certora

Security Assessment & Formal Verification Report

Squads V4 Audit

November 2024

Prepared for Squads

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Squads V4	https://github.com/Squads-Protocol/v4	d48660833989ecea3145ff726164fe640bd90696f03ce00dfd0cda258cbf2fac	Solana

Project Overview

This document describes the specification and verification of **Squads V4** using the Certora Prover and manual code review findings. The work was undertaken from **September 3** to **October 1.**

This audit is a follow-up to a previous audit performed through September - October 2023.

The methodology undertaken in this iteration of the audit was similar, and involved an initial reading of the code, focusing mostly on additions to the codebase, diffing and mapping all changes to existing code, followed by a deep-dive into the areas we deemed most error-prone.

Protocol Overview & Additions

Squads is a DeFi smart contract on the Solana blockchain, implementing a shared-custody wallet, aka a "Multisig", which can contain both fungible and non-fungible tokens. Deposits may be made into so-called vaults in the Multisig, and payments from these vaults must be approved by a quorum of members through a proposal process. In addition, the Multisig itself is governed by configuration transactions which allow changes to the configuration of the Multisig parameters subject to member consensus.

In total, 54 commits were added since the last audit.

The main applicative changes to the code, i.e. logical additions were in the following areas:

1. A new type of object called “transaction buffers” was added to the contract, with the purpose of allowing larger transactions to be serialized and later executed.
2. A new proposal cancellation flow was added, mitigating a certain edge case in which users “clawback” an already approved proposal.
3. Rent payer and creator responsibilities were separated on all accounts.
4. Global program configuration was added (for Squads administrative purposes.)
5. Spending limits are now open for participation of non-multisig members.

In addition, a few non-applicative changes were made, in this case for the purpose of optimization:

1. Rust’s `core::mem::take` is used in multiple places to prevent superflous copies.
2. The global allocator was replaced with a bottom-up bump allocator, allowing for better heap utilization before additional heap frames need to be requested from the Solana VM.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	1	1	1
Medium	0	-	-
Low	2	2	2
Informational	2	2	2
Total	5	5	5

Severity Matrix

		Likelihood		
		Low	Medium	High
Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium

Detailed Findings

ID	Title	Severity	Status
H-01	`transaction_buffer_close.rs` – Transaction buffer account may become inaccessible and unclosable	High	Fixed
L-01	`vault_transaction_create_from_buffers.rs` – Transaction buffer account may become inaccessible	Low	Fixed
L-02	`transaction_buffer_extend.rs` – Transaction buffer account may become inaccessible	Low	Fixed
I-01	`spending_limit.rs` – comment does not match check in code	Informational	Fixed
I-02	`transaction_buffer_close.rs` – comment does not match code	Informational	Fixed

High Severity Issues

H-01 Transaction buffer account may become inaccessible and unclosable		
Severity: **High**	Impact: **Medium**	Likelihood: **High**
Files: `transaction_buffer_close.rs`	Status: Fixed	

Description:

The problem is in the new code related to transaction buffers - the intention behind this new account type is to allow larger serialized transactions to be submitted and executed.

Currently, the maximum size for a serialized transaction message is bound by the maximum single Solana transaction size which is 1232 bytes.

The new transaction buffer instruction allows for creation of a container for a serialized transaction message, which can be "uploaded" to the blockchain in multiple transactions, and then "converted" to a vault transaction from storage, instead of from the transaction arguments (which are size constrained).

When creating a transaction buffer, it is always indexed by the multisig's transaction index - which is the incrementing transaction count. This is similar to how vault and config transactions are created - however as opposed to those - when creating the transaction buffer, the transaction index is incremented but not written back to the multisig.

Later, when referencing the transaction buffer for either closing, extending, or converting to a vault transaction - the current multisig transaction index is always used to derive the PDA for the transaction buffer - and not the transaction index which is saved to the transaction buffer account.

