
- If a valid referrer is provided, enforce the presence ofthese accounts.

This ensures accounts are only created and loaded when necessary, improving efficiency and reducing runtime overhead.

Developer Response Will leave as such to keep logic simpler.

2.46 Reservations on protocol_fee_recipient

Severity: Info

Status: Closed

Location:

TBC

Description: Protocol_fee_recipient is an important parameter in the protocol as this address will be collecting the fees generated in the protocol. In the current implementation protocol_fee_recipient is set by the creator calling the configure_curve(). Which opens rooms for manipulation as the creator can set any address here and collect fees.

During the swap function Protocol_fee_recipient is also passed. Since it is defined in the curve configuration , it should be referenced from it.

Proof of Concept: TBC

Recommendation: It is recommended that protocol_fee_recipient be defined in the global config which is set and initialized by the admin and while creating the curve configuration , protocol_fee_recipient should be referenced from the global_config , not from the creator's input parameter.

2.47 No validation on the distributor_state reward_amount

Severity: Info

Status: Closed

Location:

solana/programs/claim_distribution/src/instructions/configure_distributor.rs#L126-L127

Description: When the distributor_state is initialized two fields are set , where gtoken_reward_amount is passed as parameter and set as follows:

```
distributor_state.initial_gtoken_reward_amount = gtoken_reward_amount;
distributor_state.total_gtoken_grants = gtoken_reward_amount;
```

This reward amount should not be greater than the gtokenStashPct which is defined as

```
gtokenStashPct = liquidityPct + rewardPct + mtokenHolderPct + treasuryPct +
    graduationRewardPct;
```

Proof of Concept: TBC

Recommendation: It is recommended to enforce a check in the program that it doesn't exceed the gtokenStashPct

Developer Response reward amount was removed from the distributor account and is instead computed on the client using other existing distributor properties (no new property was added)

2.48 Comprehensive Lack of Input Parameter Validation

Severity: Info

Status: Closed

Location:

TBC

Description: The codebase demonstrates a systematic absence of input parameter validation across multiple functions, creating numerous attack vectors and potential system failures. This represents a fundamental security flaw that could lead to:

- Economic exploits through malformed parameters
- System crashes from invalid inputs
- Resource exhaustion attacks
- State corruption through edge case inputs

Affected Components

- All Functions in solana/src/api/
- Transaction builders in solana/src/utils/
- Core protocol functions
- User-facing interfaces

Following are some instances for reference from the SDK.

```
// src/api/getBondingCurveSwapQuote.ts
export async function getSwapQuote(
  bondingCurveClient: DynamicBondingCurveClient,
  bondingCurvePool: string,
  slippageBps: number,  // NO VALIDATION
  amountIn: bigint,
  swapBaseForQuote: boolean,
): Promise&lt;BondingCurveQuote&gt; {
  // @audit put validation on slippageBps, should be between 0 and 10000.
  // src/utils/buildStakeTx.ts
export async function buildStakeTx(params: StakeParams): Promise&lt;Transaction&gt; {
  const ts = new BN(params.timestamp);
  // @audit the timestamp should have some validation.

// src/api/getLaunchBondingCurveTxs.ts
export async function getLaunchBondingCurveTxs(params: LaunchBondingCurveParams) {
  // @audit Validation on the LaunchBondingCurveParams params is missing.

// Multiple API functions accept string addresses without validation
export async function getDistributorStatus(
  distributorPda: string,        // NO BASE58 VALIDATION
  distributorStatePda: string,   // NO BASE58 VALIDATION
  gtoken: string,               // NO MINT VALIDATION
  mtoken: string,               // NO MINT VALIDATION
  lpPool: string,               // NO ADDRESS VALIDATION
): Promise&lt;DistributorStatus&gt;;
```

Proof of Concept: TBC

Recommendation: it is recommended to put input validation across all the function's params. Since OG's implementation is a hybrid implementation with off-chain and on-chain components , its important to put validations in the functions.

Developer Response Left unaddressed, not sure it is important. We do not see other third party public Solana SDKs perform this type of validation. Could you provide examples of potential issues arising from the lack of validation?

Auditor Response Validations are important in the sense that they would ensure that correct and valid type oftransaction is formated and sent onchain.

Disclaimer:

The smart contracts provided by the client with the purpose of security review have been thoroughly analyzed in compliance with the industrial best practices till date w.r.t. Smart Contract Weakness Classification (SWC) and Cybersecurity Vulnerabilities in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment ofthe project or team, and they do not in any way guarantee the security ofthe particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics ofthe token (if any), its sale, or any other aspect ofthe project that contributes to the protocol's public marketing.

Crypto assets/ tokens are the results ofthe emerging blockchain technology in the domain of decentralized finance and they carry with them high levels oftechnical risk and uncertainty. No report provides any warranty or representation to any third-party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the reports in any way, including to make any decisions to buy or sell any token, product, service, or asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement ofthis project or team, and is not a guarantee as to the absolute security ofthe project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review ofthe programmable code and only the programmable code, we note, as being within the scope of our review within this report. The smart contract programming language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond the programming language's compiler scope that could present security risks.

This security review cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements ofthe contract. While BlockApex has done their best in conducting the analysis and producing this report, it is important to note that one should not rely on this report only - we recommend proceeding with several independent code security reviews and a public bug bounty program to ensure the security of smart contracts