Doryoku Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Farouk

Kose

July 22, 2025

Contents

1 About Cyfrin **2**

2 Disclaimer **2**

3 Risk Classification **2**

4 Protocol Summary **2**

5 Audit Scope **2**

6 Executive Summary **3**

7 Findings **5**

7.1 High Risk . . . 5

7.1.1 Multiple `vault_gkhan_account` Can be Used Resulting in DoS on `complete_vest` . . . 5

7.1.2 Hard-coded 165-byte vault account is too small for Raydium Token-2022 mints with extensions 5

7.2 Low Risk . . . 7

7.2.1 Hard-coded `9` decimals for xBELO mint can mismatch underlying BELO mint . . . 7

7.2.2 Improper Rounding in Burn Logic Lets Users Avoid Penalty . . . 7

7.2.3 Vesting Durations Might be out of Bounds Which Allows Immediate Withdrawals . . . 7

7.2.4 Steep Stepwise Jumps in Burn Percentages . . . 8

7.3 Informational . . . 9

7.3.1 Consider Two-step Ownership Transfer . . . 9

7.3.2 Unneccessary Rent Account Usage . . . 9

7.3.3 Vesting Parameter Updates are Not Constrained Sufficiently . . . 9

7.3.4 Initializer Can Be Front-Run to Gain Control of the Program . . . 9

7.3.5 Unnecessary Utility Functions in the `xbelo` Program . . . 10

7.3.6 Redundant Instruction Attributes in `Initialize` and `Vest` Instructions . . . 10

7.3.7 Same Conditions Revalidated in Instruction Logic and Account Contexts . . . 10

7.3.8 Missing Event Emission for Admin Transfers . . . 10

7.3.9 StakeCounter Struct Declared but Never Utilized . . . 11

7.3.10 Unsafe Arithmetic in `total_position_staked` Updates . . . 11

7.3.11 Unutilized `position_collection` Field in Events . . . 11

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

The xBELO contract converts liquid BELO into a frozen receipt token that cannot be transferred. Users deposit BELO into a vault owned by the contract's state PDA, receive an equal amount of xBELO, and later choose a vesting period that burns a percentage immediately while locking the remainder until maturity. Once the lock ends, the user retrieves the vested portion from the vault and the contract burns the remainder.

The main security expectations are that the vault balance always equals the circulating xBELO supply, that temporary thaws for minting or burning never open a transferable window, and that admin updates to vesting parameters cannot introduce insolvency. The CLMM LP Farming contract allows liquidity providers to lock Raydium concentrated-liquidity NFTs for fixed terms of one, three, six, or twelve months. Each position is transferred into a program-owned vault account and later returned to the user when the term ends, while off-chain logic tallies staking rewards from emitted events.

Correctness hinges on creating vault token accounts that match the ever-growing Token-2022 extension size, deriving vault and authority PDAs deterministically to prevent spoofing, maintaining accurate counts of staked positions, and ensuring that the pause and admin transfer mechanisms cannot be abused. With these focal areas addressed, the combined system is positioned to deliver non-transferable vesting receipts and NFT farming rewards without exposing user funds to unintended transfer paths or accounting mismatches.

5 Audit Scope

Cyfrin conducted an audit of the xBELO and BELO CLMM LP Farming programs based on the code present at commit hashes f37639c, 7601b7b with scope:

```
belo_lp_farming/programs/clmm_lp_farming/src/lib.rs
xbelo/programs/xbelo/src/lib.rs
```

6 Executive Summary

Over the course of 3 days, the Cyfrin team conducted an audit on the Doryoku smart contracts provided by Belobaba. In this period, a total of 17 issues were found.

Our review covered the xBELO vesting contract and the CLMM LP Farming contract, examining every instruction, the associated PDA layouts, and their interaction with SPL-Token 2022 semantics in a local validator. While the architecture is conceptually simple non-transferable receipt tokens for BELO deposits on one side and NFT staking for Raydium liquidity positions on the other, the implementation leaves several critical paths to insolvency and denial-of-service.

