Accretion

SECURITY ASSESSMENT

Provided by Accretion Labs Pte Ltd. for MetaDAO
October 02, 2025
A25MET3

MetaDAO

AUDITORS

Role	Name
Lead Auditor	Robert Reith (robert@accretion.xyz)
Auditor	David Ratiney (david@accretion.xyz)

CLIENT

MetaDAO (https://metadao.fi) is the first blockchain implementation of Futarchy, which is a DAO system offering an alternative to traditional voting DAOs. In the MetaDAO, users don't vote on proposals, instead they trade them. The MetaDAO implements and maintains Futarchy and realated technology for other DAOs. They engaged Accretion to conduct an assessment of a new program which allows tokens to be unlocked upon reaching specific price targets.

ENGAGEMENT TIMELINE

19 Sep — Project Kickoff
Initial planning and scope definition

20 Sep — Assessment Begins
Security review and testing phase

21 Sep — Review Fixes
Security recommendations are given and implemented

02 Oct — Project Completion
Report delivery and on-chain confirmation

AUDITED CODE

Program 1

ProgramID: pbPPQH7jyKoSLu8QYs3rSY3YkDRXEBojKbTgnUg7NDS

Repository: https://github.com/metaDAOproject/programs/tree/develop/programs/price_based_performance_package

ASSESSMENT

The security assessment of the MetaDAO's performance package program revealed a simple program implementing price-based token unlocks. The program is well built and we only had minor remarks and one significant finding.

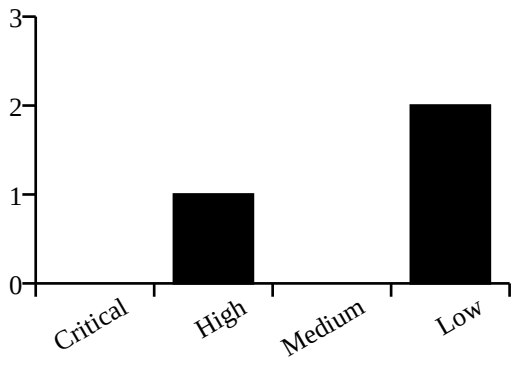
CODE ASSESSMENT

The code was well written and kept simple, integrating well into the existing suite of MetaDAO programs. It's written in Anchor, and includes as testsuite covering the main functionality and failure cases.

KEY FINDINGS

Besides our minor remarks, one high severity issue was found in the code which implements the price based unlock feature within a token launch. It allowed users to set an arbitrary starting price when the launch authority failed to complete the launch on time.

SEVERITY DISTRIBUTION



ENGAGEMENT SCOPE

The scope of this security assessment was a full review of the following items:

Item 1: Price Based Performance Package

Link: https://github.com/metaDAOproject/programs/tree/develop/programs/price_based_performance_package
Commit: 5d270a1bcf88b0b56adec5a72a4ff56a18dd9db7
Program ID: pbPPQH7jyKoSLu8QYs3rSY3YkDRXEBojKbTgnUg7NDS
Audit Result:

- **Audited Commit:** 0847ed51a956f607cca89c7020bb617b0c686f19
- **Build Hash:** 8b1143ed77551ee1e39b8e92b3bc4cd189f747bd82c43d7d3c51d4d6fc2e4869
- **Status:** verified
- **Comment:** On-chain program matches local build

ISSUES SUMMARY

ID	TITLE	SEVERITY	STATUS
ACC-H1	In complete launch, anyone can set an arbitrary price after two days	high	fixed
ACC-L1	Potential TWAP reset griefing attack possible	low	wontfix
ACC-L2	`ChangeRequest` can become unclosable	low	fixed
ACC-I1	Lack of validation of `twap_length_seconds`	info	fixed

DETAILED ISSUES

ID	ACC-H1
Title	In complete launch, anyone can set an arbitrary price after two days
Severity	high
Status	fixed

