Security Audit - Marinade Canonical Stake and Deposit Fee Upgrade

conducted by Neodyme AG

Lead Auditor:	Jasper Slusallek
Second Auditor:	Tobias Bucher
Administrative Lead:	Jasper Slusallek

June 18, 2026



Table of Contents

1 Executive Summary **3**

2 Introduction **4**
Summary of Findings 4

3 Scope **5**

4 Project Overview **6**
Functionality 6
On-Chain Data and Accounts 6
Instructions 7

5 Findings **8**

Appendices

A About Neodyme **9**

B Methodology **10**
Select Common Vulnerabilities 10

C Vulnerability Severity Rating **12**

1 | Executive Summary

Neodyme was engaged by **Marinade** to conduct a security audit of changes to Marinade's on-chain liquid staking program during May 2026. A prior, initial audit of the on-chain liquid staking program was performed in 2023 and included comprehensive threat modeling, architectural and economical analysis.

Since the audit focused on two smaller changes, the economical analyses and threat modeling of the original report remain valid.

Consistent with the previous audit, the changes appear well-implemented and build upon the clean design of the contract.

Due to the complex threat model for staking contracts, the scope of this audit included both implementation security and the security of the overall design. According to Neodyme's Rating Classification, **0 security-relevant** and **0 informational** issues were found. The number of findings identified throughout the audit, grouped by severity, can be seen in Figure 1.

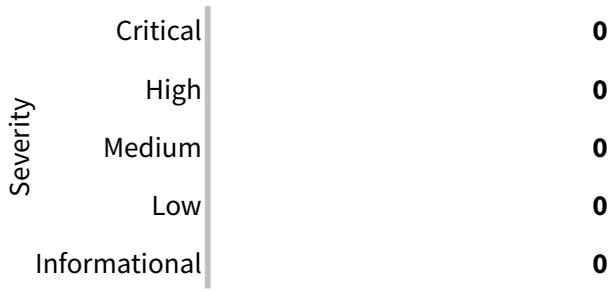


Figure 1: Overview of Findings

Neodyme delivered the Marinade team a list of nitpicks and additional notes that are not part of this report.

2 | Introduction

During May 2026, Marinade engaged Neodyme to do a detailed security analysis of two changes to their on-chain liquid staking program.

A team of one senior and one junior security researcher from Neodyme conducted a full audit of the contract in the third week of May. The lead auditor was the same as in the previous audit. The lead auditor has a long track record of finding critical and other vulnerabilities in Solana programs, as well as in Solana's core code itself, and has extensive knowledge about the intricacies of Solana's stake program, which Marinade interacts with.

The audit focused on the impact of the changes to the contract's technical security and overall design.

Summary of Findings

All found issues were **quickly remediated**. In total, the audit revealed:

0 critical • **0** high-severity • **0** medium-severity • **0** low-severity • **0** informational

issues.

3 | Scope

The project audit's scope comprised of two major components:

- Primarily, how the changes affect the **implementation** security of the contract's source code.
- Additionally, changes to the security of the **overall design**.

Neodyme considers the the two changes in the source code, located at https://github.com/marinade-finance/liquid-staking-program, in scope for this audit. Third-party dependencies are not in scope. Marinade only relies on the Anchor library and `solana-security-txt`, both of which are well-established.

The three relevant source code revisions are:

- `e7e0bfd76c5c6e4db96b558762fd4896b1a191ed` · Implementation of canonical stake accounts
- `fc217cdf669849afd4ceb9b4f9304034d9b4457b` · Implementation of deposit fees
- `0f031c4e210a47080b09ae04731ae0984256bf4a` · Last reviewed commit

4 | Project Overview

This section briefly outlines Marinade Canonical Stake and Deposit Fee Upgrade's functionality, design, and architecture, followed by a detailed discussion of all related authorities.

Functionality

Canonical Stake

Marinade introduces the concept of canonical stake accounts. These are PDAs of the marinade program, each of which is specific to one validator. The goal of the change is that all stake of a validator is concentrated at that address.

