Accretion

SECURITY ASSESSMENT

Provided by Accretion Labs Pte Ltd. for MetaDAO
July 16, 2025
A25MET2

MetaDAO

AUDITORS

Role	Name
Lead Auditor	Robert Reith (robert@accretion.xyz)
Auditor	Niklas Brymko (niklas@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.

ENGAGEMENT SCOPE

Pull Request 310

Link: https://github.com/metaDAOproject/futarchy/pull/310
Commit: cb2abb217513f14d788041a6e217fd8441d9c3d3
ProgramID: AMMJdEiCCa8mdugg6JPF7gFirmmxisTfDJoSNSUi5zDJ, mooNhciQJi1LqHDmse2JPic2NqG2PXCanbE3ZYzP3qA, auToUr3CQza3D4qreT6Std2MTomfzvrEeCC5qh7ivW5

ENGAGEMENT TIMELINE

03 Jul — Project Kickoff
Initial planning and scope definition

03 Jul — Assessment Begins
Security review and testing phase

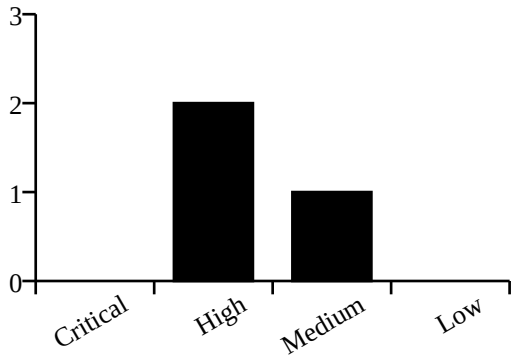
15 Jul — Review Fixes
Security recommendations are given and implemented

16 Jul — Project Completion
Report delivery and on-chain confirmation

ASSESSMENT

The security assessment of the MetaDAO's Squads PR revealed some interesting interactions regarding the permissionless account which is to be used as the public initiator and executor of proposals. We identified multiple issues relating to this public account, such as turning the account into a program which would prevent it from ever being passed as writeable, partially bricking important functionality in Squads. Furthermore, attackers could attack users who try to build a proposal in multiple transactions using Squads' batched transactions of Transaction Buffers. During the audit an elegant solution was found to assign the public account to the autocrat program, while also putting the responsibility onto users to use bundles or atomic transactions when building their proposal. We also recommended adding a third private proposer/executor account to the multisig.

SEVERITY DISTRIBUTION



AUDITED CODE

Program 1

ProgramID: AMMJdEiCCa8mdugg6JPF7gFirmmxisTfDJoSNSUi5zDJ, mooNhciQJi1LqHDmse2JPic2NqG2PXCanbE3ZYzP3qA, auToUr3CQza3D4qreT6Std2MTomfzvrEeCC5qh7ivW5
Repository: https://github.com/metaDAOproject/futarchy
Commit: c8b1c50a4fe1fcc9a09ee3ac56512def640d3a18

ISSUES SUMMARY

ID	TITLE	SEVERITY	STATUS
ACC-H1	The Permissionless Account can Partially DoS a Multisig	high	fixed
ACC-H2	Permissionless Account Can Abuse and Steal Funds from Honest Proposers	high	fixed
ACC-M1	The Permissionless Account can partially DoS certain multisig features	medium	fixed
ACC-I1	No check that a new proposal belongs to the correct multisig	info	fixed

DETAILED ISSUES

ID	ACC-H1
Title	The Permissionless Account can Partially DoS a Multisig
Severity	high
Status	fixed

Description

We found that the permissionless account can fully DoS a multisig when someone deploys a program at the permissionless account address. The reason is that the `ProposalActivate` instruction of Squads requires the member who is the initiator of the proposal to be passed as mutable to the program. The solana runtime does not allow executable accounts to be passed as mutable, which means that it becomes impossible to call `ProposalActivate`, which is a required step of executing draft proposals through squads.

It would still be possible to create proposals through `ProposalCreate` and setting the params to skip the drafting stage, which means we won't need to call `ProposalActivate`. However features that require the drafting stage will become fully bricked, such as batched transactions.

