sec3™

Security Assessment Report

Marinade Finance Liquid Staking

October 25, 2023

Summary

The sec3 team (formerly Soteria) was engaged to do a thorough security analysis of the Marinade Finance Liquid Staking Program at https://github.com/marinade-finance/liquid-staking-program. The initial audit was done on the source code of the following version

- **Contract "marinade-finance":**
  - commit `4b5a6c60016ddefd1126755253f5269b557221bd`

The review revealed 13 issues. The team responded with a second version for the post-audit review to see if the reported issues were resolved. The audit was concluded on commit `1bd5133d3198c0af05a0952d1ca8cd0d1e19fad6`, which is the version with all fixes applied to be deployed.

Table of Contents

Result Overview

MARINADE FINANCE LIQUID STAKING PROGRAM

Issue	Impact	Status
[L-1] split_stake_account rent not returned in some cases	Low	Resolved
[L-2] min_stake and slots_for_stake_delta validation in initialization	Low	Resolved
[L-3] Add validator without approval	Low	Resolved
[L-4] duplication_flag account is not properly closed	Low	Resolved
[L-5] Small rounding error in deposit	Low	Resolved
[I-1] Conflicts between code and comments in deposit_stake_account	Info	Resolved
[I-2] Redundant validator update in stake_reserve	Info	Resolved
[I-3] Inconsistent comments/implementations in remove_liquidity	Info	Resolved
[I-4] MIN_UPDATE_WINDOW may be shorter than expected	Info	Resolved
[I-5] Execution not terminated when lp_mint.supply > liq_pool.lp_supply	Info	Resolved
[I-6] The global extra_stake_delta_runs can be maliciously consumed	Info	Resolved
[I-7] stake_target may be less than min_stake in stake_reserve	Info	Resolved
[I-8] delayed_unstake_fee and withdraw_stake_account_fee checks	Info	Resolved

Findings in Detail

[L -1] split_stake_account rent not returned in some cases

In some crank instructions, a newly initialized stake account, `split_stake_account`, is used during the operation.

However, in some cases where this account is not used, the rent fee is not returned to the bot or user calling this crank. Malicious users might be able to construct specific scenarios to trigger the bots and consume the bot's balance, potentially enabling a DoS attack.

```
/* programs/marinade-finance/src/instructions/crank/deactivate_stake.rs */
#[account(
    init,
    payer = split_stake_rent_payer,
    space = std::mem::size_of::<StakeState>(),
    owner = stake::program::ID,
)]
pub split_stake_account: Account<'info, StakeAccount>,

/* programs/marinade-finance/src/instructions/crank/deactivate_stake.rs */
// compute how much we should unstake from this validator
let validator_active_balance = validator.active_balance; // record for event
if validator_active_balance <= validator_stake_target {
    msg!(
        "Validator {} has already reached unstake target {}",
        validator.validator_account,
        validator_stake_target
    );
    return Ok(()); // Not an error. Don't fail other instructions in tx
}

/* programs/marinade-finance/src/instructions/crank/deactivate_stake.rs */
// we must perform partial unstake
// Update validator.last_stake_delta_epoch for split-stakes only because
// probably we need to unstake multiple whole stakes for the same validator
if validator.last_stake_delta_epoch == self.clock.epoch {
    // note: we don't consume self.state.extra_stake_delta_runs
    // for unstake operations. Once delta stake is initiated
    // only one unstake per validator is allowed (this maximizes mSOL price increase)
    msg!(
        "Double delta stake command for validator {} in epoch {}",
        validator.validator_account,
        self.clock.epoch
    );
    return Ok(()); // Not an error. Don't fail other instructions in tx
}

/* programs/marinade-finance/src/instructions/management/partial_unstake.rs */
// if validator is already on-target (or the split will be lower than min_stake),
// exit now
if validator.active_balance <= validator_stake_target
                                + self.state.stake_system.min_stake {
    msg!(
        "Current validator {} stake {} is <= target {} +min_stake",
        validator.validator_account,
        validator.active_balance,
        validator_stake_target
    );
    return Ok(()); // Not an error. Don't fail other instructions in tx
}
```

Resolution

The issue was fixed by commit `1bd5133`.

