F Y E O

Security Code Review of Solana Staker Vote Override

Turbine

November 2025
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

Executive Summary

Overview

Turbine engaged FYEO Inc. to perform a Security Code Review of Solana Staker Vote Override.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on August 25 - August 29, 2025, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-GOV-01 – The snapshot program is not pinned
- FYEO-GOV-02 – Unvalidated spl_vote_account allows duplicate supports
- FYEO-GOV-03 – Incomplete Vote Tallying Logic
- FYEO-GOV-04 – create_proposal does not validate spl_vote_account against merkle data
- FYEO-GOV-05 – Proposal.voting field unused in on-chain logic
- FYEO-GOV-06 – Code clarity
- FYEO-GOV-07 – Missing active_stake > 0 check
- FYEO-GOV-08 – Unchecked deserialization

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Code Review of Solana Staker Vote Override. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a public repository at https://github.com/3uild-3thos/govcontract/ with the commit hash 4db2a43cc5df76ffa2c285cf045f8d455c95b809.

An updated version was submitted with the commit hash 3abb53c6232b73afd8a17cfd151cc310de51e46c.

Remediations were provided with the commit hash c019e3cf2a6acbe3191bdcc538013459920f16f3.

Files included in the code review

```
govcontract/
└── contract/
    └── programs/
        ├── govcontract/
        │   └── src/
        │       ├── instructions/
        │       │   ├── add_merkle_root.rs
        │       │   ├── cast_vote.rs
        │       │   ├── cast_vote_override.rs
        │       │   ├── create_proposal.rs
        │       │   ├── finalize_proposal.rs
        │       │   ├── initialize_index.rs
        │       │   ├── mod.rs
        │       │   ├── modify_vote.rs
        │       │   ├── modify_vote_override.rs
        │       │   └── support_proposal.rs
        │       ├── state/
        │       │   ├── mod.rs
        │       │   ├── proposal.rs
        │       │   ├── proposal_index.rs
        │       │   ├── support.rs
        │       │   ├── vote.rs
        │       │   ├── vote_override.rs
        │       │   └── vote_override_cache.rs
        │       ├── constants.rs
        │       ├── error.rs
        │       ├── events.rs
        │       ├── lib.rs
        │       ├── merkle_helpers.rs
```

Files included in the code review

```
│       └── utils.rs
└── mock-gov-v1/
    └── src/
        ├── instructions/
        │   ├── close_meta_merkle_proof.rs
        │   ├── create_consensus_result.rs
        │   ├── init_meta_merkle_proof.rs
        │   ├── mod.rs
        │   └── verify_merkle_proof.rs
        ├── state/
        │   ├── consensus_result.rs
        │   ├── helper_structs.rs
        │   ├── meta_merkle_proof.rs
        │   └── mod.rs
        └── lib.rs
```

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review of Solana Staker Vote Override, we discovered:

- 2 findings with HIGH severity rating.
- 2 findings with MEDIUM severity rating.
- 1 finding with LOW severity rating.
- 3 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

