



SECURITY ASSESSMENT

Provided by Accretion Labs Pte Ltd. for MetaDAO
March 24, 2025
A25MET1



AUDITORS

Role	Name
Lead Auditor	Robert Reith (robert@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 related technology for other DAOs.

ENGAGEMENT SCOPE

MetaDAO Launchpad

Link: <https://github.com/metaDAOproject/futarchy>

Commit: 3bcb3a735530899c7565de670bd3c0277872359

ProgramID: AfJJJ5UqxhBKoE3grkKAZZsoXDE9kncbMKvqSHGsCnRE

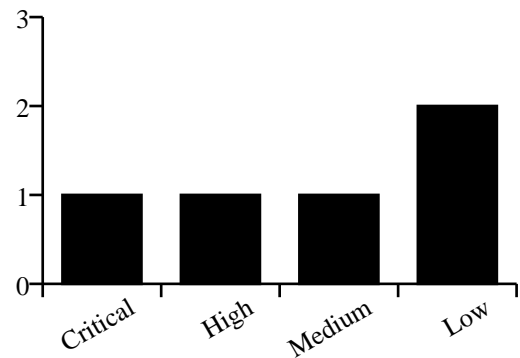
ENGAGEMENT TIMELINE

- 01 Mar **Project Kickoff**
Initial planning and scope definition
- 03 Mar **Assessment Begins**
Security review and testing phase
- 10 Mar **Review Fixes**
Security recommendations are given and implemented
- 24 Mar **Project Completion**
Report delivery and onchain verification

ASSESSMENT

The security assessment of MetaDAO's Launchpad revealed two critical and high severity issues within the initialization of a new launch, while the remaining program contained mostly lower severity issues. The overall quality of the program is very good, with a clear coding style and a simple program design. The MetaDAO team responded quickly to all issues, resolving them competently.

SEVERITY DISTRIBUTION



VERIFIED ON-CHAIN CODE

Program 1

ProgramID: AfJJJ5UqxhBKoE3grkKAZZsoXDE9kncbMKvqSHGsCnRE

Repository: <https://github.com/metaDAOproject/futarchy>

Build Hash: aefa93700e9f373bff88e8bc1eb8de9d7bcf465cc24757a99b4c804e4ee326b7

Commit: 4124b68f6021c1dde2782b6231cec3341518d0b9

ISSUES SUMMARY

ID	TITLE	SEVERITY	STATUS
ACC-C1	Launch Signer not checked on initialization	critical	fixed
ACC-H1	The launch mint is not validated properly, allowing for malicious freeze authority	high	fixed
ACC-M1	Lauch token metadata isn't claimed and can be stolen	medium	fixed
ACC-L1	Launch Completion crank controls DAO pubkey	low	fixed
ACC-L2	Possible to fund after funding period is over	low	fixed
ACC-I1	Using slots as a time measurement	info	fixed
ACC-I2	Payer-Signer pattern	info	fixed

DETAILED ISSUES

ID	ACC-C1
Title	Launch Signer not checked on initialization
Severity	critical
Status	fixed

Description

In the `InitializeLaunch` instruction, the `launch_signer` account is passed as an `Unchecked` account, but in the constraints it is asserted that it is the authority of the given `usdc_vault`, `token_vault`, and the mint authority. The handler then derives the key for the launch signer itself, and writes the derived key into the launch state account, without checking that the derivation matches the provided key in the accounts. At the end of the function, the token program is called to mint tokens using the unchecked launch signer from the accounts, and the launch signer seeds are passed as a signer. If an attacker signs this transaction using their fake launch signer, this signature will also be passed and the call to the token program will succeed. This means an attacker can create a fake launch signer account and the corresponding mint and token vault accounts, and call the initialization instruction, signing as the launch signer. Later on, they could potentially reassign the owner of the token accounts and the mint authority to the correct launch signer, so that the launchpad program works without any disturbances - however before that they will be able to mint arbitrary tokens, or withdraw from the treasuries.