In other words it uses the shared global multisig transaction index instead of the private, per transaction buffer index.

The effect of this is that in order for this feature to properly work - all related blockchain calls (create, extend, close/create vault transaction) must be done in sequence, and with no other transactions being created in between - which is of course impossible to guarantee.

In terms of impact - this has several potential effects:

1. In reality it would be impossible to use this feature - you would have to guarantee no-one creates a transaction while you are creating and then extending your transaction buffer.
2. Once the transaction index advances - the transaction buffer account is now bricked and cannot be closed - and thus its creator forfeits its rent exemption deposit.
3. An adversary within the multisig could exploit this for griefing attacks. This is less likely since said adversary could be booted out of the multisig given enough votes.

Exploit Scenario: Alice opens a transaction buffer account with the purpose of calling extend and then convert it to a vault transaction.

Eve, an adversary within the multisig, forces the creation of a config transaction immediately after Alice opens the transaction buffer, but before she converts it to a vault transaction.

Due to the way the PDA is derived, the multisig contract can no longer access the transaction buffer account, and as such Alice can no longer close it, making her liable for it forever and forfeiting her rent deposit.

Recommendations: Use the multisig transaction index to index and access the transaction buffer PDAs in the same manner as other transaction types (vault, config, etc)

Customer's response: Fixed

Fix Review: This root cause is shared between H-01 and L-01, L-02 - but customer opted to fix only H-01 for the moment, which mitigates a large part of the impact, but still allows for griefing attacks, albeit ameliorated by the nature of the multisig which allows to vote out malicious members.

Low Severity Issues

L-01 - Transaction buffer account may become inaccessible		
Severity: **Low**	Impact: **Low**	Likelihood: **High**
Files: vault_transaction_create_from_buffers.rs	Status: Fixed	

Description: For the same reason as finding H-01, once the multisig global transaction index increases, the create from buffer instruction may no longer be invoked on the transaction buffer.

Once the global multisig transaction index increases, the PDA derived for the transaction buffer account will no longer be correct.

Exploit Scenario: Alice opens a transaction buffer account with the purpose of calling extend and then convert it to a vault transaction.

Eve, an adversary within the multisig, forces the creation of a config transaction immediately after Alice opens the transaction buffer, but before she converts it to a vault transaction.

Due to the way the PDA is derived, Alice can no longer invoke the create from buffer instruction, meaning she will have to initialize a new transaction buffer and try to perform all the actions in order without Eve interrupting.

Recommendations: Use the multisig transaction index to index and access the transaction buffer PDAs in the same manner as other transaction types (vault, config, etc)

Customer's response: {Customer feedback}

Fix Review: See H-01

L-02 - Transaction buffer account may become inaccessible		
Severity: **Low**	Impact: **Low**	Likelihood: **High**
Files: transaction_buffer_extend.rs	Status: Fixed	

Description: For the same reason as finding H-01, once the multisig global transaction index increases, the transaction buffer extend instruction may no longer be invoked on the transaction buffer.

Once the global multisig transaction index increases, the PDA derived for the transaction buffer account will no longer be correct.

Exploit Scenario: Alice opens a transaction buffer account with the purpose of calling extend and then convert it to a vault transaction.

Eve, an adversary within the multisig, forces the creation of a config transaction immediately after Alice opens the transaction buffer, but before she extends it.

Due to the way the PDA is derived, Alice can no longer invoke the transaction buffer extend instruction, meaning she will have to initialize a new transaction buffer and try to perform all the actions in order without Eve interrupting.

Recommendations: Use the multisig transaction index to index and access the transaction buffer PDAs in the same manner as other transaction types (vault, config, etc)

Customer's response: Fixed

Fix Review: See H-01

Informational Severity Issues

I-01. Comment does not match check in code

Description: In `spending_limits.rs`, the comment states “The amount must be positive”, but the code checks that the amount is not equal to zero instead.