[L -2] min_stake and slots_for_stake_delta validation in initialization

In the initialize instruction, values for min_stake and slots_for_stake_delta are assigned to the state account without undergoing any preliminary validation.

```
/* programs/marinade-finance/src/instructions/admin/initialize.rs */
pub fn process(
    &mut self,
    InitializeData {
        min_stake,
        slots_for_stake_delta,
    }: InitializeData,
    reserve_pda_bump: u8,
) -> Result<()> {
    self.state.set_inner(State {
        stake_system: StakeSystem::new(
            self.state_address(),
            *self.stake_list.key,
            &mut self.stake_list.data.as_ref().borrow_mut(),
            slots_for_stake_delta,
            min_stake,
            0,
            additional_stake_record_space,
        )?,
    });
```

In contrast, the config_marinade instruction incorporates a range validation procedure for these two numerical parameters.

```
/* programs/marinade-finance/src/instructions/admin/config_marinade.rs */
let slots_for_stake_delta_change =
    if let Some(slots_for_stake_delta) = slots_for_stake_delta {
        require_gte!(
            slots_for_stake_delta,
            StakeSystem::MIN_UPDATE_WINDOW,
            MarinadeError::UpdateWindowIsTooLow
        );
        let old = self.state.stake_system.slots_for_stake_delta;
        self.state.stake_system.slots_for_stake_delta = slots_for_stake_delta;
        Some(U64ValueChange {
            old,
            new: slots_for_stake_delta,
        })
    } else {
        None
    };

let min_stake_change = if let Some(min_stake) = min_stake {
    require_gte!(
        min_stake,
        5 * self.state.rent_exempt_for_token_acc,
        MarinadeError::MinStakeIsTooLow
    );
    let old = self.state.stake_system.min_stake;
    self.state.stake_system.min_stake = min_stake;
    Some(U64ValueChange {
        old,
        new: min_stake,
    })
} else {
    None
};
```

Resolution

The issue was fixed by commit `00bee19`.

[L -3] Add validator without approval

```
/* programs/marinade-finance/src/instructions/user/deposit_stake_account.rs */
pub struct DepositStakeAccount<'info> {
    pub stake_authority: Signer<'info>,
    #[account(
        mut,
        owner = system_program::ID
    )]
    pub rent_payer: Signer<'info>,
}

/* programs/marinade-finance/src/instructions/user/deposit_stake_account.rs */
pub fn process(&mut self, validator_index: u32) -> Result<()> {
    let validator_active_balance =
        if validator_index == self.state.validator_system.validator_count() {
            if self.state.validator_system.auto_add_validator_enabled == 0 {
                return err!(MarinadeError::AutoAddValidatorIsNotEnabled);
            }
```

In deposit_stake_account instruction, it's possible to add new validators repeatedly. Once the list size is fixed by validator_list.data, malicious users may keep adding validators without securing the approvals before validator_system.manager_authority adding validators, leading to DoS issues.

By contracts, the add_validator instruction requires the signature from the validator_system.manager_authority.

Resolution

The issue was fixed by commits 265e0d7 and 4606d9a.

[L -4] duplication_flag account is not properly closed

The owner of the duplication_flag account should be assigned to the system program.

```
/* programs/marinade-finance/src/instructions/management/remove_validator.rs */
// record for event, then remove all flag-account lamports to remove flag
let operational_sol_balance = self.operational_sol_account.lamports();
let rent_return = self.duplication_flag.lamports();
**self.duplication_flag.try_borrow_mut_lamports()? = 0;
**self.operational_sol_account.try_borrow_mut_lamports()? += rent_return;

emit!(RemoveValidatorEvent {
    state: self.state.key(),
    validator: validator_vote,
    index,
    operational_sol_balance,
});
```

Resolution

The issue was fixed by commit 5e0bc70.