Location

https://github.com/Squads-Protocol/v4/blob/de4a03b594b2f06ed0b558f99a35ebbf8dc8913d/programs/squads_multisig_program/src/instructions/proposal_activate.rs#L14-L15

Relevant Code

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

Mitigation Suggestion

We will certainly have to find an alternative solution to the permissionless account model. A simpler fix could be assigning the permissionless accounts program owner to the autocrat program, so that it can't be turned into a program.

Remediation

The permissionless account has been assigned to the autocrat program, which prevents it from becoming a program. https://explorer.solana.com/tx/2xPdX4TZx2gMssCtLkxxeYzmGoTGaQFsqpy8BCKp1WXrsqoyE4nd9sZebC8WnV6t4FBpU3H2FbvkbL7CGSMih5id

ID	ACC-H2
Title	Permissionless Account Can Abuse and Steal Funds from Honest Proposers
Severity	high
Status	fixed

Description

We found that as the program uses a permissionless account as a member of the treasury multisig, and this account has initiate and execute rights, and the private key for that account is public, this means that users who create proposal for the multisig are at risk from other users with access to the same permissionless account adding instructions to their proposal. The attack would look like this:

User A uses the permissionless account to initialize a batch proposal: • squads::batch_create (transaction index 1) • squads::proposal_create (transaction index 1) • squads::batch_add_transaction • squads::batch_add_transaction <-- ATTACKER SENDS TRANSACTION: squads::batch_add_transaction • squads::proposal_activate (transaction index 1) • metadao::create proposal

The user might not realize that the proposal that they created now has an additional batch transaction.

There are also other attacks that anyone can do by abusing the public and shared init authority. For example, anyone can activate a proposal early, before the proposer has added all instructions.

In a more severe attack, which leads to LoF of the honest proposer, the attacker can close a transaction buffer before it is turned into a vault transaction. For this the attacker simply waits for any transaction buffer to be created, and then immediately calls the `transaction_buffer_close` instruction on it to close the buffer and steal the rent that the honest proposal creator had deposited. An attacker could also close and reopen a transaction buffer at the same buffer index, to try to sneak a malicious transaction through.

Location

https://github.com/metaDAOproject/futarchy/blob/cb2abb217513f14d788041a6e217fd8441d9c3d3/programs/autocrat/src/instructions/initialize_dao.rs#L54-L58

Relevant Code

```
/// initialize_dao.rs L54-L58
pub mod permissionless_account {
    use anchor_lang::prelude::declare_id;

    declare_id!("EP3SoC2SvR3d4c2eXVBvhEMWSr2j3YtoCY3UMiQV7BPD");
}
```

Mitigation Suggestion

The only real protection would be to enforce that all proposal creations are done atomically, either in a single transaction or as a bundle. This way, a dishonest 3rd party can't abuse the public private key to insert themselves in the process of someone else's proposal creation. However, we can't really enforce this type of usage and it would be better to find an on-chain solution. One option would be to make the public proposer/executor account a PDA of a helper program which ensures correct usage of squads, multiplexing access to this account while maintaining proposal integrity. However, this would require a whole new, potentially complex program. We could also use a PDA and limit access to squads by only allowing few instructions to be called.

We could leave the program as-is, BUT each proposals instructions have to be carefully examined so that no one was able to insert any additional instructions, or change any instructions. Users have to be aware that they may lose funds when creating transaction buffers. And proposals are at risk of being griefed by proposals being activated early, instructions being added, and so on. As these proposal go to trade, this could create a different opportunity for an attacker. They can wait for a proposal to be created, maliciously add an instruction to it, and wait for trading to start on this proposal, while fully aware that the malicious instruction will be caught. Using this information advantage, the attacker can now potentially buy into the fail market before anyone notices that a malicious instruction has been added to the proposal.

So generally we would strongly recommend finding a real solution to this issue.