Recommendation: Either fix the comment or the code, to reflect the actual intention - since the value is an unsigned integer, most likely the comment needs to be updated.

Customer’s response: Fixed

Fix Review: The comment was modified. The value itself is an unsigned integer.

I-02. Comment does not match code

Description: In ``transaction_buffer_close.rs` - the comment above the `transaction_buffer_close()` function states “Creates a new vault transaction”, which is incorrect.

Recommendation: Update the comment.

Customer’s response: Fixed

Fix Review: The comment was modified to reflect the function.

Formal Verification

Summary

We have written and proven two new properties P-01 and P-06. The first new property states critical properties about the correctness of the new bump allocator. The second new property ensures that a proposal has always enough allocated space. This includes cancellations. The rest of the properties show that this new version does not violate any of the properties proven in the previous version.

Verification Notations

Formally Verified	The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule.
Formally Verified After Fix	The rule was violated due to an issue in the code and was successfully verified after fixing the issue
Violated	A counter-example exists that violates one of the assertions of the rule.

General Assumptions and Simplifications

1. Prover Configuration

    - The Solana contracts were compiled to SBFv1 using the Rust compiler version 1.75 The Solana version was solana-cli 1.18.16.
    - All loops were unrolled at most 3 iterations.

2. Main assumptions for verification

- All verification harnesses call Squads instructions that take Anchor contexts as input. Thus, no serialization/deserialization code has been taken into account by the prover. The prover assumes that all Anchor accounts are initially filled with arbitrary values.
- Stubs for solana system calls
- Clock::get() returns an arbitrary value but always greater than the last returned value.
- CPIs are ignored.
- PDA computations are ignored.
- No account reallocation
- The Vec class used by members, approved, rejected, and canceled has been replaced by a simpler implementation NoResizableVec that assumes that the vectors cannot be resized. For that, the prover always proves that the length of each of those vectors is less than their capacities so the vectors never need to grow. The correctness of NoResizableVec have been proven separately.

3. Code Modifications and refactoring
    - vault_transaction_accounts_close, config_transaction_accounts_close, and batch_accounts_close have been refactored so that the verification harnesses call the functions with an already deserialized proposal.
    - vault_transaction_execute: calls to ExecutableTransactionMessage::new_validated and ExecutableTransactionMessage::execute_message are ignored by verification.
    - config_transaction_execute: ConfigAction::AddSpendingLimit and ConfigAction::RemoveSpendingLimit are ignored by verification.
    - batch_execute_transaction: same assumptions as vault_transaction_execute.
    - vault_transaction_create: the conversion from TransactionMessage to VaultTransactionMessage has been replaced with a nondeterministic VaultTransactionMessage

Formal Verification Properties

allocator

P-01. [New] The function alloc always return valid pointers	
Status: Verified	Prover options: -solanaUsePTA false -useBitVectorTheory true

Rule Name	Status	Description	Link to rule report
rule_integrity_allocator	Verified	*This rule verifies that any pointer returned by alloc is either null or always in-bounds. Moreover, for any given two pointers returned by alloc, they can never alias.*	*Report*

multisig

P-02. Any function that might modify the multisig always calls invariant and all the invariants described by invariant hold

Status: Verified

Rule Name	Status	Description	Link to rule report
multisig_invariant_create	Verified	*multisig_create_V2*	[Report]()
multisig_invariant_config	Verified	*multisig_add_member, multisig_remove_member, multisig_change_threshold, multisig_set_time_lock, multisig_set_config_authority, multisig_set_rent_collector*	[Report]()
multisig_invariant_tx_create	Verified	*vault_transaction_create, config_transaction_create, and batch_transaction_create*	[Report]()
multisig_invariant_config_tx_execute	Verified	*config_transaction_execute*	[Report]()

P-03. Any function that might modify the multisig consensus parameters always calls invalidate_prior_transactions and multisig.transaction_index is always equal to multisig.stale_transaction_index

