

Squads v4

Audit

Presented by:

OtterSec

Ajay Kunapareddy

Robert Chen

contact@osec.io

d1r3wolf@osec.io

r@osec.io



Contents

01 Executive Summary	2
Overview	2
Key Findings	2
02 Findings	3
03 Vulnerabilities	4
OS-SQD-ADV-00 [low] Inaccurate CPI Semantics	5
OS-SQD-ADV-01 [low] Incorrect Reallocation Size	6
OS-SQD-ADV-02 [low] Blocked Transaction Addition	7
04 General Findings	8
OS-SQD-SUG-00 Lack Of Member Checks	9
OS-SQD-SUG-01 Missing Permission Range Check	10
OS-SQD-SUG-02 Clarify CPI Semantics	11
 Appendices	
A Vulnerability Rating Scale	12
B Procedure	13

01 | Executive Summary

Overview

Squads Protocol engaged OtterSec to perform an assessment of the v4 program. This assessment was conducted between July 6th and September 22nd, 2023. For more information on our auditing methodology, see [Appendix B](#).

Key Findings

Over the course of this audit engagement, we produced 6 findings in total.

In particular, we addressed noted a number of subtle issues regarding inaccurate CPI account writability semantics ([OS-SQD-ADV-00](#)), improper calculation of the space required for reallocating accounts ([OS-SQD-ADV-01](#)), and potential denial of service issues when adding transactions to a batch ([OS-SQD-ADV-02](#)).

We also made various recommendations concerning validation checks for spending limit members ([OS-SQD-SUG-00](#)), range checks for member permissions ([OS-SQD-SUG-01](#)), and improved mutable CPI checks to prevent unnecessarily denying transactions ([OS-SQD-SUG-02](#)).

Overall, we noted that the code quality of the program was high and the design was solid. The team was also very knowledgeable and responsive to our feedback.

Scope

The source code was delivered to us in a git repository at github.com/Squads-Protocol/v4. This audit was performed against commit [7d79e69](#).

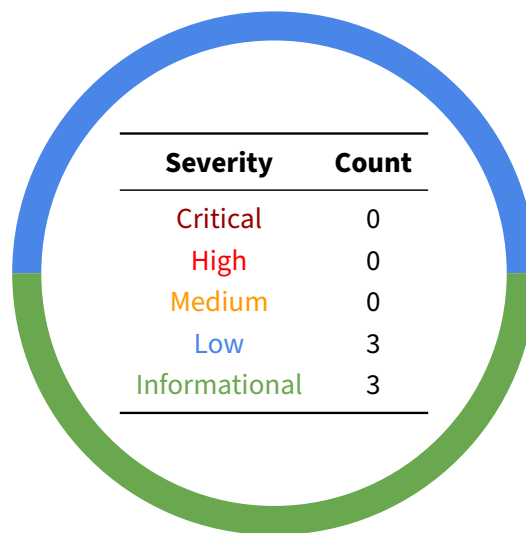
A brief description of the programs is as follows.

Name	Description
v4	<p>As a programmable multisig wallet for Solana, Squads provides the functionality to easily manage developer assets such as programs, tokens, validators, and treasury assets.</p> <p>The multisig enables configurable signature thresholds for the execution of transactions. In addition, a new spending limit feature is introduced, allowing for programmatic limits on token transfers as a first-class feature.</p>

02 | Findings

Overall, we reported 6 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will help mitigate future vulnerabilities.



03 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in [Appendix A](#).

ID	Severity	Status	Description
OS-SQD-ADV-00	Low	Resolved	The executor may specify inaccurate writability flags for the underlying transaction in <code>execute_message</code> .
OS-SQD-ADV-01	Low	Resolved	While adding members to an account, insufficient reallocation may occur due to incorrect calculations.
OS-SQD-ADV-02	Low	Resolved	Setting the proposal for a batch to an active state may prevent the addition of transactions to the batch.

OS-SQD-ADV-00 [low] | Inaccurate CPI Semantics

Description

When executing a transaction from a Squads multisig, the executor is able to alter the `is_writable` status of accounts.

This is because transactions derive this flag from the passed in `AccountInfo` instead of using the stored data in `VaultTransactionMessage`.

utils/executable_transaction_message.rs

RUST

```
// `is_signer` cannot just be taken from the account info, because for
// ↪ `authority`
// it's always false in the passed account infos, but might be true in the
// ↪ actual instructions.
let is_signer = self.message.is_signer_index(account_index);

let account_meta = if account_info.is_writable {
    AccountMeta::new(*account_info.key, is_signer)
} else {
    AccountMeta::new_readonly(*account_info.key, is_signer)
```

As a result, a malicious multisig executor could violate the intended execution behavior, passing accounts as writable even though the original proposed transaction did not specify writability. Although unlikely, this technically could alter program behavior.