Description

We found that in `complete_launch`, which is out-of-scope of this audit, but contains the logic which invokes the program in-scope, contains a serious logic flaw. The function is used to finalize a launch and call external programs to do so, for example initializing a DAO from a launch, creating a performance package, and creating the liquidity pool. The instruction lets the user provide a `final_raise_amount` arg instead of using the real raise amount. While this feature is questionable by itself, it was intended to only be usable by the launch authority, and it requires that the provided raise amount is larger than or equal to the launch's minimum raise amount. This amount is then used to calculate the price, which is used for liquidity provision, DAO configuration, and the performance package. However, the logic for checking that the provided `final_raise_amount` can only be set by the launch authority is flawed. It works by:

1. Load the final raise amount from the args 2. If `launch_authority` is None, override it with the launch's `total_commited_amount` 3. ensure the final raise amount is larger than the minimum raise amount.

However, `launch_authority` has the type `Option>`, which means it can either be None, or it has to be a Signer. To prevent that we can't just sign with any account, the validation function checks the following: If the current timestamp is within two days after the launch close, the launch authority has to be set and has to match the launch's configured launch authority. Now the flaw becomes obvious: After two days, the launch authority isn't checked at all in the validation function. This means that two days after a launch has passed, anyone can supply an arbitrary signing account as the launch authority, which allows them to also set an arbitrary final raise amount. This allows for two ways to exploit this:

• set a very low `final_raise_amount`, as low as the minimum raise amount allows, which will create an artificially low price, allowing an attacker to buy tokens below fair market value in a bundle right after launch completion, and make performance package completion easier as the price anchor is lower • set a very high `final_raise_amount`, allowing an attacker to sell tokens into the LP pool at an inflated price, also setting a high performance package price anchor making completion of them much harder

Location

https://github.com/metaDAOproject/futarchy/blob/5d270a1bcf88b0b56adec5a72a4ff56a18dd9db7/programs/launchpad/src/instructions/complete_launch.rs#L148-L148 https://github.com/metaDAOproject/futarchy/blob/5d270a1bcf88b0b56adec5a72a4ff56a18dd9db7/programs/launchpad/src/instructions/complete_launch.rs#L251-L263 https://github.com/metaDAOproject/futarchy/blob/5d270a1bcf88b0b56adec5a72a4ff56a18dd9db7/programs/launchpad/src/instructions/complete_launch.rs#L269-L276

Relevant Code

```
/// complete_launch.rs L148-L148
    pub launch_authority: Option<Signer<'info>>,

/// complete_launch.rs L251-L263
        let two_days_after_close = self.launch.unix_timestamp_closed.unwrap() + 60 * 60 * 24 * 2;
        if two_days_after_close > clock.unix_timestamp {
            if self.launch_authority.is_none() {
                msg!("Launch authority must complete launch until unix timestamp {}. Current time is {}.", tw
o_days_after_close, clock.unix_timestamp);
```

```
                return Err(LaunchpadError::LaunchAuthorityNotSet.into());
            }

            require_keys_eq!(
                self.launch_authority.as_ref().unwrap().key(),
                self.launch.launch_authority,
                LaunchpadError::LaunchAuthorityNotSet
            );
        }

/// complete_launch.rs L269-L276
        let CompleteLaunchArgs {
            mut final_raise_amount,
        } = args;

        // if the launch authority hasn't come back, ignore whatever the user put in
        if ctx.accounts.launch_authority.is_none() {
            final_raise_amount = ctx.accounts.launch.total_committed_amount;
        }
```

Mitigation Suggestion

First of all, letting an authority set an arbitrary final raise amount sounds like a very bad idea and could be interpreted as a backdoor for the launch authority in the first place. An easy fix would be to delete the feature and always rely on the real contribution amount. Otherwise, the launch authority should be checked if it is set in any case, not just within the two day after launch window.

Remediation

Fixed in commit `24cb6cb3d27f04f0338c82c6d270dd2896c97b68`.

