Security Audit – Marinade Liquid Staking

conducted by Neodyme AG

Lead Auditor:	Jasper Slusallek
Second Auditor:	Simon Klier
Administrative Lead:	Thomas Lambertz

October 21st 2023

Table of Contents

1 Executive Summary

Neodyme audited **Marinade's** on-chain liquid Solana staking program during August 2023.

Due to the complex threat model for staking contracts, the scope of this audit included both implementation security and an economic analysis. The auditors found that Marinade's staking program comprised a clean design and far-above-standard code quality. According to Neodymes Rating Classification, **no vulnerabilities above low-severity** were found. The number of findings identified throughout the audit, grouped by severity, can be seen in Figure 1.

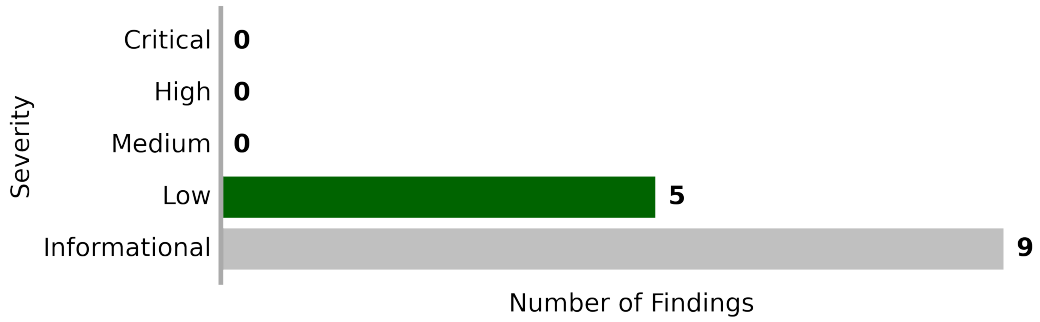

Figure 1: Overview of Findings

The auditors reported all findings to the Marinade developers, who addressed them promptly. The security fixes were verified for completeness by Neodyme. In addition to these findings, Neodyme delivered the Marinade team a list of nit-picks and additional notes that are not part of this report.

2 | Introduction

During the Summer of 2023, Marinade engaged Neodyme to do a detailed security analysis of their on-chain liquid Solana staking program. Two senior security researchers from Neodyme, Simon Klier and Jasper Slusallek, conducted independent full audits of the contract between the 21st of August and the 4th of September 2023. 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 Marinade interacts with.

The audit focused on the contract's technical security, as well as possible economic attacks. In the following sections, we present our findings and discuss worst-case scenarios for authority compromise, discuss economic attack patterns, 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 **competent** to findings of any kind. Their **in-depth knowledge** of liquid-staking programs 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. Some technical debt was apparent in the contract, but only to a minor extent. Their **code quality is far above standard**, as the code is well documented, naming schemes are clear, and the overall architecture of the program is **clean and coherent**. 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 above low severity** were found. All found issues were **quickly remediated**. In total, the audit revealed:

0 critical • **0** high-severity • **0** medium-severity • **5** low-severity • **9** informational

issues. In our opinion, the most significant current risks to Marinade users are the effects of bugs being introduced into Solana's stake program, or the effects of a manager authority compromise. Due to safeguards implemented by Marinade, the latter would in all likelihood only result in a partial loss of rewards for an epoch. We further discuss all authorities in the section on the contract's authority structure.

3 | Scope

The contract audit's scope comprised of three major components:

- Primarily, the **Implementation** security of the contract's source code
- Additionally, security of the **overall design**
- Additionally, resilience against **economical attacks**

Neodyme considers the source code, located at https://github.com/marinade-finance/liquid-staking-program/tree/main/programs/marinade-finance, in scope for this audit. Third-party dependencies are not in scope. Marinade only relies on the Anchor library, the spl-token program and bincode, all of which are well-established. During the audit, minor changes and fixes were made by Marinade, which the auditors also reviewed in-depth.

Relevant source code revisions are:

- `4b5a6c60016ddefd1126755253f5269b557221bd` • Start of the audit
- `1bd5133d3198c0af05a0952d1ca8cd0d1e19fad6` • Last reviewed revision

Note that as part of the audit, we also decided to re-review parts of Solana's Stake Program, as well as do a cursory review of Marinade's off-chain cranker bot. One of the findings is related to this cranker bot. However, we do not consider them in-scope for this audit.

4 | Project Overview

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

Functionality

Marinade provides a liquid staking solution on Solana, where deposited SOL is staked to a curated list of validators. Users can deposit SOL tokens or existing stake accounts and receive mSOL tokens in return, representing their share of the staking pool. Tokens from the pool are then staked to a manager-controlled list of validators according to a score-based distribution, such that they earn staking rewards. These rewards flow back into the staking pool, contributing to the price appreciation of mSOL.