Location

https://github.com/metaDAOproject/futarchy/blob/ea3b5347ed986c2d306b4bde054ee338bb767071/programs/launchpad/src/instructions/initialize_launch.rs#L17-L130

Relevant Code

```
#[derive(Accounts)]
pub struct InitializeLaunch<'info> {
    #[account(
        init,
        payer = creator,
        space = 8 + std::mem::size_of::<Launch>(),
        seeds = [b"launch", token_mint.key().as_ref()],
        bump
    )]
    pub launch: Account<'info, Launch>,

    /// CHECK: This is the launch signer
    pub launch_signer: UncheckedAccount<'info>,

    #[account(
        associated_token::mint = usdc_mint,
        associated_token::authority = launch_signer
    )]
    pub usdc_vault: Account<'info, TokenAccount>,

    #[account(
        associated_token::mint = token_mint,
        associated_token::authority = launch_signer
    )]
    pub token_vault: Account<'info, TokenAccount>,

    #[account(mut)]
    pub creator: Signer<'info>,
```

```

#[account(mint::decimals = 6)]
pub usdc_mint: Account<'info, Mint>,

#[account(
    mint::decimals = 6,
    mint::authority = launch_signer,
)]
pub token_mint: Account<'info, Mint>,

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

impl InitializeLaunch<'_> {
    pub fn validate(&self, _args: InitializeLaunchArgs) -> Result<()> {
        require_eq!(self.token_mint.supply, 0, LaunchpadError::SupplyNonZero);

        require!(self.token_mint.freeze_authority.is_none(), LaunchpadError::FreezeAuthoritySet);

        Ok(())
    }

    pub fn handle(
        ctx: Context<Self>,
        args: InitializeLaunchArgs,
    ) -> Result<()> {
        let (launch_signer, launch_signer_pda_bump) =
            Pubkey::find_program_address(&[b"launch_signer", ctx.accounts.launch.key().as_ref()], ctx.program_id);

        ctx.accounts.launch.set_inner(Launch {
            minimum_raise_amount: args.minimum_raise_amount,
            creator: ctx.accounts.creator.key(),
            launch_signer,
            launch_signer_pda_bump,
            launch_usdc_vault: ctx.accounts.usdc_vault.key(),
            launch_token_vault: ctx.accounts.token_vault.key(),
            total_committed_amount: 0,
            token_mint: ctx.accounts.token_mint.key(),
            usdc_mint: ctx.accounts.usdc_mint.key(),
            pda_bump: ctx.bumps.launch,
            seq_num: 0,
            state: LaunchState::Initialized,
            slot_started: 0,
            slots_for_launch: args.slots_for_launch,
            dao: None,
            dao_treasury: None,
        });

        let clock = Clock::get()?;
        emit_cpi!(LaunchInitializedEvent {
            common: CommonFields::new(&clock, 0),
            launch: ctx.accounts.launch.key(),
            creator: ctx.accounts.creator.key(),
            usdc_mint: ctx.accounts.usdc_mint.key(),
            token_mint: ctx.accounts.token_mint.key(),
            pda_bump: ctx.bumps.launch,
        });

        let launch_key = ctx.accounts.launch.key();

        let seeds = &[
            b"launch_signer",
            launch_key.as_ref(),
            &[launch_signer_pda_bump],
        ];
        let signer = &[&seeds[..]];

        // Mint total tokens to launch token vault

```

```

token::mint_to(
  CpiContext::new_with_signer(
    ctx.accounts.token_program.to_account_info(),
    MintTo {
      mint: ctx.accounts.token_mint.to_account_info(),
      to: ctx.accounts.token_vault.to_account_info(),
      authority: ctx.accounts.launch_signer.to_account_info(),
    },
    signer,
  ),
  AVAILABLE_TOKENS,
)?;

Ok(())
}
}

```

Mitigation Suggestion

Check the launch signer seeds in the constraints instead of the handler.

Remediation

Fixed in commit 2de1b497f187b22fb4cdfb4a0cfa777165914386

ID	ACC-H1
----	--------

Title	The launch mint is not validated properly, allowing for malicious freeze authority
-------	--

Severity	high
----------	------

Status	fixed
--------	-------

Description

We found that when a new launch is initialized, the launch mint is supplied pre-allocated by the user, and only the decimals, mint authority, and supply are checked. However, it may be possible for an attacker to configure a freeze authority for the mint. This authority will be able to freeze all tokens that will be minted. In practice this may allow them to get users to buy the token, then freeze their tokens, prohibiting them from selling,

Location

https://github.com/metaDAOproject/futarchy/blob/3bcbb3a735530899c7565de670bd3c0277872359/programs/launchpad/src/instructions/initialize_launch.rs#L69-L73 https://github.com/metaDAOproject/futarchy/blob/3bcbb3a735530899c7565de670bd3c0277872359/programs/launchpad/src/instructions/initialize_launch.rs#L82

Relevant Code

```
#[derive(Accounts)]
pub struct InitializeLaunch<'info> {

    // [...] other accounts...

    #[account(
        mint::decimals = 6,
        mint::authority = launch,
    )]
    pub token_mint: Account<'info, Mint>,

    // [...] other accounts ...
}

// [...] impl ...

pub fn validate(&self, _args: InitializeLaunchArgs) -> Result<()> {
    require_eq!(self.token_mint.supply, 0, LaunchpadError::SupplyNonZero);
}
```

Mitigation Suggestion

A direct fix for this issue would be to check that the freeze authority is not set. A better fix would be to derive the mint as a PDA and create the mint with the token program in the same instruction. This guarantees that the mint is created on your terms, and that the address at which the mint is created has not been groomed.

