Security Assessment Report

Marginfi v2 PR238

December 20, 2024

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Marginfi v2 PR238 smart contracts.

The artifact of the audit was the source code of the following programs, excluding tests, in PR#238.

The initial audit focused on the following versions and revealed 9 issues or questions.

program	type	commit
marginfi PR#238	Solana	`883c97f35319068816af2bb94fc1f0911dcdec0e`

This report provides a detailed description of the findings and their respective resolutions.

Table of Contents

Result Overview

Issue	Impact	Status
MARGINFI PR#238		
[M-01] Incorrect insurance withdrawal accounting	Medium	Resolved
[L-01] Liquid insurance fund initialization typo	Low	Resolved
[I-01] The "lending_account_withdraw" may fail in corner cases	Info	Acknowledged
[I-02] Copy-paste error in "lending_pool_deposit_insurance"	Info	Resolved
[I-03] The "deposit_amount" in insurance event reflects pre-fee amount	Info	Resolved
[I-04] Validate "min_withdraw_period"	Info	Resolved
[I-05] Consider restricting token extensions when adding pools	Info	Acknowledged
[I-06] Consider adding interface consistency check	Info	Acknowledged
[I-07] Unused "last_update" in "liquid_insurance_fund"	Info	Resolved

Findings in Detail

MARGINFI PR#238

[M-01] Incorrect insurance withdrawal accounting

The insurance withdrawal logic is flawed because it mixes up the value and number of shares.

In particular, the `withdraw_shares` at line 83, `amount` at lines 199 and 262 are the number of shares to be withdrawn.

However, at line 266, the minimum of `current_amount` and (the total value `total_shares * lazy_share_value`) and `amount` (the shares to be withdrawn) is stored in `delta_decrease`.

When `delta_decrease` equals to `amount`, the `share_decrease`, which is the number of shares to be removed from the `total_shares`, is calculated as `amount / lazy_share_value` at line 268.

The logic only makes sense if the `amount` is the value of the shares to be withdrawn. However, since `amount` represents the number of shares, not their value, the logic incorrectly conflates share quantity with value.

```
/* programs/marginfi/src/instructions/liquid_insurance_fund/withdraw_claim.rs */
064 | pub fn settle_withdraw_claim_in_liquid_insurance_fund<'info>(
065 |     mut ctx: Context<'_, '_, 'info, 'info, SettleWithdrawClaimInLiquidInsuranceFund<'info>>,
066 | ) -> MarginfiResult {
102 |     // 3) Calculate share value in tokens
103 |     let user_withdraw_amount: u64 = liquid_insurance_fund.process_withdrawal(withdrawal)?;

/* programs/marginfi/src/state/liquid_insurance_fund.rs */
045 | impl LiquidInsuranceFund {
072 |     pub fn process_withdrawal(
073 |         &mut self,
074 |         withdrawal: &mut LiquidInsuranceFundWithdrawal,
075 |     ) -> MarginfiResult<u64> {
076 |         // Fetch shares to withdraw
077 |         let withdraw_shares: I80F48 = withdrawal.shares.into();
082 |         // Decrement total share count and update share value
083 |         self.withdraw_shares(withdraw_shares)?;
089 |     }
... |
197 |     pub(crate) fn withdraw_shares(&mut self, amount: I80F48) -> MarginfiResult {
198 |         // Update the internal count of shares
199 |         self.decrease_balance_internal(amount)?;
... |
261 |     /// Internal arithmetic for decreasing the balance of the liquid insurance fund
262 |     pub(crate) fn decrease_balance_internal(&mut self, amount: I80F48) -> MarginfiResult {
265 |         let current_amount = self.get_value(self.total_shares.into())?;
266 |         let delta_decrease = min(current_amount, amount);
268 |         let share_decrease = self.get_shares(delta_decrease)?;
270 |         // Remove shares from existing collection of shares
271 |         self.remove_shares(share_decrease)?;
```

Resolution

Fixed by commit `07c03f5`.

MARGINFI PR#238

[L-01] Liquid insurance fund initialization typo

The assignment at line 63, `self.admin_shares` refers to the `admin_shares` field to be assigned, instead of the value converted from the `balance` parameter at line 46.

As a result, the admin's share is always 0 and inconsistent with the `total_shares`.

```
/* programs/marginfi/src/state/liquid_insurance_fund.rs */
024 | pub struct LiquidInsuranceFund {
031 |     pub total_shares: WrappedI80F48,
036 |     pub admin_shares: WrappedI80F48,
043 | }

/* programs/marginfi/src/state/liquid_insurance_fund.rs */
045 | impl LiquidInsuranceFund {
046 |     pub fn initialize(
047 |         &mut self,
054 |         balance: u64,
055 |     ) {
056 |         let admin_shares = I80F48::from(balance);
057 |         *self = LiquidInsuranceFund {
061 |             total_shares: admin_shares.into(),
063 |             admin_shares: self.admin_shares.into(),
069 |         };
070 |     }
```

Resolution

Fixed by commit `f7af6ef`.

MARGINFI PR#238

[I-01] The "lending_account_withdraw" may fail in corner cases

In the `lending_account_withdraw` instruction, if the bank mint has a transfer fee extension, the current implementation treats the user-provided expected withdrawal amount as the post-fee amount. It then calculates a pre-fee amount based on the transfer fee rate and performs the withdrawal using this pre-fee amount.