Remediation

Use a writable `AccountMeta` depending on the passed writability specified in `self.message`.

Patch

Resolved in [c3d2177](#) and [2ddacd2](#) by setting the writable attribute based on `loaded_writable_accounts`.

OS-SQD-ADV-01 [low] | Incorrect Reallocation Size

Description

When adding a member to a multi-signature account, if the pre-allocated memory becomes fully utilized, it is necessary to reallocate. `realloc_if_needed` assesses the requirement for reallocation and performs reallocation for the account, providing additional space for ten members.

The number of members is expanded in two instances:

- `multisig_add_member`.
- `config_transaction_execute`.

While the former only increases the member count by one, the latter may increase the member count by more than one due to the possibility of executing multiple actions. Therefore, if the size of members to add exceeds the sum of the remaining space and the size of ten members, the fixed increment of ten members may result in a shortage of space.

Remediation

Select the larger value between the size increased by ten members and the size increased as needed.

state/multisig.rs

DIFF

```
@@ -79,14 +81,17 @@ impl Multisig {  
-     let new_size = current_account_size + (10 * Member::INIT_SPACE);  
+     let new_size = max(  
+         current_account_size + (10 * Member::INIT_SPACE),  
+         account_size_to_fit_members,  
+     );
```

Patch

Fixed in [5640af0](#) by taking the larger size for new members or fixed ten members.

OS-SQD-ADV-02 [low] | Blocked Transaction Addition

Description

After creating a batch, adding transactions necessitates a proposal generated by `proposal_create`. The program-derived address of the proposal uses the transaction index utilized in `batch_create` as a seed. When initializing the proposal, the caller may select between the `Draft` and `Active` states.

instructions/proposal_create.rs

RUST

```
pub proposal: Account<'info, Proposal>,
pub fn proposal_create(ctx: Context<Self>, args: ProposalCreateArgs) ->
    Result<()> {
    // ...
    proposal.status = if args.draft {
        ProposalStatus::Draft {
            timestamp: Clock::get()?.unix_timestamp,
        }
    } else {
        ProposalStatus::Active {
            timestamp: Clock::get()?.unix_timestamp,
        }
    };
};
```

In `BatchAddTransaction::validate`, safeguards permit the addition of transactions to the batch only when the proposal is in a `Draft` state. However, a malicious attacker may set the proposal to `Active`, preventing the utilization of transactions created for the batch.

instructions/batch_add_transaction.rs

RUST

```
impl BatchAddTransaction<'_> {
    fn validate(&self) -> Result<()> {
        // `proposal`
        require!(
            matches!(proposal.status, ProposalStatus::Draft { .. }),
            MultisigError::InvalidProposalStatus
        );
    };
};
```

Remediation

Add a cross-program invocation for `proposal_create` at the end of `batch_create` to ensure the creation of a proposal in the `Draft` state for the batch.

Patch

Fixed in [3906ce9](#) by restricting the proposal creation for multi-signature members with the `initiate` or `vote` role.

04 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may lead to security issues in the future.

ID	Description
OS-SQD-SUG-00	Lack of validation checks for members when adding them to <code>SpendingLimit</code> .
OS-SQD-SUG-01	When adding members to the multi-signature, verify the range of their permission values.
OS-SQD-SUG-02	Clarify semantics for the incorrect passing of mutable accounts.

OS-SQD-SUG-00 | Lack Of Member Checks

Description

`config_transaction_execute` allows for the configuration of a `SpendingLimit`, regulating the member's transfer regarding the period, vault, amount, destination, and token type. When initializing the `SpendingLimit` account, it would be efficient to ensure that there are no duplicates in the members and that it is not an empty vector.