The xBELO program will accept any token account owned by its state PDA as the vault, so an attacker can leverage that to lock the funds of other users. The farming program allocates only 165 bytes for vault token accounts, a size that predates token-extension support; a Raydium position NFT that carries extra extensions will always fail to stake. Further issues include a hard-coded nine-decimal assumption that misprices the entire supply whenever the BELO mint differs in precision.

On the positive side, both programs benefit from clear event logging, a deterministic PDA pattern, and a straightforward admin separation, which together make remediation tractable. By adopting deterministic vault seeds, dynamically sizing token accounts, enforcing decimal parity, and hardening parameter validation, the protocol can close the most dangerous gaps and align with Solana security best practice.

Summary

Project Name	Doryoku
Repository	xbelo
Commit	7601b7bc4c05...
Repository 2	belo_clmm_lp_farming
Commit	f37639cd7586...
Audit Timeline	Jul 4th - Jul 8th, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	2
Medium Risk	0
Low Risk	4
Informational	11
Gas Optimizations	0
Total Issues	17

Summary of Findings

[H-1] Multiple `vault_gkhan_account` Can be Used Resulting in DoS on `complete_vest`	Resolved
[H-2] Hard-coded 165-byte vault account is too small for Raydium Token-2022 mints with extensions	Resolved
[L-1] Hard-coded `9` decimals for xBELO mint can mismatch underlying BELO mint	Resolved
[L-2] Improper Rounding in Burn Logic Lets Users Avoid Penalty	Resolved
[L-3] Vesting Durations Might be out of Bounds Which Allows Immediate Withdrawals	Resolved
[L-4] Steep Stepwise Jumps in Burn Percentages	Resolved
[I-01] Consider Two-step Ownership Transfer	Resolved
[I-02] Unneccessary Rent Account Usage	Resolved
[I-03] Vesting Parameter Updates are Not Constrained Sufficiently	Resolved
[I-04] Initializer Can Be Front-Run to Gain Control of the Program	Resolved
[I-05] Unnecessary Utility Functions in the `xbelo` Program	Resolved
[I-06] Redundant Instruction Attributes in `Initialize` and `Vest` Instructions	Resolved
[I-07] Same Conditions Revalidated in Instruction Logic and Account Contexts	Resolved
[I-08] Missing Event Emission for Admin Transfers	Resolved
[I-09] StakeCounter Struct Declared but Never Utilized	Resolved
[I-10] Unsafe Arithmetic in `total_position_staked` Updates	Resolved
[I-11] Unutilized `position_collection` Field in Events	Resolved

7 Findings

7.1 High Risk

7.1.1 Multiple `vault_gkhan_account` Can be Used Resulting in DoS on `complete_vest`

Description: The program validates `vault_gkhan_account` only by checking that

- `vault_gkhan_account.owner == state.key()` and
- `vault_gkhan_account.mint == state.belo_mint`

It **does not** verify that the account is the canonical vault ATA created via the Associated Token Account (ATA) program or a PDA derived with hard-coded seeds. Consequently, *any* SPL token account whose owner field is set to the `state` PDA passes the constraints—even if it was created by an attacker and is fully under their close-authority control.

1. **Alice** makes a normal deposit
   - Deposits `100 BELO` into the official vault ATA (the one originally initialized).
   - Receives `100 xBELO`.
2. **Bob** crafts a phantom vault
   - Creates a new token account `phantom_vault` with `owner = state_pda`, `mint = belo_mint`, and sets himself as `close_authority`.
   - Calls `deposit`, supplying `phantom_vault` as `vault_gkhan_account`, depositing `50 BELO`.
   - Receives `50 xBELO`.
3. Bob later calls `complete_vest` (after his vest duration) but passes the **official** vault ATA.
   - `50 BELO` are transferred out of the official vault and his `burn_amount` is burned there.
4. When Alice's vest completes, the official vault holds only `50 BELO`, so her transaction fails, effectively stealing locking her funds.