Remediation

The permissionless account has been assigned to the autocrat program, to partially fix these issues. For instance, anyone will still be able to close a Buffer, but the funds will go to the permissionless account, while its owned by autocrat, so an attacker doesn't gain an advantage as they can't withdraw funds from the account. The program is still vulnerable to unauthorized additions to proposal batches and edits to buffers, but the responsibility of preventing this is laid upon the user, for example by using Jito bundles. https://explorer.solana.com/tx/2xPdX4TZx2gMssCtLkxxeYzmGoTGaQFsqpy8BCKp1WXrsqoyE4nd9sZebC8WnV6t4FBpU3H2FbvkbL7CGSMih5id

ID	ACC-M1
Title	The Permissionless Account can partially DoS certain multisig features
Severity	medium
Status	fixed

Description

We found that the permissionless account can be made to partially DoS some squads multisig features. In particular, the `VaultTransactionCreateFromBuffer` method can be completely DoSed. This method lets a multisig member with the initiation permission create a Transaction Buffer, fill that buffer with a long transaction, and then turn that transaction buffer into a `VaultTransaction`. When the buffer is turned into a `VaultTransaction`, the buffer is closed and the rent is refunded to the creator of the buffer. This creator of the buffer has to be the permissionless account, which is passed as mutable to this instruction. However, if an account is executable, it can't be passed as mutable to an instruction. This means, if anyone deploys a program at the permissionless account, which anyone can do, the `VaultTransactionCreateFromBuffer` becomes unusable for this Multisig, because it requires this account to be passed as mutable. We will still be able to create transactions for the multisig using other methods.

Location

https://github.com/Squads-Protocol/v4/blob/de4a03b594b2f06ed0b558f99a35ebbf8dc8913d/programs/squads_multisig_program/src/instructions/vault_transaction_create_from_buffer.rs#L11C1-L34C32

Relevant Code

```
#[account(
        mut,
        close = creator,
        // Only the creator can turn the buffer into a transaction and reclaim
        // the rent
        constraint = transaction_buffer.creator == creator.key() @ MultisigError::Unauthorized,
        seeds = [
            SEED_PREFIX,
            vault_transaction_create.multisig.key().as_ref(),
            SEED_TRANSACTION_BUFFER,
            creator.key().as_ref(),
            &transaction_buffer.buffer_index.to_le_bytes(),
        ],
        bump
    )]
    pub transaction_buffer: Box<Account<'info, TransactionBuffer>>,

    // Anchor doesn't allow us to use the creator inside of
    // vault_transaction_create, so we just re-pass it here with the same constraint
    #[account(
        mut,
        address = vault_transaction_create.creator.key(),
    )]
    pub creator: Signer<'info>,
```

Mitigation Suggestion

We will need to find a different solution for the permissionless account, such as a PDA. A simpler fix could be assigning the permissionless accounts program owner to the autocrat program, so that it can't be turned into a program.

Remediation

The permissionless account has been assigned to the autocrat program to prevent these attacks. https://explorer.solana.com/tx/2xPdX4TZx2gMssCtLkxxeYzmGoTGaQFsqpy8BCKp1WXrsqoyE4nd9sZebC8WnV6t4FBpU3H2FbvkbL7CGSMih5id

ID	ACC-I1
Title	No check that a new proposal belongs to the correct multisig
Severity	info
Status	fixed

Description

We found that when a new proposal is created using `InitializeProposal`, there is no check that the `squads_proposal` belongs to the DAO's squads multisig. It could be a proposal belonging to a different multisig altogether.

Furthermore, there should be additional checks on the proposal, such as that it's not an old proposal.

Location

https://github.com/metaDAOproject/futarchy/blob/cb2abb217513f14d788041a6e217fd8441d9c3d3/programs/autocrat/src/instructions/initialize_proposal.rs#L27-L27

Relevant Code

```
/// initialize_proposal.rs L27-L27
    pub squads_proposal: Box<Account<'info, squads_multisig_program::Proposal>>,
```

Mitigation Suggestion

Add a check that the proposal belongs to the official multisig of this dao. Add a check that confirms the state of the proposal. It should be up for voting.

Remediation

Fixed in commits `970f9063415ce393b5b37388e12436489da527e6` and `c8b1c50a4fe1fcc9a09ee3ac56512def640d3a18`.

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.