Remediation

Verify that the length of the vector is greater than zero and check for duplicates within the vector.

instructions/config_transaction_execute.rs

DIFF

```
@@ -216,8 +216,12 @@ impl<'info> ConfigTransactionExecute<'info> {
+     let mut members = members.to_vec();
+     // Make sure members are sorted.
+     members.sort();
-     SpendingLimit {
+     let spending_limit = SpendingLimit {
@@ -227,10 +231,14 @@ impl<'info> ConfigTransactionExecute<'info> {
-         members: members.to_vec(),
+         members,
+         destinations: destinations.to_vec(),
-     }
-     .try_serialize(&mut &mut spending_limit_info.data.borrow_mut()[..])?;
+     };
+     spending_limit.invariant()?;
+     spending_limit
+     .try_serialize(&mut &mut spending_limit_info.data.borrow_mut()[..])?;
```

state/spending_limit.rs

DIFF

```
@@ -62,6 +64,16 @@ impl SpendingLimit {
+     pub fn invariant(&self) -> Result<()> {
+         require!(!self.members.is_empty(), MultisigError::EmptyMembers);
+
+         // There must be no duplicate members, we make sure members are sorted when
+         ↩ creating a SpendingLimit.
+         let has_duplicates = self.members.windows(2).any(|win| win[0] == win[1]);
+         require!(!has_duplicates, MultisigError::DuplicateMember);
+
+         Ok(())
+     }
+ }
```

Patch

Fixed in [13240af](#) by checking if the vector that includes members is empty or contains duplicates.

OS-SQD-SUG-01 | Missing Permission Range Check

Description

Members of a multi-signature may be one of three types of permissions. These permissions are retained in a bitmask format within the u8 data type. Therefore, as there are three types of permissions, the value of a member's permission should be confined to a range between zero and seven.

state/multisig.rs

RUST

```
pub enum Permission {
    Initiate = 1 << 0,
    Vote = 1 << 1,
    Execute = 1 << 2,
}
```

Introducing a range check to ensure that the unutilized bit areas are not exceeded would be beneficial when initializing members in `multisig_create` and adding members through `add_member`.

Remediation

Implement a function that validates the permissions and ensure to invoke it whenever adding a member.

state/multisig.rs

DIFF

```
@@ -136,6 +136,12 @@ impl Multisig {
+     // Members must not have unknown permissions.
+     require!(
+         members.iter().all(|m| m.permissions.mask < 8), // 8 = Initiate | Vote
+         | Execute
+         MultisigError::UnknownPermission
+     );
```

error.rs

DIFF

```
@@ -60,4 +60,6 @@ pub enum MultisigError {
    #[msg("Decimals don't match the mint")]
    DecimalsMismatch,
+    #[msg("Member has unknown permission")]
+    UnknownPermission,
}
```

Patch

Fixed in [4089d5d](#) by limiting permission to less than eight.

OS-SQD-SUG-02 | Clarify CPI Semantics

Description

CPI semantics regarding the passing of mutable accounts to vault execution are a bit muddled. The underlying issue is that after the `execute_message` concludes, the mutable accounts will deserialize again, meaning any changes during the CPI will be lost.

This is partially checked via the existing `ix.program_id` checks.

```
utils/executable_transaction_message.rs  RUST

if ix.program_id == id() {
    require!(
        ix.data[..8] !=
        ↪ crate::instruction::VaultTransactionExecute::DISCRIMINATOR,
        MultisigError::ExecuteReentrancy
    );

    require!(
        ix.data[..8] !=
        ↪ crate::instruction::BatchExecuteTransaction::DISCRIMINATOR,
    );
}
```

However, these checks are too strict. As an example, a user should be allowed to call `execute` on an unrelated `Multisig`.

Remediation

Explicate denylisted accounts and ensure that none of the denylisted accounts are passed as writable.

Patch

Resolved in [#24](#) by explicitly passing in denylisted mutable accounts.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the [General Findings](#) section.

Critical	<p>Vulnerabilities that immediately lead to loss of user funds with minimal preconditions</p> <p>Examples:</p> <ul style="list-style-type: none">• Misconfigured authority or access control validation• Improperly designed economic incentives leading to loss of funds
High	<p>Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit.</p> <p>Examples:</p> <ul style="list-style-type: none">• Loss of funds requiring specific victim interactions• Exploitation involving high capital requirement with respect to payout
Medium	<p>Vulnerabilities that could lead to denial of service scenarios or degraded usability.</p> <p>Examples:</p> <ul style="list-style-type: none">• Malicious input that causes computational limit exhaustion• Forced exceptions in normal user flow
Low	<p>Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.</p> <p>Examples:</p> <ul style="list-style-type: none">• Oracle manipulation with large capital requirements and multiple transactions
Informational	<p>Best practices to mitigate future security risks. These are classified as general findings.</p> <p>Examples:</p> <ul style="list-style-type: none">• Explicit assertion of critical internal invariants• Improved input validation

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the implementation of the program requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.