Status: Verified

Rule Name	Status	Description	Link to rule report
invalidate_prior_transactions_config	Verified	*multisig_add_member, multisig_remove_member, multisig_change_threshold, multisig_set_time_lock, multisig_set_config_authority*	[Report]()
invalidate_prior_transactions_config_tx_execute	Verified	*config_transaction_execute*	[Report]()

P-04. Integrity of controlled multisig

Status: Verified	

Rule Name	Status	Description	Link to rule report
integrity_of_controlled_multisig	Verified	*Only the config authority can call the functions multisig_add_member, multisig_remove_member, multisig_change_threshold, multisig_set_time_lock, and multisig_set_config_authority*	[Report]()

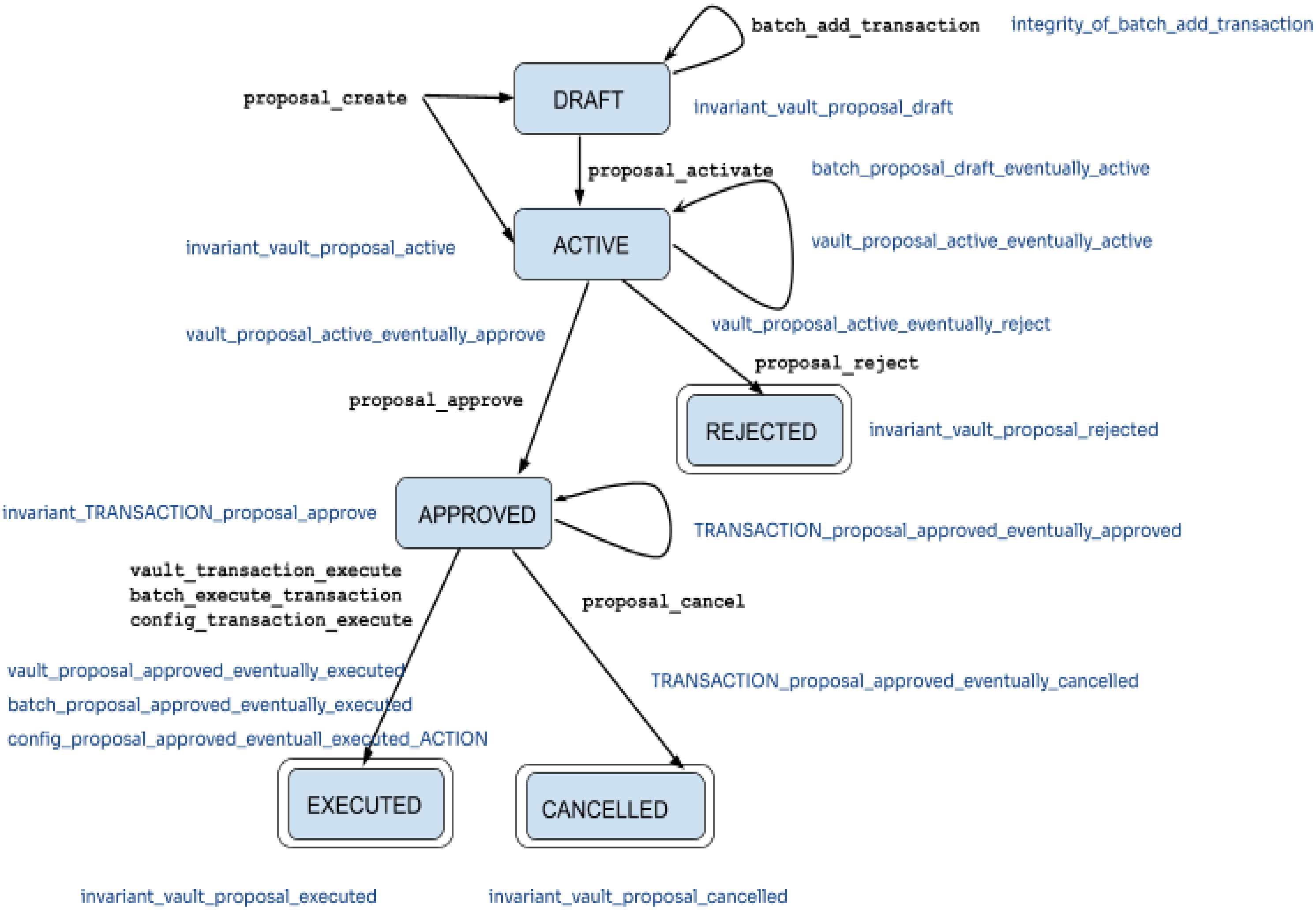
P-05. Integrity of non-controlled multisig