Remediation

Fixed in commit e174226f313707963057d90d24cf845af518e93c.

ID	ACC-M1
Title	Launch token metadata isn't claimed and can be stolen
Severity	medium
Status	fixed

Description

When a new launch is created, the launch mint's metadata are not created. This means that someone could create arbitrary metadata for a new launch, and potentially secure the metadata authority for themselves. To do this, a new mint has to be created with the mint authority set to the attackers key. Using the mint authority, the attacker claims the metadata for the coin and sets themselves as the metadata authority. Afterwards, they change the mint authority to the token launchpad authority, as this program requires, and initialize the launch. This is partially because the program accepts a preallocated mint, and doesn't allocate it itself, and partially because it doesn't try to claim the metadata.

Location

https://github.com/metaDAOproject/futarchy/blob/3bcbb3a735530899c7565de670bd3c0277872359/programs/launchpad/src/instructions/initialize_launch.rs#L69-L73

Relevant Code

```
#[derive(Accounts)]
pub struct InitializeLaunch<'info> {

    // [...] other accounts...

    #[account(
        mint::decimals = 6,
        mint::authority = launch,
    )]
    pub token_mint: Account<'info, Mint>,

    // [...] other accounts ...
}
```

Mitigation Suggestion

We suggest creating the mint account in the initialization instruction, which guarantees that the mint didn't have a different mint authority before. In addition to that, we suggest also initializing the metadata in the same instruction.

Remediation

Fixed in commit 34640635238e53e42d40e1c211e2546ca0a6bbe6.

ID	ACC-L1
Title	Launch Completion crank controls DAO pubkey
Severity	low
Status	fixed

Description

Whoever cranks the launch completion instruction will be able to set the DAO's main pubkey because they provide and sign a Keypair account which will be used as the DAO account. One malicious scenario would be for an attacker to grind a highly offensive Pubkey, for example something starting with a slur. Then, they would be able to make this slur Pubkey the official Pubkey of a legitimate launchpad DAO that has successfully raised funds. We believe this could directly impact the impacted DAOs value, and also would dissuade future DAOs from using the launchpad.

Location

https://github.com/metaDAOproject/futarchy/blob/bc25f3bde90d2f347770b6577f9cd57101b74b0b/programs/launchpad/src/instructions/complete_launch.rs#L159-L161

Relevant Code

```
/// CHECK: this is the DAO account, init by autocrat
#[account(mut)]
pub dao: Signer<'info>,
```

Mitigation Suggestion

We believe that the DAO pubkey should be a PDA derived from the launch and an additional arbitrary seed defined at launch creation. This would allow a launch to use a good novelty address.

Remediation

Fixed in commit 7b684ac6384649591e91909147e3456d3eea5902 by making the DAO account a PDA.

ID	ACC-L2
Title	Possible to fund after funding period is over
Severity	low
Status	fixed

Description

As long as the `complete_launch` instruction hasn't been called yet, anyone can keep funding a launch even when the launch period time is over.

Location

<https://github.com/metaDAOproject/futarchy/blob/34640635238e53e42d40e1c211e2546ca0a6bbe6/programs/launchpad/src/instructions/fund.rs#L48-L56>

Relevant Code

```
pub fn validate(&self, amount: u64) -> Result<()> {  
    require!(amount > 0, LaunchpadError::InvalidAmount);  
  
    require_gte!(self.funder_usdc_account.amount, amount, LaunchpadError::InsufficientFunds);  
  
    require!(self.launch.state == LaunchState::Live, LaunchpadError::InvalidLaunchState);  
  
    Ok(())  
}
```

Mitigation Suggestion

Add a requirement that the current time/slot is within the funding period.

Remediation

Fixed in commit 06bc0a13d2bf34fa1e97faff2dc938aa234be65b.

ID	ACC-I1
Title	Using slots as a time measurement
Severity	info
Status	fixed

Description

The program uses slots as a measure of time. However, this is imprecise, as slot times not only vary in length naturally, but also there are concrete plans to reduce slot times from the current 400ms in the future. Instead, an actual timestamp should be used.