[L -5] Small rounding error in deposit

```
/* programs/marinade-finance/src/instructions/user/deposit.rs */
120 | let user_msol_buy_order = self.state.calc_msol_from_lamports(lamports)?;
121 | msg!("--- user_m_sol_buy_order {}", user_msol_buy_order);
122 |
128 | let msol_leg_balance = self.liq_pool_msol_leg.amount;
129 | let msol_swapped: u64 = user_msol_buy_order.min(msol_leg_balance);
130 | msg!("--- swap_m_sol_max {}", msol_swapped);
131 |
132 | //if we can sell from the LiqPool
133 | let sol_swapped = if msol_swapped > 0 {
134 |     // how much lamports go into the LiqPool?
135 |     let sol_swapped = if user_msol_buy_order == msol_swapped {
136 |         //we are fulfilling 100% the user order
137 |         lamports //100% of the user deposit
138 |     } else {
141 |         self.state.msol_to_sol(msol_swapped)?
142 |     };

/* programs/marinade-finance/src/instructions/user/deposit.rs */
181 | let sol_deposited = lamports - sol_swapped;
182 | // check if we have more lamports from the user
183 | let msol_minted = if sol_deposited > 0 {
184 |     self.state.check_staking_cap(sol_deposited)?;
190 |     let msol_to_mint = self.state.calc_msol_from_lamports(sol_deposited)?;
191 |     msg!("--- msol_to_mint {}", msol_to_mint);
```

sol_swapped at line 141 may be smaller than the actual value.

As a result, msol_to_mint minted to the user may be inaccurate, a proper value should be user_msol_buy_order - msol_swapped to reduce error.

Resolution

The issue was fixed by commit df4e1c4.

[I -1] Conflicts between code and comments in deposit_stake_account

```
/* programs/marinade-finance/src/instructions/user/deposit_stake_account.rs */
116 | // require the stake is active since current_epoch + WAIT_EPOCHS
117 | require_gte!(
118 |     self.clock.epoch,
119 |     delegation.activation_epoch + Self::WAIT_EPOCHS,
120 |     MarinadeError::DepositingNotActivatedStake
121 | );
```

The comment on line 116 and the actual code implementation do not align. It appears the `current_epoch` in the comments should be `activation_epoch`.

Resolution

The issue was fixed by commit `4fb77b5`.

[I -2] Redundant validator update in stake_reserve

```
/* programs/marinade-finance/src/instructions/crank/stake_reserve.rs */
154 | if validator.last_stake_delta_epoch == self.clock.epoch {
155 |     // check if we have some extra stake runs allowed
156 |     if self.state.stake_system.extra_stake_delta_runs == 0 {
157 |         msg!(
158 |             "Double delta stake command for validator {} in epoch {}",
159 |             validator.validator_account,
160 |             self.clock.epoch
161 |         );
162 |         return Ok(()); // Not an error. Don't fail other instructions in tx
163 |     } else {
165 |         self.state.stake_system.extra_stake_delta_runs -= 1;
166 |     }
167 | } else {
168 |     // first stake in this epoch
169 |     validator.last_stake_delta_epoch = self.clock.epoch;
170 | }

/* programs/marinade-finance/src/instructions/crank/stake_reserve.rs */
277 | validator.active_balance += stake_target;
278 | validator.last_stake_delta_epoch = self.clock.epoch;
279 | // Any stake-delta activity must activate stake delta mode
280 | self.state.stake_system.last_stake_delta_epoch = self.clock.epoch;
281 | self.state.validator_system.set(
282 |     &mut self.validator_list.data.as_ref().borrow_mut(),
283 |     validator_index,
284 |     validator,
285 | )?;
```

In the `stake_reserve` instruction, the `validator.last_stake_delta_epoch` is updated at line 278 after the operation. However, the relevant operation has already been performed in the conditional statement from lines 154 to 170.

Resolution

The issue was fixed by commit `6ff86ad`.

[I -3] Inconsistent comments/implementations in remove_liquidity

No error was returned in the true branch.

```
/* programs/marinade-finance/src/instructions/liq_pool/remove_liquidity.rs */
if lp_mint_supply > self.state.liq_pool.lp_supply {
    msg!("Someone minted lp tokens without our permission or bug found");
    // return an error
} else {
    // maybe burn
    self.state.liq_pool.lp_supply = lp_mint_supply;
}
```

Resolution

The issue was fixed by commit `448dd28`.

[I -4] MIN_UPDATE_WINDOW may be shorter than expected

```
/* programs/marinade-finance/src/state/stake_system.rs */
impl StakeSystem {
    pub const STAKE_WITHDRAW_SEED: &'static [u8] = b"withdraw";
    pub const STAKE_DEPOSIT_SEED: &'static [u8] = b"deposit";
    pub const DISCRIMINATOR: &'static [u8; 8] = b"staker__";
    pub const MIN_UPDATE_WINDOW: u64 = 3_000; //min value is 3_000 => half an hour
```

In the current definition, MIN_UPDATE_WINDOW is set to 3000 slots, and it is mentioned in the comments as being equivalent to half an hour.