Status: Verified	

Rule Name	Status	Description	Link to rule report
integrity_of_noncontrolled_multisig	Verified	*The multisig config authority must be Pubkey::default()*	*Report*

proposal

The following automata shows the different states in which a proposal can be and all its valid transitions. Each state in this automata corresponds to one of the values of ProposalStatus. The rules are in blue. We attach each state and transition to one or more rules. A proposal can be initially either Draft or Active. While a proposal is active, no multisig consensus parameters can be modified, and members can vote to either approve or reject the proposal. A proposal that is Rejected is considered a final state. Once a proposal is Approved it can become only either Executed or Canceled. These two states are also final states.


batch_add_transaction
integrity_of_batch_add_transaction
proposal_create
DRAFT
invariant_vault_proposal_draft
proposal_activate
batch_proposal_draft_eventually_active
ACTIVE
invariant_vault_proposal_active
vault_proposal_active_eventually_active
vault_proposal_active_eventually_approve
vault_proposal_active_eventually_reject
proposal_reject
proposal_approve
REJECTED
invariant_vault_proposal_rejected
invariant_TRANSACTION_proposal_approve
APPROVED
TRANSACTION_proposal_approved_eventually_approved
vault_transaction_execute
batch_execute_transaction
config_transaction_execute
proposal_cancel
vault_proposal_approved_eventually_executed
batch_proposal_approved_eventually_executed
config_proposal_approved_eventuall_executed_ACTION
TRANSACTION_proposal_approved_eventually_cancelled
EXECUTED
CANCELLED
invariant_vault_proposal_executed
invariant_vault_proposal_cancelled

TRANSACTION = vault | batch | config
ACTION = add_member | remove_member | change_threshold |
Set_time_lock | add_spending_limit | remove_spending_limit

P-06. Proposal has always enough allocated space

Status: Verified

Rule Name	Status	Description	Link to rule report
vault_proposal_has_enough_allocated_space_1 **vault_proposal_has_enough_allocated_space_2**	Verified	*The number of bytes occupied by a vault proposal is less or equal than the actual allocated space for the proposal which must be always bounded by the current number of multisig members.*	Report Report

P-07. The code implements the finite automata depicted above

Status: Verified

Rule Name	Status	Description	Link to rule report
invariant_vault_proposal_draft	Verified	*If the vault proposal has status Draft then it can only be changed to Active*	Report
invariant_vault_proposal_active	Verified	*If the vault proposal has status Active then it can only be changed to Approved, Rejected, or remains Active.* *- If the proposal changed to Approved then the function*	Report

		proposal_approve was the last called function and the size of approved vector equal to the threshold of multisig, the size of rejected vector is less than cutoff of multisig, and the size of the cancelled vector is zero. *– If the proposal changed to Rejected then the function proposal_reject was the last called function and the size of rejected is greater or equal than the cutoff of the multisig.* *- If the proposal changed to Approved or Rejected then the transaction cannot be stale.* *- If the proposal remains Active then the size of cancelled is zero.*	
vault_proposal_active_eventually_approved	Verified	*(liveness) If the vault proposal has status Active then it can be eventually changed to Approved.*	Report
vault_proposal_active_eventually_rejected	Verified	*(liveness) If the vault proposal has status Active then it can be eventually changed to Rejected*	Report
vault_proposal_active_eventually_active	Verified	*(liveness) If the vault proposal has status Active then it can remain as Active, and the size of approved is less than the threshold, and the size of rejected is less than the cutoff of the multisig*	Report
invariant_vault_proposal_approved	Verified	*If the vault proposal has status Approve then it can only be changed to Canceled, Executed, or remains Approve.*	Report