Location

https://github.com/metaDAOproject/futarchy/blob/34640635238e53e42d40e1c211e2546ca0a6bbe6/programs/launchpad/src/instructions/initialize_launch.rs#L110

Relevant Code

```
ctx.accounts.launch.set_inner(Launch {  
  // ...  
    slots_for_launch: args.slots_for_launch,  
  // ...  
});
```

Mitigation Suggestion

Replace time measurements by slots with timestamps.

Remediation

Fixed in commit 7a86f3919b93ee55e81ffa9e4c75d19c37c3cd39.

ID	ACC-I2
Title	Payer-Signer pattern
Severity	info
Status	fixed

Description

We recommend implementing the payer-signer pattern so that you have a separate account for fee payments and one for account creation fees. This allows the program to be better used by multisigs, DAOs, or other programs. This affects both fund, where the funding account is created and initialize launch, where the launch account is created.

Location

<https://github.com/metaDAOproject/futarchy/blob/34640635238e53e42d40e1c211e2546ca0a6bbe6/programs/launchpad/src/instructions/fund.rs#L10-L45> https://github.com/metaDAOproject/futarchy/blob/34640635238e53e42d40e1c211e2546ca0a6bbe6/programs/launchpad/src/instructions/initialize_launch.rs#L25-L85

Relevant Code

```
pub struct Fund<'info> {
    #[account(
        mut,
        has_one = launch_signer,
        has_one = launch_usdc_vault,
    )]
    pub launch: Account<'info, Launch>,

    #[account(
        init_if_needed,
        payer = funder,
        space = 8 + std::mem::size_of::<FundingRecord>(),
        seeds = [b"funding_record", launch.key().as_ref(), funder.key().as_ref()],
        bump
    )]
    pub funding_record: Account<'info, FundingRecord>,

    /// CHECK: just a signer
    pub launch_signer: UncheckedAccount<'info>,

    #[account(mut)]
    pub launch_usdc_vault: Account<'info, TokenAccount>,

    #[account(mut)]
    pub funder: Signer<'info>,

    #[account(
        mut,
        token::mint = launch.usdc_mint,
        token::authority = funder
    )]
    pub funder_usdc_account: Account<'info, TokenAccount>,

    pub token_program: Program<'info, Token>,
    pub system_program: Program<'info, System>,
}

pub struct InitializeLaunch<'info> {
    #[account(
```

```

        init,
        payer = creator,
        space = 8 + std::mem::size_of::<Launch>(),
        seeds = [b"launch", token_mint.key().as_ref()],
        bump
    )]
pub launch: Account<'info, Launch>,

#[account(
    init,
    payer = creator,
    mint::decimals = 6,
    mint::authority = launch_signer,
)]
pub token_mint: Account<'info, Mint>,

/// CHECK: This is the token metadata
#[account(
    mut,
    seeds = [b"metadata", MPL_TOKEN_METADATA_PROGRAM_ID.as_ref(), token_mint.key().as_ref()],
    seeds::program = MPL_TOKEN_METADATA_PROGRAM_ID,
    bump
)]
pub token_metadata: UncheckedAccount<'info>,

/// CHECK: This is the launch signer
#[account(
    seeds = [b"launch_signer", launch.key().as_ref()],
    bump
)]
pub launch_signer: UncheckedAccount<'info>,

#[account(
    associated_token::mint = usdc_mint,
    associated_token::authority = launch_signer
)]
pub usdc_vault: Account<'info, TokenAccount>,

#[account(
    init,
    payer = creator,
    associated_token::mint = token_mint,
    associated_token::authority = launch_signer
)]
pub token_vault: Account<'info, TokenAccount>,

#[account(mut)]
pub creator: Signer<'info>,

#[account(mint::decimals = 6)]
pub usdc_mint: Account<'info, Mint>,

pub rent: Sysvar<'info, Rent>,

pub token_program: Program<'info, Token>,
pub associated_token_program: Program<'info, AssociatedToken>,
pub system_program: Program<'info, System>,
pub token_metadata_program: Program<'info, Metadata>,
}

```

Mitigation Suggestion

Add a separate signer at the beginning of the accounts struct who will be responsible for paying rent fees. Users will still be able to provide the same account for fee payer and for funder or creator.

Remediation

Fixed in commits 4212442e4ba622e045a422a57d55410c5c44a853 and
f3bf97d2128105d01f7c60274c28fecc2d737e7e.

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.