The user can decide to swap their mSOL back to SOL at any time. Apart from swapping on the open market, they have three options:

- Immediately swap the tokens via the in-built SOL-mSOL liquidity pool of Marinade, hence immediately obtaining liquid SOL.
- Burn their mSOL for a delayed-unstaking ticket, which they can claim against liquid SOL after enough time has passed to allow for the corresponding amount of stake to be deactivated and cool down.
- Burn their mSOL in return for an active staking account being transferred to them.

All methods of withdrawing have fees attached. For the latter two options, this is needed to prevent economic attacks, which we discuss later on in this report.

To enable liquid unstaking and maintain as much liquid SOL in the in-built SOL-mSOL liquidity pool as possible, Marinade relies on two mechanisms:

- When depositing SOL, users do not necessarily receive newly-minted mSOL. Instead, the contract first tries to do a feeless trade using the liquidity pool, and only when the SOL side of the pool is depleted does it start minting new mSOL. The swap price is determined by the contract's internal mSOL-SOL price, which slowly rises as staking rewards accumulate.
- Marinade also provides an LP mechanism. Liquidity providers can deposit SOL into the pool in exchange for LP tokens, which they can later burn for a proportional share of both sides of the pool. Liquid providers don't profit from the feeless SOL-to-mSOL-trades mentioned above, but they receive 50% of all liquid unstaking fees (with the current on-chain configuration). Liquidity providers do not face the impermanent loss issues encountered in traditional liquidity pools.

Marinade also maintains an off-chain cranker bot, which keeps the total Marinade stake in line with user deposits and withdrawals, and keeps the total stake of each validator proportional to its score. The appropriate staking operations are performed in a so-called stake-delta window toward the end of an epoch.

On-Chain Data and Accounts

The on-chain liquid staking contract needs to keep track of a potentially large amount of data, including:

- configuration parameters (e.g., parameters that constrain certain actions or authority addresses),
- up-to-date accounting values summarizing the contract's funds,
- information on all validators in the curated set,
- information on all stake accounts managed by the Marinade instance,
- the state of the liquidity pool, and
- all pending delayed-unstake tickets.

This data is represented on-chain as follows.

An instance of the Marinade protocol has a State account, which stores almost all of this information except for:

- A list of stake accounts managed by the Marinade instance, as well as some metadata attached to them. This list is stored in a separate account whose address is referenced in the main state account.
- The curated list of validators and their scores, as well as other metadata such as their total active Marinade stake. This is also stored in a separate list account whose address is again referenced in the main state account.
- A duplication flag PDA for each validator in the validator list, which prevents re-adding a validator to the list.
- Information on each of the delayed-unstake tickets that are currently open. Each ticket is tracked in a separate contract-owned account and is tied to the instance's state address.

Any PDAs of the Marinade instance are bound to their respective state account's address via derivation seeds. Of those, the following are maintained as authority PDAs:

mSOL Mint Authority spl-token mint authority of mSOL
LP Mint Authority spl-token mint authority of LP tokens
Stake Deposit Authority set as the delegate authority on all Marinade-controlled stake accounts
Stake Withdraw Authority set as the withdraw authority on all Marinade-controlled stake accounts

mSOL leg Authority spl-token owner of the mSOL token account containing the liquidity pool's mSOL

Finally, the contract's funds are stored in the following accounts:

- Marinade-controlled stake accounts, containing all activating / active / deactivating / deactivated but not-yet-withdrawn stake in SOL
- the reserve PDA, containing all not-yet-staked or ready-for-withdrawal SOL
- the SOL leg PDA, containing the liquidity pool's SOL side
- the mSOL leg token account, containing the liquidity pool's mSOL side

All contract fees go either into the reserve PDA and contribute to mSOL price appreciation, or are sent/minted to the mSOL treasury, which is an external mSOL token account. Furthermore, the cranker bot maintains an external operational SOL account, which is used for rent in new stake accounts and duplication flag accounts. This rent is returned to the bot upon closing of those accounts.

Instructions

The contract has a total of 26 instructions, which we briefly summarize here.

Instruction	Category	Summary
Initialize	Permissionless	Initializes a new Marinade instance with a new state account, validator list account and stake list account using the parameters given to it
ChangeAuthority	Admin-Only	If specified, changes any contract-stored authority addresses, as well as the address of the treasury and the operational SOL account
ConfigLP	Admin-Only	If specified, changes any of the liquidity pool's fee parameters
ConfigMarinade	Admin-Only	If specified, changes the Marinade instance's fee parameters, stake-delta window length, feature flags, and some constraints on stake account size, total funds managed or user position delta per instruction
ConfigValidatorSystem	Admin-Only	Changes the number of additional stake-delta runs allowed
Pause / Unpause	PauseAuthority-Only	Disables or Reenables all instructions which aren't admin-only instructions or Unpause