		– If the proposal changed to Executed then the function vault_transaction_execute was the last called function *- If the proposal changed to Executed then the time that passed between the proposal was Approved until it was executed is greater or equal than the time_lock of the multisig.* *– If the proposal changed to Cancelled then the function proposal_cancel was the last called function and the size of cancelled is greater or equal than the multsig threshold.* *- The size of approved remains greater or equal than the threshold of the multisig (i.e., the approved vector is not modified even if the proposal is executed or got cancelled)*	
vault_proposal_approved_eventually_executed	Verified	*(liveness) If the vault proposal has status Approved then it can be eventually changed to Executed*	Report
vault_proposal_approved_eventually_cancelled	Verified	*(liveness) If the vault proposal has status Approved then it can be eventually changed to Cancelled*	Report
vault_proposal_approved_eventually_approved	Verified	*(liveness) If the vault proposal has status Approved then it can be remain as Approved.*	Report
invariant_vault_proposal_rejected	Verified	*If the vault proposal has status Rejected then the proposal status will not change anymore (final*	Report

		state). Moreover, the sizes of approved, rejected, and cancelled vectors do not change.	
invariant_vault_proposal_cancelled	Verified	*If the vault proposal has status Cancelled then the proposal status will not change anymore (final state)*	[Report]()
invariant_vault_proposal_executed	Verified	*If the vault proposal has status Executed then the proposal status will not change anymore (final state)*	[Report]()
invariant_config_proposal_approved	Verified	*If the config proposal has status Approve then it can only be changed to Canceled, Executed, or remains Approve* *- If the proposal changed to Executed then the function config_execute_transaction was the last called function* *- If the proposal changed to Executed then the time that passed between the proposal was Approved until it was executed is greater or equal than the time_lock of the multisig.* *- If the proposal changed to Executed then the transaction cannot be stale.* *- If the proposal changed to Cancelled then the function proposal_cancel was the last called function and the size of cancelled is greater or equal than the threshold of the multisig.* *- The size of approved remains greater or equal than the threshold of the multisig (i.e., the*	[Report]()

		approved vector is not modified even if the proposal is executed or got cancelled)	
config_proposal_approved_eventually_canceled	Verified	*(liveness) If the config proposal has status Approved then the proposal status can be eventually changed to Cancelled.*	Report
config_proposal_approved_eventually_approved	Verified	*(liveness) If the config proposal has status Approved then the proposal status can remain as Approved.*	Report
config_proposal_approved_eventually_executed_add_member	Verified	*(liveness) If the config proposal has status Approved then the proposal status can be eventually changed to Executed, and the last executed action is ConfigAction::AddMember.*	Report
config_proposal_approved_eventually_executed_remove_member	Verified	*(liveness) If the config proposal has status Approved then the proposal status can be eventually changed to Executed, and the last executed action is ConfigAction::RemoveMember.*	Report
config_proposal_approved_eventually_executed_set_time_lock	Verified	*(liveness) If the config proposal has status Approved then the proposal status can be eventually changed to Executed, and the last executed action is ConfigAction::SetTimeLock.*	Report
config_proposal_approved_eventually_executed_change_threshold	Verified	*(liveness) If the config proposal has status Approved then the proposal status can be eventually*	Report