Impact: Attackers can create "phantom" vaults, deposit through them, and later redeem from the *real* vault. This drains the legitimate vault balance, rendering the protocol insolvent and leaving honest users unable to claim their vested tokens.

Recommended Mitigation:

- **Bind the vault address deterministically.** Derive the vault PDA with fixed seeds, e.g.

```
let (vault_key, _bump) = Pubkey::find_program_address(
    &[b"vault", state.belo_mint.as_ref()],
    program_id,
);
require!(vault_gkhan_account.key() == vault_key, ErrorCode::InvalidTokenAccount);
```

- **or** strictly enforce ATA-derivation: require `vault_gkhan_account.key() == get_associated_token_address(&state.key(), &state.belo_mint)`.

Doryoku: Fixed in f527d44 and 034eaac.

Cyfrin: Verified.

7.1.2 Hard-coded 165-byte vault account is too small for Raydium Token-2022 mints with extensions

Description: `stake_clmm_position` creates the vault token account for the raydium position mint with a size 165 Bytes

```rust
        // Create vault token account if it doesn't exist
        let vault_account_info = ctx.accounts.vault_position_token_account.to_account_info();
        if vault_account_info.data_is_empty() {
            // Create the vault token account
            let vault_account_space = 165; // Standard token account size
            let vault_account_lamports = ctx.accounts.rent.minimum_balance(vault_account_space);

            let farm_key = ctx.accounts.farm.key();
            let position_mint_key = ctx.accounts.position_mint.key();
            let vault_seeds: &[&[u8]] = &[
                VAULT_SEED,
                farm_key.as_ref(),
                position_mint_key.as_ref(),
                &[ctx.bumps.vault_position_token_account],
            ];
            let vault_signer_seeds = &[vault_seeds];

            // Create account
            anchor_lang::system_program::create_account(
                CpiContext::new_with_signer(
                    ctx.accounts.system_program.to_account_info(),
                    anchor_lang::system_program::CreateAccount {
                        from: ctx.accounts.user.to_account_info(),
                        to: vault_account_info.clone(),
                    },
                    vault_signer_seeds,
                ),
                vault_account_lamports,
                vault_account_space as u64,
                &ctx.accounts.token_program.key(),
            )?;

            // Initialize the token account
            anchor_spl::token_interface::initialize_account3(
                CpiContext::new(
                    ctx.accounts.token_program.to_account_info(),
                    InitializeAccount3 {
                        account: vault_account_info.clone(),
                        mint: ctx.accounts.position_mint.to_account_info(),
                        authority: ctx.accounts.vault_authority.to_account_info(),
                    },
                ),
            )?;
        }
```

The constant **165 bytes** is the legacy SPL-Token size. Raydium CLMM position NFTs are **Token-2022** mints that append several on-chain extensions (e.g., `MetadataPointer`, `MintCloseAuthority`).

Allocating only 165 bytes causes `initialize_account3` to fail with InvalidAccountData.

Impact: Users cannot stake their legitimate Raydium positions, the transaction aborts, blocking the farming pool.

Recommended Mitigation: It is recommended to dynamically compute required space based on the mint's extension, here is an example from Raydium's codebase

Doryoku: Fixed in fad227e.

Cyfrin: Verified.

7.2 Low Risk

7.2.1 Hard-coded `9` decimals for xBELO mint can mismatch underlying BELO mint

Description: During `initialize` the program creates the xBELO receipt token like this:

```
#[account(
    init, payer = admin,
    mint::decimals = 9,             // ← hard-coded
    mint::authority = state,
    mint::freeze_authority = state
)]
pub xgkhan_mint: Account<'info, Mint>,
```

The number of decimals is fixed at **9** instead of copying `belo_mint.decimals`. If the canonical BELO mint was deployed with any other precision (e.g., 6 or 18), the 1 : 1 accounting assumption between BELO locked in the vault and xBELO outstanding is violated.