Instruction	Category	Summary
AddValidator	Manager-Only	Adds a new validator record to the validator list and creates the accompanying duplication flag account
RemoveValidator	Manager-Only	Removes a validator record from the validator list and closes the accompanying duplication flag account
SetValidatorScore	Manager-Only	Changes the score of a validator
EmergencyUnstake	Manager-Only	Unstakes a stake account from a validator whose score has been set to 0 and sets its status as emergency cooling down
PartialUnstake	Manager-Only	Unstakes (part of) a stake account from an overstaked validator and sets its status as emergency cooling down
AddLiquidity	Liquidity Provider	Liquidity provider (permissionless) deposits SOL into the liquidity pool's SOL side and receives a proportional share of newly minted LP tokens
RemoveLiquidity	Liquidity Provider	Liquidity provider burns their LP tokens in exchange for a proportional share of both the SOL and mSOL sides of the liquidity pool
LiquidUnstake	User	User swaps mSOL for liquid SOL via the liquidity pool; fees are levied in mSOL and are split among the pool's mSOL side and the treasury
Deposit	User	User deposits SOL and receives a proportional amount of mSOL; this is first done via a feeless liquidity pool swap, and only when the liquidity pool is out of mSOL are new mSOL minted
DepositStakeAccount	User	User deposits active stake account and receives a proportional amount of newly minted mSOL; stake account must be staked to curated validator, unless the validator system currently allows auto-adding of validators, in which case the corresponding validator record is added to the validator list and the accompanying duplication flag created

Instruction	Category	Summary
OrderUnstake	User	User burns mSOL in exchange for a delayed-unstake ticket which they may exchange for liquid SOL after some time; amount of SOL they receive is calculated from the contract's mSOL-to-SOL rate at time of ticket creation; fee is applied to the amount of liquid SOL the user receives from the reserve and contributes to mSOL price appreciation
Claim	User	User hands in their delayed-unstake ticket after it has matured, and receives the corresponding amount of SOL
WithdrawStakeAccount	User	User burns mSOL in exchange for control of a stake account which split from one of the Marinade-controlled stake accounts; fees are levied in mSOL and sent to the treasury
UpdateActive	Crank	Updates an active stake account by withdrawing any superfluous lamports from it (which may for example come from MEV revenue sharing validators) and calculating stake rewards since last update; all funds received in either way are subject to a fee, which is levied by minting mSOL to the treasury
UpdateDeactivated	Crank	Updates a fully deactivated stake account by withdrawing all lamports to the reserve; rent is forwarded to the operational SOL account; for any rewards or other funds that have come in since the last update, apply a fee to them by minting mSOL to the treasury
StakeReserve	Crank	Stakes some of the reserve to a validator who, according to the current score distribution, is understaked; new stake account is added to the stake list
DeactivateStake	Crank	Deactivates all or part of a stake account from an overstaked validator, splitting the stake account if needed

Instruction	Category	Summary
Redelegate	Crank	Redelegate all or part of a stake account from an overstaked validator to an understaked validator; if needed, splits the old stake account and adds the new one to the stake list
MergeStakes	Crank	Merges two stake accounts that are delegated to the same validator; old stake account is removed from the stake list

Note that while cranker instructions are permissionless, there is no incentive reward for executing them.

Authority Structure and Off-Chain Components

In Marinade, there are four privileged authorities: The upgrade authority, the admin authority, the pause authority, and the manager authority. Marinade also maintains an off-chain cranking infrastructure, which periodically calls the permissionless cranking instructions of the contract to maintain its operation. Each of these entities has different powers, and the potential effects of each of them being abused are very different.

It is apparent that Marinade has put considerable thought into the security of these authorities. The two authorities that can cause the most damage – the upgrade and admin authority – are operated via multisigs.

In this section, we discuss what powers each of the authorities has, and what the worst-case effect of a compromise of each of them would be.

Upgrade Authority

As with any contract, the upgrade authority has complete control over the program and the funds it controls. It is hence one of the most critical components of the security of the protocol. By maliciously upgrading the contract, the upgrade authority can irreversibly transfer control of all funds in the reserve, in the liquidity pool, and in all stake accounts to itself or other parties.

Marinade is aware of this and has put considerable effort into making the upgrade authority safe. They have established a robust 6-out-of-13 multisig as its upgrade authority. The on-chain multisig is managed by a non-upgradable deployment of Coral Multisig (formerly Serum Multisig) and can be

viewed here. Marinade states that the addresses belonging to this multisig are owned by: Marinade team members (3 votes), Jupiter, Mango, Miton C, Orca, Phantom, Raydium, Solend, Solflare, Staking Facilities and Triton.One. We independently verified this with Jupiter, Mango, Orca, Raydium, Solend, Solflare, Staking Facilities and Triton.One.

Since the multisig members are well-established ecosystem members with good reputation, the likelihood of a collusion between them can be seen as extremely low.

Admin Authority

Marinade's admin authority is set to a governance of the Marinade DAO, specifically the "Marinade Liquid Staking Admin Authority Wallet" here. This can be verified in the main state account data here. The realms instance has a 4-out-of-7 council consisting of Marinade team members. Under normal conditions, this council does all admin operations.