		changed to Executed, and the last executed action is ConfigAction::ChangeThreshold.	
config_proposal_approved_eventually_executed_add_spending_limit	Verified	*(liveness) If the config proposal has status Approved then the proposal status can be eventually changed to Executed, and the last executed action is ConfigAction::AddSpendingLimit.*	Report
config_proposal_approved_eventually_executed_remove_spending_limit	Verified	*(liveness) If the config proposal has status Approved then the proposal status can be eventually changed to Executed, and the last executed action is ConfigAction::RemoveSpendingLimit.*	Report
batch_proposal_draft_eventually_active	Verified	*(liveness) If a batch transaction has status Draft can be eventually changed to Active*	Report
invariant_batch_proposal_approve	Verified	*If the batch proposal has status Approved then it can only be changed to Canceled, Executed, or remains Approved.* - *If the proposal changed to Executed then the time that passed between the proposal was Approved until it was executed is greater or equal than the time_lock of the multisig.* - *If the proposal changed to Executed then the size of the batch is equal to*	Report

		executed_transaction_index - *If the proposal changed to Approved then the size of the batch is greater than executed_transaction_index* - *If the proposal changed to Cancelled then the function proposal_cancel was the last called function and the size of cancelled is greater or equal than the threshold of the multisig.* - *The size of approved remains greater or equal than the threshold of the multisig (i.e., the approved vector is not modified even if the proposal is executed or got cancelled)*	
batch_proposal_approved_eventually_executed_1 **batch_proposal_approved_eventually_executed_2**	Verified	*(liveness) If a batch transaction has status Approved can be eventually changed to Executed*	[Report]() [Report]()
batch_proposal_approved_eventually_cancelled	Verified	*(liveness) If a batch transaction has status Approved can be eventually changed to Cancelled*	[Report]()
batch_proposal_approved_eventually_approved	Verified	*(liveness) If a batch transaction has status Approved can remain in Approved*	[Report]()

P-08. No double approve

Status: Verified

Rule Name	Status	Description	Link to rule report
vault_proposal_no_double_approve	Verified	*The same member cannot approve twice the same active proposal*	[Report]

P-09. No double reject

Status: Verified

Rule Name	Status	Description	Link to rule report
vault_proposal_no_double_reject	Verified	*The same member cannot reject twice the same active proposal*	[Report]

P-10. No double cancel

Status: Verified

Rule Name	Status	Description	Link to rule report
vault_proposal_no_double_cancel	Verified	*The same member cannot cancel twice the same approved proposal*	[Report]()

P-11. Integrity of close account

Status: Verified

Rule Name	Status	Description	Link to rule report
integrity_of_vault_accounts_close	Verified	*vault_transaction_accounts_close can only succeed if the proposal is in a final state (Rejected, Cancelled, or Executed) or is stale but not Approved*	[Report]()
integrity_of_vault_accounts_close_no_proposal	Verified	*If a vault transaction does not have a proposal then vault_transaction_accounts_close can only succeed if the transaction is stale.*	[Report]()
integrity_of_config_accounts_close	Verified	*config_transaction_accounts_close*	[Report]()

		can only succeed if the proposal is in final state (Rejected, Cancelled, or Executed) or is stale.	
integrity_of_config_accounts_close_no_proposal	Verified	*If a config transaction does not have a proposal then config_transaction_accounts_close can only succeed if the transaction is stale.*	[Report]()
integrity_of_batch_accounts_close_1 **integrity_of_batch_accounts_close_2**	Verified	*If the function batch_accounts_close does not revert then all its vault batch transactions have been previously closed (i.e., size of the batch is 0) and the proposal is either Executed, Rejected or Cancelled or if it is stale then it cannot be Approved.*	[Report]() [Report]()

P-12. Proposal becomes stale if multisig consensus parameter changes

Status: Verified	

Rule Name	Status	Description	Link to rule report
vault_proposal_stales_if_multisig_changes	Verified	*If a vault proposal is in an arbitrary state and then if either multisig_add_member, multisig_remove_member, multisig_change_threshold, multisig_set_time_lock, or multisig_set_config_authority is executed then the proposal becomes stale*	[Report]()

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.