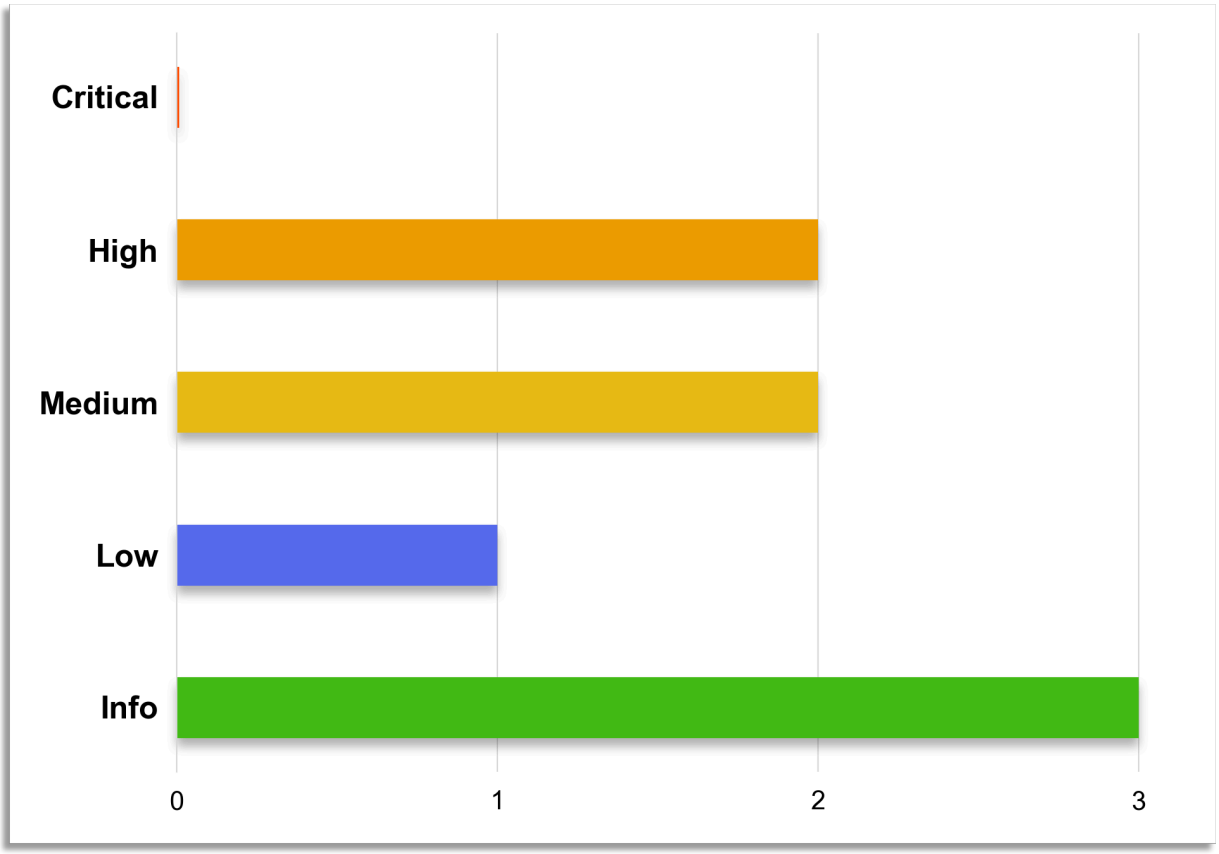

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description	Status
FYEO-GOV-01	High	The snapshot program is not pinned	Acknowledged
FYEO-GOV-02	High	Unvalidated spl_vote_account allows duplicate supports	Remediated
FYEO-GOV-03	Medium	Incomplete Vote Tallying Logic	Remediated
FYEO-GOV-04	Medium	create_proposal does not validate spl_vote_account against merkle data	Remediated
FYEO-GOV-05	Low	Proposal.voting field unused in on-chain logic	Remediated
FYEO-GOV-06	Informational	Code clarity	Remediated
FYEO-GOV-07	Informational	Missing active_stake > 0 check	Remediated
FYEO-GOV-08	Informational	Unchecked deserialization	Remediated

Table 2: Findings Overview

The Classification of vulnerabilities

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

The update significantly expands the contract's governance workflow and security around who can propose, support, and vote. Proposal creation and the support phase now rely on an external snapshot verification step so that validator identities and their stake are verified against a trusted snapshot before they can create or support proposals. The support flow accumulates verified stake from supporters and can automatically activate voting when a cluster-wide threshold is reached, while finalization of proposals is now a distinct action that records the final tallies and marks the proposal closed. The system also introduces many more runtime checks and safer arithmetic paths, plus explicit events emitted for important lifecycle moments (proposal created, supported, votes cast/modified/overridden, merkle root added, and proposal finalized), improving observability and allowing off-chain systems to track governance activity reliably.

On the voting side, votes are now weight-aware: the contract converts stake amounts and vote percentages into effective lamport counts so each vote's influence reflects verified stake. There is full support for delegators to override a validator's vote — including a caching mechanism for overrides when the validator's vote account isn't present — and the ability to modify both validator and delegator votes after they are submitted. Overall, the changes make governance more robust and verifiable by integrating snapshot-based proof checks, adding protection against arithmetic and state errors, and producing structured events so frontends and indexers can follow the complete proposal lifecycle.