In order to achieve this, the program's stake account merge logic is modified. Before the change, any two mergeable stake accounts that are controlled by Marinade and delegated to the same validator could be merged by the cranker bot. After the change, stake accounts can only be merged into the canonical stake account of the validator that both stake accounts are delegated to.

This change also introduces a new instruction, which simply initializes the canonical stake account at the PDA address from an existing stake account of the marinade program.

Deposit Fee

Additionally, deposit fees are introduced. When SOL or a stake account gets deposited into Marinade's liquid staking program, liquid staking tokens (mSOL) are minted to the depositor. The deposit fee for SOL and stake accounts can be configured independently, and it is subtracted from the amount of deposited SOL before the amount of liquid staking tokens is calculated.

The program ensures that the deposit fees – both for SOL and stake deposits – are less or equal to 0.2%.

On-Chain Data and Accounts

Canonical Stake

For each $validator$, the contract has a potential canonical stake account at the PDA with the seed $state.addr \parallel validator.addr \parallel$ '`canonical_stake`'. Except for its address, this is a normal stake account owned by the program.

As the `Pubkey::find_program_address` documentation notes, it is important for different PDAs to use different seeds.

The contract has a total of 9 PDA kinds. As the new PDA has a unique seed length, it cannot clash with any of the existing 8 PDAs:

Len	Seed	Description
39	$state.addr$ ‖ '`deposit`'	stake deposit authority
39	$state.addr$ ‖ '`liq_sol`'	sol leg address
39	$state.addr$ ‖ '`reserve`'	reserve address
39	$state.addr$ ‖ '`st_mint`'	msol mint authority
40	$state.addr$ ‖ '`liq_mint`'	lp mint authority
40	$state.addr$ ‖ '`withdraw`'	stake withdraw authority
52	$state.addr$ ‖ '`liq_st_sol_authority`'	msol leg authority
79	$state.addr$ ‖ $validator.addr$ ‖ '`canonical_stake`'	**canonical stake**
80	$state.addr$ ‖ '`unique_validator`' ‖ $validator.addr$	duplication flag

Note that the canonical stake seed and the duplication seed could collide for different validators if `validator.addr` were variable-length. However, since `canonical_stake` (length 15) and `unique_validator` (length 16) are of different length and since `validator.length` is fixed-length, they cannot collide for seed length reasons. Nonetheless, Neodyme reported this footgun to marinade as a nitpick. Marinade acknowledged it.

Deposit Fee

Two new fields are added to the global `State` account of Marinade:

```
1       pub deposit_sol_fee: FeeCents,
2       pub deposit_stake_account_fee: FeeCents,
```

Instructions

Canonical Stake

There's a new `CreateCanonicalStake` cranking instruction that moves an existing stake account for a validator to the canonical stake account address for that validator, failing if the canonical stake account already exists. This moves works by splitting all lamports in the existing source into a newly created stake account at the canonical stake address.

The `MergeStakes` cranking instruction was modified to only allow merging into the canonical stake account.

Deposit Fee

The existing `Deposit` and `DepositStakeAccount` have been modified to deduct a fee before issuing the mSOL tokens.

5 | Findings

This section normally outlines all of our findings, but this audit had none.

Neodyme delivered the Marinade team a list of nitpicks and additional notes which are not part of this report.

A | About Neodyme

Security is difficult.

To understand and break complex things, you need a certain type of people. People who thrive in complexity, who love to play around with code, and who don't stop exploring until they fully understand every aspect of it. That's us.

Our team never outsources audits. Having found over 80 High or Critical bugs in Solana's core code itself, we believe that Neodyme hosts the most qualified auditors for Solana programs. We've also found and disclosed critical vulnerabilities in many of Solana's top projects and have responsibly disclosed issues that could have resulted in the theft of over $10B in TVL on the Solana blockchain.