Impact: This can break the 1 : 1 accounting assumption between BELO and xBELO.

Recommended Mitigation: * At `initialize`, **read the decimals of** `gkhan_mint` **and reuse them**:

```
      #[account(
          init, payer=admin,
-         mint::decimals=9,
+         mint::decimals=gkhan_mint.decimals,
          mint::authority=state,
          mint::freeze_authority=state
      )]
      pub xgkhan_mint: Account<'info, Mint>,
```

Doryoku: Fixed in 74212b7.

Cyfrin: Verified.

7.2.2 Improper Rounding in Burn Logic Lets Users Avoid Penalty

Description: Vesting can happen with any positive value, it doesn't have a minimum value requirement as long as it is positive. The amount that will be burned is currently configured to be in between 2% and 50% according to vesting time. `burn_amount` is calculated as:

```
        let burn_amount = amount
            .saturating_mul(burn_pct)
            .checked_div(100)
            .ok_or(ErrorCode::ArithmeticError)?;
```

Here, `amount` represents the exact amount user is vesting, while `burn_pct` represents the burn percentage, and it's value will always be in between **2** and **50**. Considering the burned amount will be calculated with `amount` multiplied by a value that is lower than **100**, and then it will be divided to **100**, this value will be rounding down. While it is always optimal to round up fee/burn related values and this implementation breaks that, it also creates an opportunity to game the program to get rid of burning altogether.

Here is an example scenario that shows the impact:

- Alice vests **49** token with maximum vest time, hence burn_pct became **2**.
- This results with burn_amount that is equal to **0**.
- Alice vested without burning and can repeat this without any limit.

Impact: While the material impact is very limited, it let users vest without burning anything. Calculation also leads to underestimation of burn amount by **1** for every vest.

Recommended Mitigation: Consider adding **1** to the `burn_amount` after the division is performed:

```
        let burn_amount: u64 = amount
            .saturating_mul(burn_pct)
+           .checked_add(99)
+           .ok_or(ErrorCode::ArithmeticError)?;
            .checked_div(100)
            .ok_or(ErrorCode::ArithmeticError)?;
```

Doryoku: Fixed in 2a4e99c.

Cyfrin: Verified.

7.2.3 Vesting Durations Might be out of Bounds Which Allows Immediate Withdrawals

Description: Vesting has a limit for minimum and maximum duration it can have. Within initial configuration, minimum duration is *10 days* while the maximum is *180 days*.

However `vest` doesn't confirm vesting duration is within bounds (specifically, relevant check is commented out):

```
        // Validate vest duration is within bounds
        // require!(dur_secs >= min_secs, ErrorCode::VestDurationTooShort);
        // require!(dur_secs <= max_secs, ErrorCode::VestDurationTooLong);
```

Because of this, user's can vest less than minimum duration and be treated as if they have vested the minimum duration. Particularly it is possible to vest for *1 second* and then withdraw as if the amount is vested for *10 days*.

Impact: Vesting duration bounds are not protected, users can bypass minimum duration and perform back to back deposit and withdraw, leading to receiving funds without vesting.

Recommended Mitigation: Uncomment checks for vesting duration.

Doryoku: Fixed in d16e369.

Cyfrin: Verified.

7.2.4 Steep Stepwise Jumps in Burn Percentages

Description: `burn_pct` (burn percentage) does not utilize any precision multiplier and its value can be minimum **2** and maximum **50** with initial configuration. Calculation for `burn_pct` as follows:

```
            (state.min_burn_pct as u64)
                .saturating_sub(((over_min * burn_range) / time_range) as u64)
```

Changing these variables with their initial configuration, we can approximate the calculation with this formula:

```
    50 - 48 * ( (vest_duration - 10 days ) / 170 days )
```

This formula consists of steep stepwise jumps with every jump occurring approximately in **3.5 day** intervals.

Hence, the percentages of tokens that will be burned will be the same for a user who vests for *10 days* and the user who vests for *13.4 days*.