The admin authority has the power to set many contract parameters, including fee percentages, as well as change the address to which fees and account rent funds are sent. They can also change the addresses of lower authorities, i.e., the manager and the pause authority.

There are multiple attacks which a compromised admin authority would be able to do:

Cause a bank run and profit from high withdrawal fees. By setting the treasury address to themselves, they will receive all fees from stake account withdrawals, as well as part of the fees for liquidity pool swaps. They can also set these fees up to the contract's hardcoded maximums, resulting in them getting 0.2% of all stake account withdrawals and roughly 7.5% of all liquidity pool swaps (by setting the LP's liquidity target to `u64::MAX`, the LP's maximum fee to 10% and the treasury cut to 75%). They can then cause a bank run on Marinade, e.g., by publicizing that the admin authority has been compromised. Presumably, a non-negligible percentage of users will exit their funds via the liquidity pool or via withdrawing stake accounts before the upgrade authority multisig can react, meaning the attacker will receive considerable profit from fees.

Steal an epoch or more of staking rewards. By changing the manager authority to themselves, they can add their own validator, reconfigure score parameters and shift most or all Marinade-control stake to their validator. That validator can have 100% staking fees. By changing the pause authority and pausing the contract, they can prevent users from withdrawing that stake. Note that they themselves are not subject to pausing, since they can do any of their non-admin operations in a transaction that is sandwiched by an unpause and a pause operation. This can only be resolved by the upgrade authority. By timing the attack in the last minutes of an epoch, the attacker is guaranteed to steal at least one epoch of staking rewards.

Grief the chain using a high-staked validator Using the attack pattern above, they now also control a validator with a large amount of stake. Since they will presumably only have this stake for

one epoch and will lose it afterward, no matter what, they have no reason not to misbehave. However, as long as the fraction of stake Marinade controls is not much more than it currently does, this only has a very limited effect. For example, they would get leader slots, in which they could refuse to process transactions. In the very unlikely event that Marinade stake pushes them over 1/3 of the global stake, they could completely DOS the chain.

Grief other stakers by hitting global activation and deactivation limits In the event that the addition of the Marinade stake makes an attacker control more than the global stake activation limit, they could annoy other stakers by redelegating all of it. As a result, all stake accounts that are (de)activating in that epoch would only be partially (de)activated. However, this is – at best – only a nuisance.

Note that all fee parameters the admin controls are subject to strict hardcoded maxima. One of the main revenue streams of the protocol, the fee on staking rewards, is capped at 10% (with the on-chain config currently set to 6%).

Pause Authority

The pause authority has exactly one ability, which is to pause and unpause the contract. It can effectively DOS the contract and lock user funds into the contract, but only until the admin authority replaces it.

The pause authority was recently introduced and is not yet deployed on-chain. Marinade states that the authority will be a dedicated Security Council of people who can review exploit disclosures and react accordingly. They stated that its members will be appointed by MNDE holder votes.

Manager Authority

The manager authority has the power to add and remove validators, as well as to change the score of all validators. However, Marinade implemented a limit on the maximum amount of stake that a manager can move in one epoch (see PR #70). Even if a manager turns malicious, it could only steal a portion of an epoch's staking rewards via malicious stake redistribution. The same applies to the griefing and activation limit-attacks described for the admin authority. Both also have a more limited impact due to the stake movement limit. Note that the manager cannot prevent users from withdrawing by pausing the contract.

Currently, the manager authority seems to be a hot wallet.

Cranker Bot

Marinade maintains an off-chain bot that periodically calls the contract's crank instructions in the relevant time windows. These crank instructions are permissionless, but like with many decentralized applications, the team-maintained cranker is currently the only active cranker.

If the bot were compromised, the worst effect that should result is that staking and unstaking of newly deposited or newly ordered-to-unstake funds is delayed by an epoch. Hence staking rewards or withdraw liquidity may be affected in a very limited way.

5 | Findings

This section outlines all of our findings. They are classified into one of five severity levels, detailed in Appendix C. In addition to these findings, Neodyme delivered the Marinade team a list of nit-picks and additional notes which are not part of this report.

All findings are listed in Table 2 and further described in the following sections.