All of our team members have a background in competitive hacking. During such hacking competitions, called CTFs, we competed and collaborated while finding vulnerabilities, breaking encryption, reverse engineering complicated algorithms, and much more. Through the years, many of our team members have won national and international hacking competitions, and keep ranking highly among some of the hardest CTF events worldwide. In 2020, some of our members started experimenting with validators and became active members in the early Solana community. With the prospect of an interesting technical challenge and bug bounties, they quickly encouraged others from our CTF team to look for security issues in Solana. The result was so successful that after reporting several bugs, in 2021, the Solana Foundation contracted us for source code auditing. As a result, Neodyme was born.



B | Methodology

We are not checklist auditors.

In fact, we pride ourselves on that. We adapt our approach to each audit, investing considerable time into understanding the program upfront and exploring its expected behavior, edge cases, invariants, and ways in which the latter could be violated.

We use our uniquely deep knowledge of Solana internals, and our years-long experience in auditing Solana programs to find bugs that others miss. We often extend our audit to cover off-chain components in order to see how users could be tricked or the contract affected by bugs in those components.

Nonetheless, we also have a list of common vulnerability classes, which we always exhaustively look for. We provide a sample of this list here.

Select Common Vulnerabilities

Our most common findings are still specific to Solana itself. Among these are vulnerabilities such as the ones listed below:

- Insufficient validation, such as:
  - Missing ownership checks
  - Missing signer checks
  - Signed invocation of unverified programs
  - Account confusions
  - Missing freeze authority checks
  - Insufficient SPL account verification
  - Dangerous user-controlled bumps
  - Insufficient Anchor account linkage
- Account reinitialization vulnerabilities
- Account creation DoS
- Redeployment with cross-instance confusion
- Missing rent exemption assertion
- Casting truncation
- Arithmetic over- or underflows
- Numerical precision and rounding errors
- Anchor pitfalls, such as accounts not being reloaded
- Non-unique seeds
- Issues arising from CPI recursion
- Log truncation vulnerabilities
- Vulnerabilities specific to integration of Token Extensions, for example unexpected external token hook calls

Apart from such Solana-specific findings, some of the most common vulnerabilities relate to the general logical structure of the contract. Specifically, such findings may be:

- Errors in business logic
- Mismatches between contract logic and project specifications
- General denial-of-service attacks
- Sybil attacks
- Incorrect usage of on-chain randomness
- Contract-specific low-level vulnerabilities, such as incorrect account memory management
- Vulnerability to economic attacks
- Allowing front-running or sandwiching attacks

Miscellaneous other findings are also routinely checked for, among them:

- Unsafe design decisions that might lead to vulnerabilities being introduced in the future
    - Additionally, any findings related to code consistency and cleanliness
- Rug pull mechanisms or hidden backdoors

Often, we also examine the authority structure of a contract, investigating their security as well as the impact on contract operations should they be compromised.

Over the years, we have found hundreds of high and critical severity findings, many of which are highly nontrivial and do not fall into the strict categories above. This is why our approach has always gone way beyond simply checking for common vulnerabilities. We believe that the only way to truly secure a program is a deep and tailored exploration that covers all aspects of a program, from small low-level bugs to complex logical vulnerabilities.

C | Vulnerability Severity Rating

We use the following guideline to classify the severity of vulnerabilities. Note that we assess each vulnerability on an individual basis and may deviate from these guidelines in cases where it is well-founded. In such cases, we always provide an explanation.

Severity	Description
CRITICAL	Vulnerabilities that will likely cause loss of funds. An attacker can trigger them with little or no preparation, or they are expected to happen accidentally. Effects are difficult to undo after they are detected.
HIGH	Bugs that can be used to set up loss of funds in a more limited capacity, or to render the contract unusable.
MEDIUM	Bugs that do not cause direct loss of funds but that may lead to other exploitable mechanisms, or that could be exploited to render the contract partially unusable.
LOW	Bugs that do not have a significant immediate impact and could be fixed easily after detection.
INFORMATIONAL	Bugs or inconsistencies that have little to no security impact, but are still noteworthy.

Additionally, we often provide the client with a list of nit-picks, i.e. findings whose severity lies below Informational. In general, these findings are not part of the report.

Neodyme AG

Dirnismaning 55
Halle 13
85748 Garching
Germany

E-Mail: contact@neodyme.io

https://neodyme.io