However, according to data from Solana Explorer, the current Slot time is approximately 420ms, thus the time corresponding to 3000 slots is about 21 minutes, which might be shorter than the anticipated time.

Resolution

The issue was fixed by commit 15d839c.

[I -5] Execution not terminated when lp_mint.supply > liq_pool.lp_supply

- **lp_mint.supply vs. state.liq_pool.lp_supply**

In the remove_liquidity instruction, if lp_mint.supply > liq_pool.lp_supply which means someone minted lp tokens without permission or bug found, only a log message is emitted and the execution is not terminated though stated in the comment.

```
/* programs/marinade-finance/src/instructions/liq_pool/remove_liquidity.rs */
impl<'info> RemoveLiquidity<'info> {
    pub fn process(&mut self, tokens: u64) -> Result<()> {
        // Update virtual lp_supply by real one
        let lp_mint_supply = self.lp_mint.supply;
        if lp_mint_supply > self.state.liq_pool.lp_supply {
            msg!("Someone minted lp tokens without our permission or bug found");
            // return an error
        } else {
            // maybe burn
            self.state.liq_pool.lp_supply = lp_mint_supply;
        }
```

In contrast, the add_liquidity instruction will raise an error in this scenario.

```
/* programs/marinade-finance/src/instructions/liq_pool/add_liquidity.rs */
impl<'info> AddLiquidity<'info> {
    // fn add_liquidity()
    pub fn process(&mut self, lamports: u64) -> Result<()> {
        // if self.state.liq_pool.lp_supply < self.lp_mint.supply, Someone minted
        // lp tokens without our permission or bug found
        require_lte!(
            self.lp_mint.supply,
            self.state.liq_pool.lp_supply,
            MarinadeError::UnregisteredLPMinted
        );
        self.state.liq_pool.lp_supply = self.lp_mint.supply;
```

Consider porting the require_lte! check in add_liquidity to remove_liquidity.

- **msol_mint.supply vs. state.msol_supply**

```
/* programs/marinade-finance/src/instructions/user/deposit.rs */
// impossible to happen check outside bug (msol mint auth is a PDA)
require_lte!(
    self.msol_mint.supply,
    self.state.msol_supply,
    MarinadeError::UnregisteredMsolMinted
);
```

Similarly, deposit() requires msol_mint.supply is less than or equal to state.msol_supply (msol mint accounting maintained by the program) so that no token has been minted outside of the program. However, withdraw() doesn't check this so only deposit() will be blocked in such scenarios.

Resolution

The team responded that this was to avoid blocking withdrawing funds. Since this is a 0% probability situation, no action is needed.

[I -6] The global extra_stake_delta_runs can be maliciously consumed

```rust
/* programs/marinade-finance/src/instructions/crank/stake_reserve.rs */
if validator.last_stake_delta_epoch == self.clock.epoch {
    // check if we have some extra stake runs allowed
    if self.state.stake_system.extra_stake_delta_runs == 0 {
        msg!(
            "Double delta stake command for validator {} in epoch {}",
            validator.validator_account,
            self.clock.epoch
        );
        return Ok(()); // Not an error. Don't fail other instructions in tx
    } else {
        // some extra runs allowed. Use one
        self.state.stake_system.extra_stake_delta_runs -= 1;
    }
} else {
    // first stake in this epoch
    validator.last_stake_delta_epoch = self.clock.epoch;
}
```

In stake_reserve, Marinade ensures that each validator can only execute stake_reserve once per epoch by recording the last_stake_delta_epoch field for each validator. However, this restriction is not absolute. An admin can allow two or even multiple stake-delta operations in a single epoch by setting the extra_stake_delta_runs field in the stake system.

Since stake_reserve itself is a permissionless crank, a malicious user might repeatedly invoke this instruction to consume all of the extra_stake_delta_runs, although this might not bring any benefits to them.

Resolution

The team acknowledged the finding. However, considering that this might not bring any benefits and might not cause significant harm to Marinade, it does not warrant a change.