ID	ACC-L1
Title	Potential TWAP reset griefing attack possible
Severity	low
Status	wontfix

Description

Only the recipient can start an unlock, but anyone can complete an unlock. So people could grief a recipient and complete their unlocks when the TWAP hasn't quite reached the target amount, which would require the recipient to wait the full TWAP duration again. This is particularly interesting for very long TWAP periods and combined tranches. For example, the default performance package creates five tranches that unlock when the price doubles, quadruples, or does a 8x, 16x, or 32x. The TWAP period is configured as 3 months. Now consider a bull market scenario in which the TWAP has been initialized 3 months ago, the price has crossed the 2x threshold within the TWAP, and even has crossed the 4x spot price, but the TWAP still needs a few days to cross the 4x mark for the second tranche unlock. Now anyone could call `complete_unlock` to trigger paying out the first tranche prematurely and resetting the TWAP. To unlock the second tranche, another 3 months would have to pass from now on, and after two months the bull market may be over already.

Location

https://github.com/metaDAOproject/futarchy/blob/5d270a1bcf88b0b56adec5a72a4ff56a18dd9db7/programs/price_based_performance_package/src/instructions/complete_unlock.rs#L11-L47

Relevant Code

```
/// complete_unlock.rs L11-L47
pub struct CompleteUnlock<'info> {
    #[account(mut, has_one = token_mint, has_one = performance_package_token_vault)]
    pub performance_package: Box<Account<'info, PerformancePackage>>,

    /// CHECK: We will read the aggregator value from this account
    #[account(address = performance_package.oracle_config.oracle_account)]
    pub oracle_account: UncheckedAccount<'info>,

    /// The token account where locked tokens are stored
    #[account(mut)]
    pub performance_package_token_vault: Box<Account<'info, TokenAccount>>,

    /// The token mint - validated via has_one constraint on locker
    pub token_mint: Account<'info, Mint>,

    /// The recipient's ATA where tokens will be sent - created if needed
    #[account(
        init_if_needed,
        payer = payer,
        associated_token::mint = token_mint,
        associated_token::authority = token_recipient
    )]
    pub recipient_token_account: Box<Account<'info, TokenAccount>>,

    /// CHECK: validated to match locker.token_recipient
    #[account(address = performance_package.recipient @ PriceBasedPerformancePackageError::UnauthorizedChange
Request)]
    pub token_recipient: UncheckedAccount<'info>,

    /// Payer for creating the ATA if needed
    #[account(mut)]
    pub payer: Signer<'info>,
```

```
    pub system_program: Program<'info, System>,

    pub token_program: Program<'info, Token>,
    pub associated_token_program: Program<'info, AssociatedToken>,
}
```

Mitigation Suggestion

Allowing anyone to finish a TWAP and complete an unlock somewhat prevents cherry-picking good time slots, however invites this kind of attack. One fix could be to allow only the recipient and DAO to complete an unlock, allowing a bit of cherry picking on their side. Potentially a mechanism that doesn't fully reset the TWAP could be considered as well.

Remediation

This issue has been accepted as intentional behavior.

ID	ACC-L2
Title	`ChangeRequest` can become unclosable
Severity	low
Status	fixed

Description

We found that `ChangeRequest` accounts can become unclosable in the following sequence of events: • `PerformancePackage` is create with `Bob` as the `recipient` and `Alice` as the `performance_package_authority` • `Bob` proposes an oracle change (proposal `P1`) • Before executing `P1` a recipient change proposal is created and executed • The `P1` proposal cannot be executed or closed

Location

https://github.com/metaDAOproject/programs/blob/405880ed29ff14450f6252ca42468f02b3c47b6f/programs/price_based_performance_package/src/instructions/execute_change.rs#L39