The snapshot program is not pinned

Finding ID: FYEO-GOV-01
Severity: High
Status: Acknowledged

Description

Across handlers, `snapshot_program` is taken as an `UncheckedAccount` and other PDAs are only checked for `.owner == snapshot_program.key()` but `snapshot_program` itself is not required to be the known program.

Proof of Issue

File name: many
Line number: many

```
pub snapshot_program: UncheckedAccount<'info>,
```

Severity and Impact Summary

Allows passing a malicious program that can manipulate CPI behavior or bypass merkle verification. CPI calls to snapshot verifiers are high-trust operations.

Recommendation

Pin `snapshot_program` to the expected program ID.

Unvalidated spl_vote_account allows duplicate supports

Finding ID: FYEO-GOV-02
Severity: **High**
Status: **Remediated**

Description

`support_proposal` accepts any `spl_vote_account` that matches owner/size constraints but does not verify that the merkle leaf's `vote_account` equals the supplied `spl_vote_account`. This lets a single validator create multiple `Support` PDAs (using different vote accounts) and double-count the same stake.

Proof of Issue

File name: contract/programs/govcontract/src/instructions/support_proposal.rs
Line number: 42

```
#[account(
  constraint = spl_vote_account.owner == &vote_program::ID @
ProgramError::InvalidAccountOwner,
  constraint = spl_vote_account.data_len() == VoteState::size_of() @
GovernanceError::InvalidVoteAccountSize
)]
pub spl_vote_account: UncheckedAccount<'info>,
```

Severity and Impact Summary

An attacker can inflate `cluster_support_lamports` and force activation of voting by creating multiple support entries for the same logical validator stake.

Recommendation

Require `meta_merkle_leaf.vote_account == spl_vote_account.key()` or include `signer` in the Support PDA seed so a signer cannot support the same proposal more than once.

Incomplete Vote Tallying Logic

Finding ID: FYEO-GOV-03
Severity: Medium
Status: Remediated

Description

The `TallyVotes` instruction contains several issues that may lead to unexpected panics and potential DoS situations. Key concerns include: * Any validator can initiate tallying for a proposal - it is unclear if this is desired behaviour * Vote accounts are de-duplicated using a realloc(0), which leaves 0 data accounts with unreclaimable lamports * If VoteState accounts are closed between epochs, the tally function will panic * The vote count is decremented in a loop without underflow protection, which could panic if extra accounts are supplied (More than vote_count*2 accounts could be supplied)* Iterating over all votes in one transaction may exceed compute limits on large proposals. There is also a transaction size limit to consider which limits the number of accounts that can realistically be passed into the instruction

Proof of Issue

File name: contract/programs/govcontract/src/instructions/tally_votes.rs
Line number: 174, various others

```
vote.to_account_info().realloc(0, false)?;

vote_count -= 1;
```

Severity and Impact Summary

The code and logic appear unfinished and may not work as intended. Potentially preventing the program from serving its purpose.

Recommendation

1. Introduce explicit access control or document intended behaviour.
2. Handle a potentially closed VoteState account gracefully.
3. Handle additional accounts gracefully (vote_count*2).
4. Use a `tallied` flag or archive mechanism instead of `realloc(0, false)`.
5. Consider batching tally operations or introducing pagination for proposals with large voter sets. Use the tallied flag.
6. Provide a refund mechanism after the tally is complete.

create_proposal does not validate spl_vote_account against merkle data

Finding ID: FYEO-GOV-04
Severity: Medium
Status: Remediated

Description

When creating proposals, the flow does not check that the provided `spl_vote_account` maps to the snapshot/merkle data. This allows proposals to be created referencing vote accounts not tied to the merkle snapshot.

Proof of Issue

File name: contract/programs/govcontract/src/instructions/create_proposal.rs
Line number: 51

```
#[account(
    constraint = spl_vote_account.owner == &vote_program::ID @
ProgramError::InvalidAccountOwner,
    constraint = spl_vote_account.data_len() == VoteState::size_of() @
GovernanceError::InvalidVoteAccountSize
)]
pub spl_vote_account: UncheckedAccount<'info>,

...
No further checks
```

Severity and Impact Summary

Allows proposal creation with vote-account references that aren't backed by snapshot data, undermining later vote/support integrity.

Recommendation

Validate `spl_vote_account`.