Recommended Mitigation: Introduce basis points (bips) or another high-precision scaler to burning percentage calculation to smoothen stepwise jumps. Example implementation with basis points:

```
-        state.min_burn_pct = 50; // 50% @ min
-        state.max_burn_pct = 2; //  2% @ max
+        state.min_burn_pct = 5_000; // 50% in basis points (bips) @ min
+        state.max_burn_pct = 200; //  2% in basis points (bips) @ max
```

```
        let burn_amount = amount
            .saturating_mul(burn_pct)
-            .checked_div(100)
```

```
+            .checked_div(10_000) // %100 in basis points (bips)
            .ok_or(ErrorCode::ArithmeticError)?;
```

Doryoku: Fixed in 034eaac.

Cyfrin: Verified.

7.3 Informational

7.3.1 Consider Two-step Ownership Transfer

Description: In both `xbelo` and `clmm_lp_farming` programs, ownership is transferred within one function call, namely `transfer_admin`. This pattern comes with it risks, any mistake can lead to losing ability to update vesting parameters for `xbelo` program, and losing pausability functionality for `clmm_lp_farming` program considering these functions are only callable by the admin.

Recommended Mitigation: Consider implementing two step ownership transfer for safer ownership transfer. This requires two functions: 1- Assigning the next admin: Current admin proposes a `new_admin`. 2- Accepting ownership: `new_admin` accepts ownership.

Doryoku: Fixed in 41676f1 and fad227e.

Cyfrin: Verified.

7.3.2 Unneccessary Rent Account Usage

Description: Within the `Initialize` instruction context (Accounts struct) of `xbelo` program, system account `rent` is provided.

However `rent` account is neither utilized nor necessary for any action in initialization.

Recommended Mitigation: Consider removing `rent` system account.

Doryoku: Fixed in 41676f1.

Cyfrin: Verified.

7.3.3 Vesting Parameter Updates are Not Constrained Sufficiently

Description: The `update_vest_params` function in `xbelo` program, allows admin to update minimum-maximum vesting durations, and minimum-maximum burn percentages. However it doesn't contrained enough to prevent wrong configurations between interconnected variables. Hence it is possible to provide values that will break functionality of functions. For example, `vest_min` can be bigger than `vest_max` which would prevent `vesting` completely considering there will be no valid `vest_duration` that can bypass duration checks.

Recommended Mitigation: Consider adding following checks:

```
require!(new_vest_max > new_vest_min, ErrorCode::InvalidParams);
require!(new_min_burn_pct > new_max_burn_pct, ErrorCode::InvalidParams);
```

Doryoku: Fixed in 4ecb4a0.

Cyfrin: Verified.

7.3.4 Initializer Can Be Front-Run to Gain Control of the Program

Description: The `initialize` function from the `xbelo` program, and the `initialize_farm` function from the `clmm_lp_farm` program can be called by anyone after deployment. Considering these functions sets the admins for programs and initializes critical values, malicious executions of these can gave away the control of the program to the caller.

Recommended Mitigation: Make sure the deployment and initialization of the program occur in the same transaction, or add the admin address in the `Farm` seeds.

Doryoku: Fixed in 41676f1 and fad227e.

Cyfrin: Verified.

7.3.5 Unnecessary Utility Functions in the `xbelo` Program

Description: `xgkhan` program has some functions that are neither utilized nor necessary:

- `transfer_hook()`: `xGKHAN` is a regular SPL token, not a TOKEN2022 token with extensions. Hence there is no need for a hook function.
- `transfer_xgkhan()`: This custom transfer function, which always fails, does not provide any functionality. Since `xGKHAN` tokens became frozen after they are minted to user, transfers will fail at the token program level. This function won't be utilized during token transfers.
- `verify_non_transferable()`: SPL token accounts already have built-in freeze status checking functionality via `TokenAccount::is_frozen` function. Both of these functions checks the same state, hence it is not necessary to provide a functionality that is already available in the token program itself.

