Security Audit – Marinade Native Staking Proxy

conducted by Neodyme AG

Lead Auditor:	Sebastian Fritsch
Second Auditor:	Mathias Scherer

March 25th 2024

Table of Contents

1 Introduction

During February and March of 2024, Marinade engaged Neodyme to conduct a detailed security analysis of their on-chain native staking proxy program. Two security researchers from Neodyme, Sebastian Fritsch and Mathias Scherer, conducted independent full audits of the contract between the 26th of February and the 5th of March 2024. Both auditors have a long track record of finding critical and other vulnerabilities in Solana programs, as well as in Solana's core code itself, and have extensive knowledge about the intricacies of Solana's stake program, which the Marinade native staking proxy interacts with.

The audit focused on the contract's technical security. In the following sections, we present our findings and discuss worst-case scenarios for authority compromise and provide some general notes for considerations that may be useful in the future.

Neodyme would like to emphasize the high quality of Marinade's work. Marinade's team always responded **quickly and competently** to nitpicks and discussions of any kind. Their **in-depth knowledge** of Solana development was apparent during all stages of the cooperation, including excellent and crucial knowledge of Solana's stake program. Evidently, Marinade invested significant effort and resources into their product's security. The contract's source code has no unnecessary dependencies, relying mainly on the well-established Anchor framework.

Summary of Findings

We are happy to confirm that **no issues** were found.

2 | Scope

The contract audit's scope comprised one major and one minor component:

- Primarily, the **Implementation** security of the contract's source code.
- Additionally, the security of the **overall design**, including the off-chain bot.

Neodyme considers the source code, located at https://github.com/marinade-finance/native-staking, in scope for this audit. Third-party dependencies are not in scope. Marinade only relies on the Anchor library which is well-established.

The relevant source code revisions are:

- `a7cb059d7c2538728ae3fdfbb91738e35d7034c6` • Last reviewed revision

3 Project Overview

This section briefly outlines Marinade's native staking proxy functionality, design, and architecture, followed by a detailed discussion of all related authorities.

Functionality

The native staking proxy allows SOL owners to have their Solana stake accounts managed by Marinade without the smart contract risk present in, for example, stake pools.

For this, the native staking proxy utilizes Solana's native Stake program, which provides the base layer for stake accounts and validator delegations on Solana. The Solana Stake program offers two different types of authorities for Stake accounts:

- The `staker`, who can perform operations like `Delegate`, `Redelegate`, `Merge`, but cannot `Withdraw`.
- The `withdrawer`, who can `Withdraw` SOL from an undelegated Stake account and can set the `staker` as well as a new `withdrawer` authority.

By setting the `staker` authority, a user can authorize another entity to manage and delegate the stake to, for example, the most performant validators. The native Stake Program guarantees that ownership of the SOL is never lost but always remains in the hands of the `withdrawer` authority.

The native staking proxy leverages this functionality. A user can authorize the proxy to manage their stake for them. This is done via an on-chain program, which mostly acts as a pass-through for most instructions but also restricts some instruction arguments to harden the security of the protocol.

In addition to the on-chain Staking proxy, Marinade operates an off-chain bot, responsible for tracking the current validators' performance and redelegating stake to the most performant validators while keeping the network decentralized.

On-Chain Accounts and Authorities

Marinade's native staking proxy uses a single `Root` account per proxy instance to store all relevant data:

```
#[account]
pub struct Root {
    pub admin: Pubkey,
    pub operator: Pubkey,
    pub alternate_staker: Pubkey,
    pub bumps: Bumps,
}
```

The `Root` specifies the `admin` and the `operator`, which are the two authorities present in the proxy. The listing of instructions below summarizes which instruction is permissioned for which authority. The `alternate_staker` field of the `Root` can be set by the `admin` and stores the prospective staker if the `SwitchStaker` instruction is called.

Additionally, there exists the staker PDA (Program Derived Address), which is derived per proxy instance via the following seeds: `[Root::STAKER_SEED, root.key(), root.bumps.staker]`. To authorize Marinade's native staking proxy to manage a stake account, a user has to set the staker authority to Marinade's staker PDA. By using a smart contract to manage the staker authority instead of an off-chain hot wallet, the proxy limits the security implications of a private-key compromise. If the `operator` wallet gets compromised, the attacker can only execute a subset of Stake program instructions as described below and the proxy restricts the `staker` authority to an admin-defined `alternate_staker`, thereby preventing a total loss of the `staker` authority. If, in contrast, the `staker` were directly an off-chain wallet, Marinade could not guarantee those restrictions.

