Jito Labs

Security Assessment

February 10th, 2024 — Prepared by OtterSec

Harrison Green hgarrereyn@osec.io

Robert Chen notdeghost@osec.io

Table of Contents

01 — Executive Summary

Overview

Jito Labs engaged OtterSec to assess the `validator-history` program. This assessment was conducted between January 1st and February 7th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 5 findings throughout this audit engagement.

In particular, we identified a vulnerability allowing validators to potentially overwrite their stored gossip information by self-invoking the instruction, resulting in including forged data in the validator history (OS-JLB-ADV-00).

We also provided recommendations regarding the importance of sequential epoch initialization when updating stake history backward, beginning from the most recent epoch. This ensures that older slots are not inadvertently skipped during the initialization of a new account from scratch (OS-JLB-SUG-00). Additionally, we emphasized adding comments or assertions to prevent unintended omission of values in the validator history entry structure, ensuring that default values are intentionally set (OS-JLB-SUG-01). Furthermore, we suggested specific enhancements to enhance code quality and readability (OS-JLB-SUG-03).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/jito-foundation/stakenet. This audit was performed against commit fc34c25.

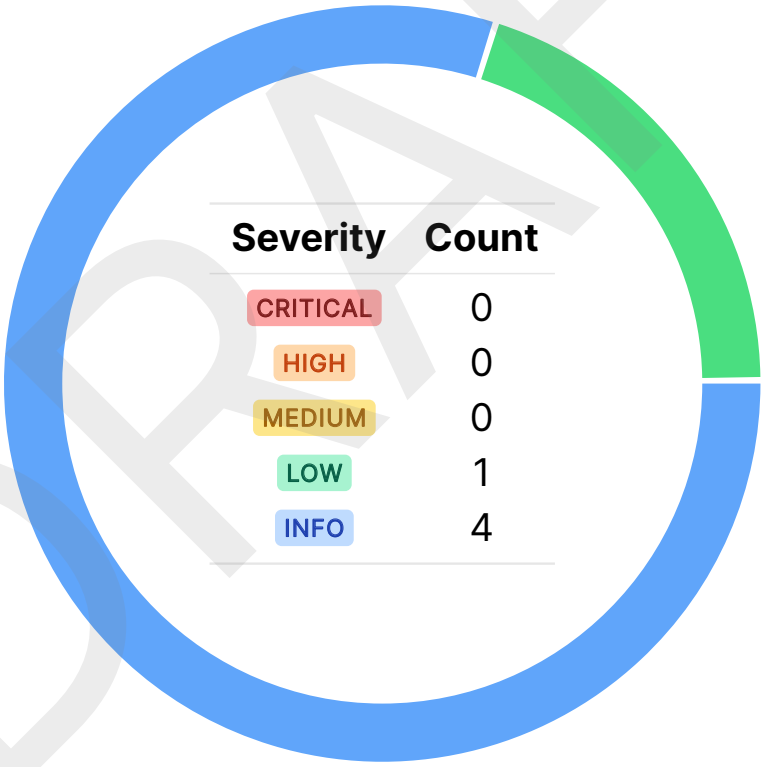
A brief description of the programs is as follows:

Name	Description
validator-history	A component of Jito StakeNet, designed to maintain an on-chain registry of validated Solana validator data, consolidating various fields accessible to the Solana runtime into a single account, facilitating the seamless integration of diverse fields within on-chain programs.

03 — Findings

Overall, we reported 5 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 aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	0
HIGH	0
MEDIUM	0
LOW	1
INFO	4

04 — 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-JLB-ADV-00	LOW	TODO	`CopyGossipContactInfo` allows validators to potentially overwrite their stored gossip information by self-nvoking the instruction, resulting in including forged data in the validator history.

Risk Of Self-Invocation LOW OS-JLB-ADV-00

Description

Validators may overwrite their stored gossip information with incorrect or malicious data at the end of every epoch. `CopyGossipContactInfo`, as it stands, allows validators to independently invoke this instruction and update their historical gossip data in the `ValidatorHistory` account. This may result in the persistence of inaccurate or malicious information in the historical records when invoked at the end of each epoch, causing the historical gossip data stored in `ValidatorHistory` to represent the gossip information at the end of each epoch rather than the comprehensive data throughout the entire epoch.

>_ src/instructions/copy_gossip_contact_info.rs rust

```rust
#[derive(Accounts)]
pub struct CopyGossipContactInfo<'info> {
    #[account(
        mut,
        seeds = [ValidatorHistory::SEED, vote_account.key().as_ref()],
        bump,
    )]
    pub validator_history_account: AccountLoader<'info, ValidatorHistory>,
    /// CHECK: Safe because we check the vote program is the owner.
    #[account(owner = solana_program::vote::program::ID.key())]
    pub vote_account: AccountInfo<'info>,
    /// CHECK: Safe because it's a sysvar account
    #[account(address = sysvar::instructions::ID)]
    pub instructions: UncheckedAccount<'info>,
    #[account(mut)]
    pub signer: Signer<'info>,
}
```

Remediation

Ensure proper authentication of this instruction to prevent validators from invoking it independently without broadcasting the new gossip information.