Table 2: Findings

Identifier	Name	Severity	State
ND-MAR01-LO-01	Depositor May Gain 1 Extra mSOL-Lamport in Some Cases	Low	Resolved
ND-MAR01-LO-02	Auto-Adding of Validators in Stake Account Deposit is DOSable	Low	Resolved
ND-MAR01-LO-03	Malicious Stake Account Depositing May Exhaust Stake List or Validator List	Low	Resolved
ND-MAR01-LO-04	Duplication Flag Can Be Revived When Removing Validator	Low	Resolved
ND-MAR01-LO-05	Possibility to DOS Cranker's StakeReserve Calls via Frontrunning	Low	Resolved
ND-MAR01-IN-01	Attacker Can Use Up Extra Stake-Delta Runs Immediately	Info	Acknowl.
ND-MAR01-IN-02	No Anchor-Init for Ticket Account	Info	Acknowl.
ND-MAR01-IN-03	Possibility to Initialize with Invalid Parameters	Info	Resolved
ND-MAR01-IN-04	Withdraw Fees Round Down	Info	Acknowl.
ND-MAR01-IN-05	Stake System Can Contain Unmergeable Transient Stake Accounts	Info	Acknowl.
ND-MAR01-IN-06	Possibility to Create Stake Accounts With Less Than `min_stake` Stake	Info	Resolved
ND-MAR01-IN-07	Data Accounts and State Aren't Checked to be Distinct at Initialization	Info	Resolved
ND-MAR01-IN-08	Data Accounts Size Unchecked at Initialization	Info	Resolved
ND-MAR01-IN-09	Multiple State Accounts	Info	Acknowl.

[ND-MAR01-LO-01] Depositor May Gain 1 Extra mSOL-Lamport in Some Cases

Severity	Impact	Affected Component	Status
Low	Loss of funds in extremely unlikely cases	Deposit instruction	Resolved

In the deposit instruction, the contract in one specific case implicitly rounds up in the calculation of how much mSOL the user receives in one of the cases.

Denote:

- L = the amount of input lamports
- B = the mSOL-lamport balance in the liquidity pool's mSOL leg
- S = the total mSOL supply
- T = the total virtual staked lamport balance

If we used real-numbered values, the "correct" behavior would be: The depositor pays L SOL-lamports and gets $x_0 = L \cdot S/T$ mSOL-lamports. This is exactly the contract's current SOL-to-mSOL rate, multiplied by the amount of input lamports. However, the contract uses integers, which round down during division.

In the deposit instruction, the case where you only swap or only mint is fine since, in both cases, you pay L SOL-lamports and get $\lfloor (LS)/T \rfloor$ mSOL-lamports which, due to the round-down operation, is slightly worse than x_0. However, the case where you both swap and mint – because you get more mSOL than there is in the liquidity pool mSOL leg – implicitly rounds up, meaning you can get a slightly better rate than you should.

Constraints for the latter case to occur are $\lfloor (LS)/T \rfloor > B$ and $B > 0$ (and obviously $L, S, T \geq 0$). In this case, the depositor gets the entirety of the liquidity pool's mSOL balance. The lamport value of that is then subtracted from the number of lamports the user deposits, and the remaining lamports are converted to mSOL again and freshly minted to the user. Overall, the user pays L SOL-lamports and gets $x_1 = B + \lfloor ((L - \lfloor (B \cdot T)/S \rfloor) \cdot S)/T \rfloor$ mSOL-lamports.

We can rewrite $x_1 = B + \lfloor (L \cdot S - ((B \cdot T) - (B \cdot T) \bmod S))/T \rfloor = \lfloor (L \cdot S + (B \cdot T) \bmod S)/T \rfloor$. It's easy to see that there are ways to achieve $x_1 > x_0$. Take e.g. $T = 1499, S = 1000, L = 5, B = 2$. We get $x_1 = 4$ mSOL-lamports, even though we should have gotten $x_0 = 3.336$.

Note that at best, we can steal $x_1 - x_0 \leq (BT \bmod S)/T \leq S/T$ mSOL-lamports. During normal operation we have $S \leq T$, so in that case it is limited to stealing at most 1 mSOL-lamport. With

transaction fees, mSOL would have to be worth a factor of several thousand over SOL for this to be profitable. Furthermore, note that a high `min_deposit` may disincentivize this attack.

Note that if $S \leq T$ is violated, e.g. due to a slashing event, more than one mSOL-lamport can potentially be stolen.

We recommended not allowing this implicit rounding-up.

Relevant code: https://github.com/marinade-finance/liquid-staking-program/blob/4b5a6c60016ddefd1126755253f5269b557221bd/programs/marinade-finance/src/instructions/user/deposit.rs#L190

Resolution

Marinade first implemented a fix by adding one to the amount of lamports that are swapped. We verified the fix: this change means the user now gets $x_1 = B + ((L - ((B \cdot T)//S + 1)) \cdot S)//T = (L \cdot S + (B \cdot T) \bmod S - S)//T$, which is always guaranteed to be less than or equal to $x_0 = (L \cdot S)/T$.

This fix was proposed for merging in Marinade PR #67.

Another fix was later merged from PR #69. It fixes the issue by only computing the mSOL buy amount once and subtracting the swap amount from it. This prevents the implicit rounding-up behavior seen before.

[ND-MAR01-LO-02] Auto-Adding of Validators in Stake Account Deposit is DOSable

Severity	Impact	Affected Component	Status
Low	Minor DOS	DepositStakeAccount instruction	Resolved

In DepositStakeAccount, the program creates the duplication flag using `system_instruction::create_account`. This fails if the account already has lamports, meaning an attacker can prevent it by sending lamports to this account beforehand.