Recommended Mitigation: Consider removing aforementioned functions and their instruction context (Accounts struct).

Doryoku: Fixed in 41676f1.

Cyfrin: Verified.

7.3.6 Redundant Instruction Attributes in `Initialize` and `Vest` Instructions

Description: In `xbelo` program's `Initialize` and `Vest` account structs, `#[instruction()]` and `#[instruction(amount: u64, vest_duration: i64)]` attributes are provided respectively. This attribute declares instruction parameters that needs to be accessed during account validation. Both of these can be safely removed since the first one declares no parameters and none are needed, and the latter one declares parameters(`amount` and `vest_duration`) that are not used in account validation constraints.

Recommended Mitigation: Consider removing `#[instruction(...)]` attributes from `Initialize` and `Vest` account structs since they are not necessary.

Doryoku: Fixed in 41676f1.

Cyfrin: Verified.

7.3.7 Same Conditions Revalidated in Instruction Logic and Account Contexts

Description: In the `clmm_lp_farming` program, certain validations are performed twice: once in functions, and once in account validations:

- `user_position_token_account.amount == 1` checked both in `stake_clmm_position` function and `StakeCLMMPosition` struct,
- `!stake.claimed` checked both in `withdraw_clmm_position` function and `WithdrawCLMMPosition` struct

Recommended Mitigation: Consider removing one of these validations to prevent code duplication.

Doryoku: Fixed in fad227e.

Cyfrin: Verified.

7.3.8 Missing Event Emission for Admin Transfers

Description: Function `transfer_admin` from `clmm_lp_farming` program changes an important state of the program but does not emit any events.

Recommended Mitigation: Consider emitting events from this function for better off-chain tracking and transparency.

Doryoku: Fixed in fad227e and 41676f1.

Cyfrin: Verified.

7.3.9 StakeCounter Struct Declared but Never Utilized

Description: The `clmm_lp_farming` program has a struct `StakeCounter` that includes a field `count`. However this struct is not utilized anywhere in the program:

```
#[account]
pub struct StakeCounter {
    pub count: u64,
}

impl StakeCounter {
    pub const LEN: usize = 8;
}
```

Recommended Mitigation: Consider removing `StakeCounter` struct with its `LEN` implementation.

Doryoku: Fixed in fad227e.

Cyfrin: Verified.

7.3.10 Unsafe Arithmetic in `total_position_staked` Updates

Description: `clmm_lp_farming` program increases and decreases `total_positions_staked` by **1** in `stake_clmm_position` and `withdraw_clmm_position` functions respectively:

```
        ctx.accounts.farm.total_positions_staked += 1;
```

```
        ctx.accounts.farm.total_positions_staked -= 1;
```

Considering rust will wrap around and will allow overflows in release mode if not explicitly stated otherwise (by adding `overflow-checks = true` to `Cargo.toml`), these operations are not safe.

Recommended Mitigation: Consider using `checked_add()` and `checked_sub` functions to prevent overflows.

Doryoku: Fixed in fad227e.

Cyfrin: Verified.

7.3.11 Unutilized `position_collection` Field in Events

Description: When a user stakes and withdraws, `CLMMPositionStaked` and `CLMMPositionWithdrawn` events are emitted respectively. Both of these events include a `Pubkey` field named `position_collection`. However in both actions, that field is emitted as `Pubkey::default()`:

```
        emit!(CLMMPositionStaked {
            user: user_stake.user,
            position_mint: ctx.accounts.position_mint.key(),
            clmm_pool: ctx.accounts.farm.clmm_pool,
            duration_months,
            start_time: user_stake.start_ts,
            end_time: user_stake.start_ts + user_stake.duration,
            position_collection: Pubkey::default(),
        });
```

Hence it is not used.

Recommended Mitigation: Consider removing the `position_collection` field from `CLMMPositionStaked` and `CLMMPositionWithdrawn` events if it has not have a functionality in off-chain system.

Doryoku: Fixed in fad227e.

Cyfrin: Verified.