05 — 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 result in security issues in the future.

ID	Description
OS-JLB-SUG-00	Ensure sequential epoch initialization when calling `UpdateStakeHistory` backward, starting from the most recent epoch.
OS-JLB-SUG-01	Addition of comments or assertions to ensure that fields in the `ValidatorHistoryEntry` structure are not accidentally left unset with default values (`T::max`).
OS-JLB-SUG-02	Monotonically increasing epochs in circular buffer wrap around in cases when the result exceeds the type's maximum value.
OS-JLB-SUG-03	Suggestions regarding improving the code quality and readability.

Sequential Epoch Initialization OS-JLB-SUG-00

Description

While initializing a new account from scratch, especially when dealing with historical stake history, calling `UpdateStakeHistory` starts from the most recent epoch and works backward, which may prevent older slots (epochs) from being filled. Initiating the initialization process from the most recent epoch and moving backward may introduce a risk of overlooking or not populating stake information for earlier epochs.

Remediation

Ensure the administrator (`stake_authority`) validates that the initialization starts from the oldest epoch when initializing historical stake history. Initiating the process from the oldest epoch ensures the proper initialization and recording of stake information for each epoch, preventing gaps or missing data in the historical record.

Clarify Default Value Semantics OS-JLB-SUG-01

Description

Currently, the program initializes the default values of fields within the `ValidatorHistoryEntry` structure to the maximum value of their respective types (`T::max`). This approach may introduce confusion or unintended outcomes, particularly if the default value is valid for the field.

>_ src/instructions/copy_gossip_contact_info.rs rust

```rust
impl Default for ValidatorHistoryEntry {
    fn default() -> Self {
        Self {
            activated_stake_lamports: u64::MAX,
            epoch: u16::MAX,
            mev_commission: u16::MAX,
            epoch_credits: u32::MAX,
            commission: u8::MAX,
            client_type: u8::MAX,
            [...]
        }
    }
}
```

In Rust, `max_value` of the `std::cmp::Ord` trait returns the maximum value a type may hold. However, sometimes, utilizing the maximum value as a default may cause unexpected behavior. If the maximum value is a valid value for a field, it may be confusing to determine whether a field has been intentionally set to its maximum value or is genuinely unset.

Remediation

Include comments explaining the deliberate utilization of `T::max` as default values, or consider incorporating assertions within the setters. This practice helps clarify the intentional choice of default values. It safeguards against inadvertent settings to their maximum values, mitigating potential issues arising from misconceptions about default values and ensuring the intended behavior of the code.

Epoch Wrapping

OS-JLB-SUG-02

Description

Currently, it is the assumption that the validator history `CircBuf` (circular buffer) should contain entries with monotonically increasing epochs. The particular focus is on `cast_epoch` and the potential implications on the system when epochs wrap around. In cases where the result exceeds the maximum value representable by a `u16`, the epochs undergo wrap-around.

Remediation

Replace `cast_epoch` with a method that asserts whether the epoch is within the bounds representable by a `u16` instead of allowing epochs to wrap around.

Code Maturity

OS-JLB-SUG-03

Description

1. While omitting the `bump` parameter is not a vulnerability due to recent anchor versions ensuring the use of the canonical program-derived address, specifying the bump whenever possible is recommended.
2. Replace the magic offsets in `CopyGossipContactInfo` with the constants defined in `solana/sdk/src/ed25519_instruction.rs`. This improves code readability, making it easier to understand the structure of serialized data, and ensures consistency by using well-named constants.

>_ solana/sdk/src/ed25519_instruction.rs rust

```rust
pub const PUBKEY_SERIALIZED_SIZE: usize = 32;
pub const SIGNATURE_SERIALIZED_SIZE: usize = 64;
pub const SIGNATURE_OFFSETS_SERIALIZED_SIZE: usize = 14;
// bytemuck requires structures to be aligned
pub const SIGNATURE_OFFSETS_START: usize = 2;
pub const DATA_START: usize = SIGNATURE_OFFSETS_SERIALIZED_SIZE + SIGNATURE_OFFSETS_START;
```

3. The comment delineating the return value of `epoch_range` is incorrect, as it asserts that it yields `None` if either `start_epoch` or `end_epoch` is not in `CircBuf`, whereas it actually returns an empty vector.

Remediation

1. Ensure to specify the `bump` for `ValidatorHistoryAccount` in `CopyGossipContactInfo`, `CopyVoteAccount`, `UpdateMevComission`, and `UpdateStakeHistory`. A similar approach is feasible for `TipDistributionAccount` in `UpdateMevCommission`. Note that this is not possible in `ReallocValidatorHistoryAccount` without storing the `bump` during `init`.
2. Implement the above recommendation.
3. Refactor the comment to reflect the actual value returned by `epoch_range`.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- 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 program's implementation 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 comprehensively understand 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.