Relevant code: https://github.com/marinade-finance/liquid-staking-program/blob/4b5a6c60016ddefd1126755253f5269b557221bd/programs/marinade-finance/src/instructions/user/deposit_stake_account.rs#L185

Resolution

Marinade resolved this issue by deprecating the auto-add-validator feature in PR #68.

[ND-MAR01-LO-03] Malicious Stake Account Depositing May Exhaust Stake List or Validator Lists

Severity	Impact	Affected Component	Status
Low	Partial DOS	Stake List and Validator List	Resolved

An attacker can reach the stake list account's size limit by depositing many stake accounts. This is recoverable via merging stake accounts, though depending on what the frequency of the cranker bot's merge operation is, it may, for a while, block reserve staking or other users wanting to deposit.

Main state account has around 18 thousand slots for stake accounts, so this would take nontrivial amounts of funds.

Note that stake accounts are not mergeable if they are in a transient state; see issue `[ND-MAR01-IN-05]`.

This attack also works for the validator list if the auto-adding for new validators is allowed in the DepositStakeAccount ix.

Resolution

Marinade implemented resizing of stake and validator lists in PR #68. In addition, auto-adding for new validators has been removed due to `[ND-MAR01-LO-02]`. That resolves this issue as well.

[ND-MAR01-LO-04] Duplication Flag Can Be Revived When Removing Validator

Severity	Impact	Affected Component	Status
Low	Manager abuse potential	RemoveValidator instruction	Resolved

In the RemoveValidator instruction, the validator's duplication flag is closed by setting its lamports to zero. Accounts are only deleted if lamports are zero at the end of a transaction. This means that the duplication flag can be revived in the same transaction by sending lamports to it.

Since this is a manager-only instruction, this means that the manager would have to be compromised / misbehave. However, if this happens, they can remove a validator and, via this bug, prevent it from ever being re-added (or properly removed) without upgrading the contract.

Relevant code: https://github.com/marinade-finance/liquid-staking-program/blob/4b5a6c60016ddefd1126755253f5269b557221bd/programs/marinade-finance/src/instructions/management/remove_validator.rs#L75

Resolution

The duplication flag is now assigned to the system program after its lamports are removed. This mimics Anchor's account-closing behavior, and is the correct fix here.

The relevant change is here.

[ND-MAR01-LO-05] Possibility to DOS Cranker's StakeReserve Calls via Frontrunning

Severity	Impact	Affected Component	Status
Low	Cranker DOS	Cranker Bot	Resolved

This is an issue in the Marinade-hosted cranker to whose private code we were given access. Please note that we only did a cursory review of the bot's code, and we do not consider it to be officially in-scope.

In the bot's two calls to StakeReserve, it first calls create_account on an account from a random keygen. The call pattern is `tx_execute(create_account_tx).and_then(marinade_call)`. Hence, if the create account tx fails, the Marinade call will not happen.

A malicious attacker can frontrun the `create_account` transaction and send lamports to the account to make it fail. They hence DOS the cranker call to Marinade.

Note that it wouldn't be trivial to frontrun all cranker calls during an epoch, so the attack might be annoying to execute in practice. It is easy if the attacker is either the leader in that slot, or if the leader provides a mempool (e.g. via jito).

Resolution

The account is Anchor-`init`ed in the StakeReserve instruction, meaning the cranker no longer needs to initialize the account separately. Anchor's `init` handles the case of a pre-funded account correctly.

The relevant change is here.

[ND-MAR01-IN-01] Attacker Can Use Up Extra Stake-Delta Runs Immediately

Severity	Impact	Affected Component	Status
Info	DOS of additional stake-delta runs	StakeReserve instruction	Acknowledged

Marinade has a concept of additional stake-delta runs, which are stake-delta operations after the first stake-delta operation in an epoch. They are used to capture any last-minute deposits. Their number is limited by a parameter stored in the state account. However, an attacker can repeatedly deposit small amounts and call StakeReserve to use up extra stake runs.

Relevant code: https://github.com/marinade-finance/liquid-staking-program/blob/4b5a6c60016ddefd1126755253f5269b557221bd/programs/marinade-finance/src/instructions/crank/stake_reserve.rs#L165

Resolution

Marinade stated that extra stake-delta runs can be incremented by the bot-manager and that this does not affect the protocol enough to justify the attack. We agree.

[ND-MAR01-IN-02] No Anchor-Init for Ticket Account

Severity	Impact	Affected Component	Status
Info	None	OrderUnstake instruction	Acknowledged

The OrderUnstake instruction creates the ticket account in a user-provided, pre-funded and program-owned account. It would be nicer to use Anchor init for this.

If a bug is introduced that, for example, introduces the possiblity to leave program-owned accounts with rent on them, this could be used to steal that rent, since the rent of the ticket account is given to the user in a successful Claim instruction.