As the `admin` authority is not used for active stake account management, Marinade decided to manage it via the Marinade DAO Council. The `operator` authority is used by the off-chain bot and therefore is not managed via the DAO.

Instructions

The contract has a total of 10 instructions, which we briefly summarize here.

Instruction	Category	Summary
InitRoot	Permissionless	Initializes a new on-chain proxy instance. The `Root` account holds information about the `admin`, `operator`, `alternate_staker`, and bump seeds of associated keys.
SetOperator	Admin-Only	Sets a new operator.
SetAdmin	Admin-Only	Sets a new admin.
SetAlternateStaker	Admin-Only	Sets a new alternate staker.
Merge	Operator-Only	Merges two proxy-managed stake accounts into one. The invariants of the native Solana stake program apply.
Split	Operator-Only	Splits a proxy-managed stake account into two. The second account gets newly created and has the same authority structure as the original account. The invariants of the native Solana stake program apply.
Deactivate	Operator-Only	Deactivates the stake of a proxy-managed stake account. The invariants of the native Solana stake program apply.
Delegate	Operator-Only	Delegates the stake of a proxy-managed stake account to a specific validator. The invariants of the native Solana stake program apply.
Redelegate	Operator-Only	Redelegates activated stake from a proxy-managed stake account to a new proxy-managed stake account. The invariants of the native Solana stake program apply.
SwitchStaker	Operator-Only	Switches the `stake` authority of a proxy-managed stake account to the alternate staker defined in the `Root` account.

4 | Findings

We are happy to confirm that **no issues** were found in the native staking proxy by Marinade.

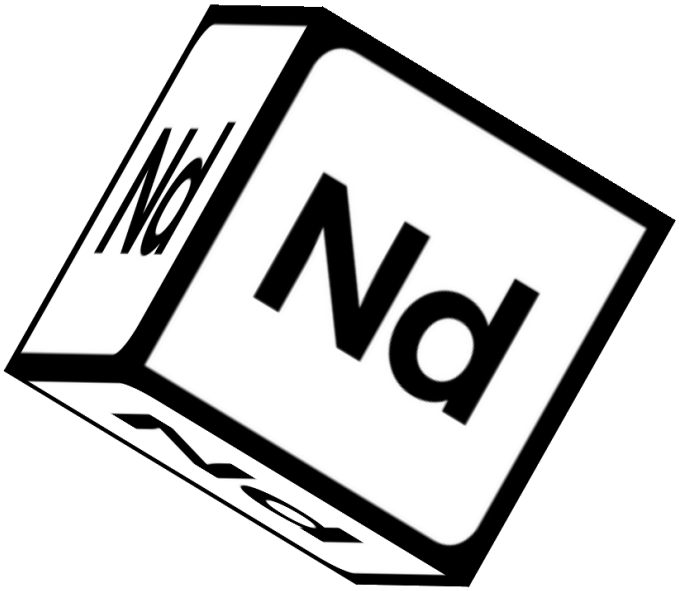
A | About Neodyme

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

Neodyme prides itself on not being a checklist auditor. 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 even 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 below.

- Rule out common classes of Solana contract vulnerabilities, such as:
  - Missing ownership checks
  - Missing signer checks
  - Signed invocation of unverified programs
  - Solana account confusions
  - Redeployment with cross-instance confusion
  - Missing freeze authority checks
  - Insufficient SPL account verification
  - Missing rent exemption assertion
  - Casting truncation
  - Arithmetic over- or underflows
  - Numerical precision errors
- Check for unsafe design decisions that might lead to vulnerabilities being introduced in the future
- Check for any other, as-of-yet unknown classes of vulnerabilities arising from the structure of the Solana blockchain
- Ensure that the contract logic correctly implements the project specifications
- Examine the code in detail for contract-specific low-level vulnerabilities
- Rule out denial of service attacks
- Rule out economic attacks
- Check for instructions that allow front-running or sandwiching attacks
- Check for rug pull mechanisms or hidden backdoors

C | Vulnerability Severity Rating

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.

Info Bugs or inconsistencies that have little to no security impact.

Neodyme AG

Dirnismaning 55
Halle 13
85748 Garching
E-Mail: contact@neodyme.io

https://neodyme.io