Relevant Code

```
pub fn validate(&self) -> Result<()> {
        if self.change_request.proposer == self.performance_package.recipient {
            // If recipient proposed, locker authority must execute
            require_keys_eq!(
                self.executor.key(),
                self.performance_package.performance_package_authority,
                PriceBasedPerformancePackageError::UnauthorizedLockerAuthority
            );
        } else if self.change_request.proposer == self.performance_package.performance_package_authority {
            // If locker authority proposed, recipient must execute
            require_keys_eq!(
                self.executor.key(),
                self.performance_package.recipient,
                PriceBasedPerformancePackageError::UnauthorizedChangeRequest
            );
        } else {
            // Proposer was neither valid party - should not happen due to proposal constraints
            return Err(PriceBasedPerformancePackageError::UnauthorizedChangeRequest.into());
        }

        Ok(())
    }
```

Mitigation Suggestion

`ChangeRequest` should be closable when the `proposer` is no longer a party to the `PerformancePackage`

Remediation

Fixed in commit `7e2201914dc2bf9202980f72b50f526a621acbf7`.

ID	ACC-I1
Title	Lack of validation of `twap_length_seconds`
Severity	info
Status	fixed

Description

We found that `twap_length_seconds` lacks validation, to prevent edge cases and mistakes with pathological values consider enforcing`0 < twap_length_seconds =< i64::MAX`

Location

https://github.com/metaDAOproject/programs/blob/405880ed29ff14450f6252ca42468f02b3c47b6f/programs/price_based_performance_package/src/instructions/initialize_performance_package.rs#L59

Mitigation Suggestion

Enforce `0 < twap_length_seconds <= i64::MAX` or even with a stricter upper bound to prevent accidental setting that could lock tokens for impractically long periods.

Remediation

Fixed in commit `3b10bf4192c4bda57c7c692f72b615fcb00634ed`.

APPENDIX

Vulnerability Classification

We rate our issues according to the following scale. Informational issues are reported informally to the developers and are not included in this report.

Severity	Description
Critical	Vulnerabilities that can be easily exploited and result in loss of user funds, or directly violate the protocol's integrity. Immediate action is required.
High	Vulnerabilities that can lead to loss of user funds under non-trivial preconditions, loss of fees, or permanent denial of service that requires a program upgrade. These issues require attention and should be resolved in the short term.
Medium	Vulnerabilities that may be more difficult to exploit but could still lead to some compromise of the system's functionality. For example, partial denial of service attacks, or such attacks that do not require a program upgrade to resolve, but may require manual intervention. These issues should be addressed as part of the normal development cycle.
Low	Vulnerabilities that have a minimal impact on the system's operations and can be fixed over time. These issues may include inconsistencies in state, or require such high capital investments that they are not exploitable profitably.
Informational	Findings that do not pose an immediate risk but could affect the system's efficiency, maintainability, or best practices.

Audit Methodology

Accretion is a boutique security auditor specializing in Solana's ecosystem. We employ a customized approach for each client, strategically allocating our resources to maximize code review effectiveness. Our auditors dedicate substantial time to developing a comprehensive understanding of each program under review, examining design decisions, expected and edge-case behaviors, invariants, optimizations, and data structures, while meticulously verifying mathematical correctness—all within the context of the developers' intentions.

Our audit scope extends beyond on-chain components to include associated infrastructure, such as user interfaces and supporting systems. Every audit encompasses both a holistic protocol design review and detailed line-by-line code analysis.

During our assessment, we focus on identifying:
• Solana-specific vulnerabilities
• Access control issues
• Arithmetic errors and precision loss
• Race conditions and MEV opportunities
• Logic errors and edge cases
• Performance optimization opportunities
• Invariant violations
• Account confusion vulnerabilities
• Authority check omissions
• Token22 implementation risks and SPL-related pitfalls
• Deviations from best practices

Our approach transcends conventional vulnerability classifications. We continuously conduct ecosystem-wide security research to identify and mitigate emerging threat vectors, ensuring our audits remain at the forefront of Solana security practices.