Resolution

Marinade acknowledged this but stated that they do not want to change the public interface.

[ND-MAR01-IN-03] Possibility to Initialize with Invalid Parameters

Severity	Impact	Affected Component	Status
Info	Creation of invalid State instance	Initialize instruction	Resolved

There are constraints which are enforced when changing parameters via ConfigMarinade. Some of those are not enforced in the Initialize instruction.

Specifically, this is the case for the following parameters:

- `min_stake`
- `slots_for_stake_delta`

This could lead to a state that users might not expect.

Resolution

This finding does not affect the main state instance, as its parameters are inside the allowed boundaries. It only affects newly created instances. Still, a fix was implemented in Pull #71, now checking the parameters of newly created instances correctly.

[ND-MAR01-IN-04] Withdraw Fees Round Down

Severity	Impact	Affected Component	Status
Info	Infeasible fee circumvention	WithdrawStakeAccount and DelayedUnstake instructions	Acknowledged

In WithdrawStakeAccount and DelayedUnstake, the fee levied by the contract rounds down. If min_withdraw is low, users can avoid the withdraw fees by withdrawing a small enough amount that fees round down to zero (or, for higher amounts, users can pay somewhat smaller fees than intended).

Transaction fees make this infeasible. For WithdrawStakeAccount, this should also be prevented by Solana's minimum stake requirement. It would still be "safer" to always have fees round up.

Resolution

Marinade chose to set the instance parameters to `min_deposit` to 0.001 and `min_withdraw` to 0.001, which avoids zero-fee withdraws without code changes.

[ND-MAR01-IN-05] Stake System Can Contain Unmergeable Transient Stake Accounts

Severity	Impact	Affected Component	Status
Info	Temporary DOS assuming certain conditions	Stake account list	Acknowledged

In rare cases when the global stake activation or deactivation limit is hit, it can happen that Marinade has transient stake accounts in its stake system, i.e. stake accounts where `delegation.stake` does not match the effective stake. Such accounts can also be deposited via the DepositStakeAccount instruction.

The stake program will reject merging these accounts, so Marinade can be stuck with many stake accounts per validator during one or more epochs which the cranker is unable to merge until they are fully activated. An attacker could theoretically use this situation to fill up the stake account list by depositing many transient stake accounts. It might also confuse the cranker.

We mention that these are very specific circumstances, and that the attack has low impact.

Resolution

Marinade acknowledged this but decided that due to the considerations in the last sentence above, the additional complexity of a fix was not proportionate. They also stated that during the fixing of `[ND-MAR01-IN-08]`, new instructions for expanding and contracting the stake account list capacity were added. Should the list be full, it can be expanded up to the maximum of 10MiB.

[ND-MAR01-IN-06] Possibility to Create Stake Accounts With Less Than `min_stake` Stake

Severity	Impact	Affected Component	Status
Info	Limited Invariant Violation	StakeReserve instruction	Resolved

The calculation for the size of a new stake account in the StakeReserve instruction can result in a stake smaller than the configured `min_stake` if `total_stake_delta < min_stake`.

Say `min_stake` is 10 SOL and there's only one validator (for simplicity). Then an attacker can e.g. repeatedly Deposit 11 SOL and DelayedWithdraw 10 SOL, for a total stake delta of 1 SOL. Then they call StakeReserve every time. In total, they create one stake account with only 1 SOL per stake delta run. This violates the invariant that no Marinade stake account has less than `min_stake` stake.

Relevant code: https://github.com/marinade-finance/liquid-staking-program/blob/4b5a6c60016ddefd1126755253f5269b557221bd/programs/marinade-finance/src/instructions/crank/stake_reserve.rs#L200

Resolution

Marinade fixed this in this pull request.

[ND-MAR01-IN-07] Data Accounts and State Aren't Checked to be Distinct at Initialization

Severity	Impact	Affected Component	Status
Info	Creation of unusable State instance	Initialize instruction	Resolved

In the initialize instruction, the stake list, validator list and state account aren't checked to be pairwise distinct.

If the initializer sets, for example, validator list and stake list to be equal, this results in an unusable state since that account will first be initialized as stake list, then overwritten by validator list. Hence that instance wouldn't have a stake list at all. Setting them all equal means both list's data accounts get overwritten and you don't have either.

Resolution

Marinade implemented a check here that all three accounts are pairwise distinct. Neodyme verified the fix.

[ND-MAR01-IN-08] Data Accounts Size Unchecked at Initialization

Severity	Impact	Affected Component	Status
Info	Creation of limited-usability State instance	Initialize instruction	Resolved

In the initialize instruction, the size of the stake list account and the validator list account is unchecked (besides checking that it fits the discriminator). So you could have a stake list account that could not hold any stake records, or only a very limited number of stake records. Same for the validator list.

Resolution

Marinade added two new instructions for resizing the stake list (relevant commit) and validator list account (relevant commit).