[I -7] stake_target may be less than min_stake in stake_reserve

```
/* programs/marinade-finance/src/instructions/crank/stake_reserve.rs */
// compute stake_target
// stake_target = target_validator_balance - validator.balance, at least
// self.state.min_stake and at most delta_stake
let stake_target = validator_stake_target
    .saturating_sub(validator_active_balance)
    .max(self.state.stake_system.min_stake)
    .min(total_stake_delta);

// if what's left after this stake is < state.min_stake, take all the remainder
let stake_target = if total_stake_delta - stake_target
                            < self.state.stake_system.min_stake {
    total_stake_delta
} else {
    stake_target
};
```

In the stake_reserve instruction, the calculated stake_target should be ensured to be at least min_stake and at most total_stake_delta. However, the current implementation cannot guarantee this: in the extreme case where total_stake_delta is less than min_stake, the final value of stake_target will be less than min_stake.

Resolution

This issue has been fixed by commit 9695c0d.

[I -8] delayed_unstake_fee and withdraw_stake_account_fee checks

According to the comments, delayed_unstake_fee and withdraw_stake_account_fee should not be zero.

```
/* programs/marinade-finance/src/state/mod.rs */
pub struct State {
    // delayed unstake account fee
    // to avoid economic attacks this value should not be zero
    // (this is required because tickets are ready at the end of the epoch)
    // preferred value is one epoch rewards
    pub delayed_unstake_fee: FeeCents,
    // withdraw stake account fee
    // to avoid economic attacks this value should not be zero
    // (this is required because stake accounts are delivered immediately)
    // preferred value is one epoch rewards
    pub withdraw_stake_account_fee: FeeCents,
    pub withdraw_stake_account_enabled: bool,

/* programs/marinade-finance/src/instructions/delayed_unstake/order_unstake.rs */
// apply delay_unstake_fee to avoid economical attacks
// delay_unstake_fee must be >= one epoch staking rewards
let delay_unstake_fee_lamports = self
    .state
    .delayed_unstake_fee
    .apply(sol_value_of_msol_burned);

/* programs/marinade-finance/src/instructions/user/withdraw_stake_account.rs */
// apply withdraw_stake_account_fee to avoid economical attacks
// withdraw_stake_account_fee must be >= one epoch staking rewards
let withdraw_stake_account_fee_lamports =
    self.state.withdraw_stake_account_fee.apply(sol_value);
```

However, they are initialized to be zero.

```
/* programs/marinade-finance/src/instructions/admin/initialize.rs */
delayed_unstake_fee: FeeCents::from_bp_cents(0),
withdraw_stake_account_fee: FeeCents::from_bp_cents(0),
withdraw_stake_account_enabled: false,
```

Later, when setting them to different values, there are only upper bound checks.

```
/* programs/marinade-finance/src/instructions/admin/config_marinade.rs */
let delayed_unstake_fee_change = if let Some(delayed_unstake_fee)
                                        = delayed_unstake_fee {
    require_lte!(
        delayed_unstake_fee,
        State::MAX_DELAYED_UNSTAKE_FEE,
        MarinadeError::DelayedUnstakeFeeIsTooHigh
    );
    let old = self.state.delayed_unstake_fee;
    self.state.delayed_unstake_fee = delayed_unstake_fee;
    Some(FeeCentsValueChange {
        old,
        new: delayed_unstake_fee,
    })
} else {
    None
};
```

Resolution

The team responded that this is expected. The comment says, "To avoid economic attacks, it should be > 0". But such attacks are allowed by default. No action is needed.

Appendix: Methodology and Scope of Work

The sec3 (formerly Soteria) audit team, which consists of Computer Science professors and industrial researchers with extensive experience in Solana smart contract security, program analysis, testing, and formal verification, performed a comprehensive manual code review, software static analysis, and penetration testing.

Assisted by the sec3 Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
  - Missing ownership checks
  - Missing signer checks
  - Signed invocation of unverified programs
  - Solana account confusions
  - Arithmetic over- or underflows
  - Numerical precision errors
  - Loss of precision in calculation
  - Insufficient SPL-Token account verification
  - Missing rent exemption assertion
  - Casting truncation
  - Did not follow security best practices
  - Outdated dependencies
  - Redundant code
  - Unsafe Rust code
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of the scope of this work

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a sec3 (the "Company") and Marinade Finance (the "Client"). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The Report's sole purpose is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation, or covenant that the Assessed Code: (i) is error and/or bug-free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, sec3 (formerly Soteria) is a leading blockchain security company that currently focuses on Solana programs. We are also building sophisticated security tools incorporating static analysis, penetration testing, and formal verification.

At sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.

sec3™