Proposal.voting field unused in on-chain logic

Finding ID: FYEO-GOV-05
Severity: Low
Status: Remediated

Description

`Proposal` struct contains `pub voting: bool` which is used by frontend/tests but never enforced or checked in on-chain program logic.

Proof of Issue

File name: contract/programs/govcontract/src/state/proposal.rs
Line number: 24

```
pub voting: bool,
```

Severity and Impact Summary

Divergence between on-chain behavior and frontend/test assumptions can cause unexpected states and UI/test mismatches; may open subtle policy gaps.

Recommendation

Decide intended semantics: either enforce `voting` checks in contract handlers or remove the field and update frontends/tests accordingly.

Code clarity

Finding ID: FYEO-GOV-06
Severity: **Informational**
Status: **Remediated**

Description

The codebase uses hard-coded numeric values (magic numbers), repeated validation patterns, and redundant `Result` wrapping, all of which reduce readability, consistency, and maintainability.

Proof of Issue

File name: contract/programs/govcontract/src/state/proposal.rs
Line number: 7

```
#[max_len(50)]
pub title: String,

#[max_len(250)]
pub description: String,
```

File name: contract/programs/govcontract/src/instructions/create_proposal.rs
Line number: 54

```
require!(title.len() <= 50, GovernanceError::TitleTooLong);
require!(description.len() <= 250, GovernanceError::DescriptionTooLong);
```

File name: contract/programs/govcontract/src/utils.rs
File name: contract/programs/govcontract/src/instructions/cast_vote.rs
File name: contract/programs/govcontract/src/instructions/modify_vote.rs
File name: contract/programs/govcontract/src/instructions/tally_votes.rs

```
10_000
```

File name: various

```
let vote_account_data = self.spl_vote_account.data.borrow();
let version = u32::from_le_bytes(
    vote_account_data[0..4]
        .try_into()
        .map_err(|_| GovernanceError::InvalidVoteAccount)?,
);
require!(version <= 2, GovernanceError::InvalidVoteAccount);


let node_pubkey =
    Pubkey::try_from(&vote_account_data[4..36]).map_err(|_|
GovernanceError::InvalidVoteAccount)?;

require_keys_eq!(
    node_pubkey,
    self.signer.key(),
    GovernanceError::InvalidVoteAccount
);
```

This check is used in many places and duplicated for each instance.

File name: contract/programs/govcontract/src/lib.rs

```
ctx.accounts.support_proposal(&ctx.bumps)?;
Ok(())
```

All handlers use this pattern, but could just return the handler call.

Severity and Impact Summary

These issues are not security concerns but will degrade the maintainability of the code base.

Recommendation

1. Replace magic numbers with named constants.
2. Abstract `VoteAccount` parsing.
3. Simplify redundant return patterns

Missing active_stake > 0 check

Finding ID: FYEO-GOV-07
Severity: **Informational**
Status: **Remediated**

Description

The handler `support_proposal` adds `meta_merkle_leaf.active_stake` to `cluster_support_lamports` without verifying the stake is positive. Other handlers explicitly require `active_stake > 0`.

Proof of Issue

File name: contract/programs/govcontract/src/instructions/support_proposal.rs
Line number: 113

```
self.proposal.add_cluster_support(meta_merkle_leaf.active_stake)?;
```

Severity and Impact Summary

Zero leaf stake could be accepted, this is inconsistent with other handlers.

Recommendation

Require `meta_merkle_leaf.active_stake > 0` before adding support.

Unchecked deserialization

Finding ID: FYEO-GOV-08
Severity: **Informational**
Status: **Remediated**

Description

Handlers do not check the discriminator on `consensus_result` & `meta_merkle_proof` accounts. While this should be checked in the CPI call, this just assumes accounts are of the correct type.

Proof of Issue

File name: many
Line number: many

```
pub consensus_result: UncheckedAccount<'info>,
pub meta_merkle_proof: UncheckedAccount<'info>,

...

let consensus_result_data = self.consensus_result.try_borrow_data()?;
let consensus_result =

try_from_slice_unchecked::<ConsensusResult>(&consensus_result_data[ANCHOR_DISCRIMINATOR.
.])?;
```

Severity and Impact Summary

Deserialization assumptions may cause silent mismatches or parsing errors.

Recommendation

Validate expected discriminators.

Our Process

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.