[ND-MAR01-IN-09] Multiple State Accounts

Severity	Impact	Affected Component	Status
Info	Possible user confusion	State Account	Acknowledged

The state account does not have a hardcoded seed, hence multiple state accounts can exist. Anyone can initialize a new state account. In its current state, no confusion between different state accounts can occur in the contract itself, and the Marinade team states that they have checked that no confusion can occur in the front end or the cranker bots.

An attacker could theoretically still create a new instance with a new state account, add their own validators as the only options, and accrue user stakes by tricking them into depositing – which some might be tricked into since it is the official Marinade program at the official address. They can then pause the program to prevent destaking or withdrawal, and set their validator staking fees to 100%.

Resolution

Marinade acknowledged this but did not change the behavior. The relevant discussion points were that the set of users that would be fooled by such an “attack” is very small, and more than one state account is useful for testing and other purposes.

6 | Discussion of Economic Attack Vectors

In this section, we discuss several economic attack patterns which arise from the functionality of the program. None of them are currently of major concern.

Validator Commission Attack

One of the most obvious attack vectors for almost all SOL staking protocols on Solana is validators' total control over their own staking commission. Validators can set their staking fee arbitrarily high just before the end of an epoch, and it is still applied to the staking rewards of any funds delegated to them for that epoch. Hence, if they set their fees to 100%, they will obtain all rewards of funds staked to them.

Typical commissions for validators are between 1% and 10%. Hence with this attack, they would obtain 10-100 epochs worth of rewards in one epoch. Marinade states that they check for commission rugs and blacklist offending validators. This attack should, therefore, be limited to one epoch.

Attacks Related to Rewards

We discuss two attacks that users could run to try to profit from rewards in a way that abuses the contract's functionality and why they're currently prevented and not of concern, respectively.

User Deposits SOL and Immediately Withdraws Stake Account

If the stake account withdrawal fee is lower than one epoch of staking rewards, it becomes profitable to deposit SOL and immediately withdraw a stake account. Hence, let's look at what a sensible choice would be.

Let r = `ms_per_year` / (`ms_per_slot` * `slots_per_epoch`) where `slots_per_epoch` = 432,000 and `ms_per_year` ~ 31,536,000,000. Then the factor applied to a stake's value to calculate its value after one epoch of rewards is $\sqrt[r]{1 + \mathrm{APY}}$, where APY denotes the current rewards APY.

Generously assuming maximum `ms_per_slot` at 800ms, and generously assuming maximum staking rewards plus MEV rewards at 20% (note that apy also depends on slot times, so this is even more generous than the delta to the current best apy), we have $r = 91.25$ and a one-epoch rewards factor of 1.001978. Hence the rewards of one epoch are just under 0.2% in this extremely generous setting.

This suggests that the current on-chain setting of 0.2% stake account withdrawal fees is a sensible choice.

This should obviously be monitored and adjusted in case of significant changes in slot times or staking/MEV APY.

User Deposits to Capitalize on Large Reward Delta Between Epochs

If a user knows that rewards will drop next epoch, or the reward is very high only in the current epoch for whatever reason, and the user has undelegated SOL, they can capitalize on that using Marinade: They call Deposit to deposit their SOL and obtain mSOL in return. Via mSOL appreciation, they immediately profit from that epoch's rewards. Without Marinade, they would have needed to wait for their stake to activate and could only have gotten rewards from the next epoch onward.

However, this is disincentivized by the fact that all withdrawal methods have a fee attached. The reward delta between epoch would have to be very large for this to become even remotely feasible.

Attacks Related to Slashing

Slashing has not been implemented on Solana and likely won't be in the near future. However, when or if it is, the following patterns may be of concern.

Attacker Shorts mSOL and Purposefully Causes Slashing

A validator can purposefully misbehave to cause slashing while shorting mSOL.

Say, e.g., they control validators that have a combined ~10% of all stake staked via Marinade. They let all their validators severely misbehave, causing 20% of all their stake to be slashed. mSOL is now worth approximately 2% less. Via leveraged shorting of mSOL, the attacker may capitalize on this quite effectively. The potential immediate upside from this may outweigh the loss of the staking fees they would have received over time.

This is an attack vector that any liquid staking protocol whose token is traded on the open market would have once/if slashing is implemented. In Marinade's case, it would be best to only have strongly trusted validators in the curated set, ideally ones that also have significant non-Marinade stake, so that their staking reward loss is larger in relation to the potential gain from the mSOL price delta.

Arbitraging Imminent Slashing

A slashing event on a validator staked by Marinade wouldn't be detected until the next call to the Update instruction. If the slashing has not happened yet, or the update has not picked up on it, Withdraw instructions use the old mSOL price.

This will result in an arbitrage opportunity, or in extreme cases perhaps a small bank run when a slashing occurs, or once it becomes apparent a slashing may occur soon. Users that get their tickets (or withdraw a stake account) before the update happens still get the old rate, leaving Marinade with "bad debt," which will result in even worse rates for remaining users.

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