This approach can lead to a corner case where the pre-fee amount exceeds the balance of the lending account but is greater than or equal to the user-provided amount. In this specific scenario, the withdrawal fails due to the restriction of `WithdrawOnly`.

```
/* programs/marginfi/src/instructions/marginfi_account/withdraw.rs */
071 | let amount_pre_fee = if withdraw_all {
072 |     bank_account.withdraw_all()?
073 | } else {
074 |     let amount_pre_fee = maybe_bank_mint
075 |         .as_ref()
076 |         .map(|mint| {
077 |             utils::calculate_pre_fee_spl_deposit_amount(
078 |                 mint.to_account_info(),
079 |                 amount,
080 |                 clock.epoch,
081 |             )
082 |         })
083 |         .transpose()?
084 |         .unwrap_or(amount);
085 |
086 |     bank_account.withdraw(I80F48::from_num(amount_pre_fee))?;
087 |
088 |     amount_pre_fee
089 | };
```

Resolution

The team acknowledged this finding.

MARGINFI PR#238

[I-02] Copy-paste error in "lending_pool_deposit_insurance"

In the implementation of the `lending_pool_deposit_insurance` instruction, there is a copy-paste error where `withdraw_spl_transfer`, instead of `deposit_spl_transfer`, is incorrectly used.

```
/* programs/marginfi/src/instructions/marginfi_group/collect_bank_fees.rs */
303 | pub fn lending_pool_deposit_insurance<'a, 'info>(
304 |     mut ctx: Context<'a, 'info, 'info, 'info, LendingPoolAdminDepositWithdrawInsurance<'info>>,
305 |     amount: u64,
306 | ) -> MarginfiResult {
341 |     bank.withdraw_spl_transfer(
342 |         amount,
343 |         admin_token_account.to_account_info(),
344 |         insurance_vault.to_account_info(),
345 |         admin.to_account_info(),
346 |         maybe_bank_mint.as_ref(),
347 |         token_program.to_account_info(),
348 |         bank_signer!(
349 |             BankVaultType::Insurance,
350 |             bank_loader.key(),
351 |             bank.insurance_vault_authority_bump
352 |         ),
353 |         ctx.remaining_accounts,
354 |     )?;
357 | }
```

Resolution

Fixed by commit `d988c7c`.

MARGINFI PR#238

[I-03] The "deposit_amount" in insurance event reflects pre-fee amount

The `deposit_amount` at line 122 in the `MarginfiDepositIntoLiquidInsuranceFundEvent` event is the amount before the transfer fees so it can be different from the amount deposited into insurance.

```
/* programs/marginfi/src/instructions/liquid_insurance_fund/deposit.rs */
055 | pub fn deposit_into_liquid_insurance_fund<'info>(
056 |     mut ctx: Context<'_, '_, 'info, 'info, DepositIntoLiquidInsuranceFund<'info>>,
057 |     deposit_amount: u64,
058 | ) -> MarginfiResult {
089 |     // 2) Calculate deposit_num_shares(deposit_amount)
090 |     let postfee_deposit_amount = maybe_bank_mint
091 |         .as_ref()
092 |         .map(|mint_ai| {
093 |             calculate_post_fee_spl_deposit_amount(
094 |                 mint_ai.to_account_info(),
095 |                 deposit_amount,
096 |                 clock.epoch,
097 |             )
098 |         })
099 |         .unwrap_or(Ok(deposit_amount))?;
118 |     emit!(MarginfiDepositIntoLiquidInsuranceFundEvent {
122 |         amount: deposit_amount,
123 |         signer_token_address: signer_token_account.key(),
124 |     });
```

Resolution

Fixed by commit `5ee4335`.

MARGINFI PR#238

[I-04] Validate "min_withdraw_period"

The `min_withdraw_period` should be larger than `0`.

```
/* programs/marginfi/src/instructions/liquid_insurance_fund/create_fund.rs */
061 | pub fn create_liquid_insurance_fund(
062 |     ctx: Context<CreateLiquidInsuranceFund>,
063 |     min_withdraw_period: i64,
064 | ) -> MarginfiResult {
079 |     lif.initialize(
083 |         min_withdraw_period,
087 |     );
```

Resolution

Fixed by commit `55224ac`.

MARGINFI PR#238

[I-05] Consider restricting token extensions when adding pools

Certain token 2022 extensions can lead to unexpected side effects. For instance, the permanent delegate extension allows the stored authority to transfer or burn tokens from any account.

It is recommended to whitelist supported extensions and validate the following mints when adding or configuring banks.

1. bank_mint in lending_pool_add_bank

```
/* programs/marginfi/src/instructions/marginfi_group/add_pool.rs */
074 | pub struct LendingPoolAddBank<'info> {
086 |     pub bank_mint: Box<InterfaceAccount<'info, Mint>>,
```

2. bank_mint in lending_pool_add_bank_with_seed

```
/* programs/marginfi/src/instructions/marginfi_group/add_pool.rs */
234 | pub struct LendingPoolAddBankWithSeed<'info> {
246 |     pub bank_mint: Box<InterfaceAccount<'info, Mint>>,
```

3. emissions_mint in lending_pool_setup_emissions

```
/* programs/marginfi/src/instructions/marginfi_group/configure_bank.rs */
099 | pub struct LendingPoolSetupEmissions<'info> {
114 |     pub emissions_mint: InterfaceAccount<'info, Mint>,
```

Resolution

The team acknowledged this finding.

MARGINFI PR#238

[I-06] Consider adding interface consistency check

Currently, it's possible to use `token-2022` program to process `spl-token` mint.

For example, it's better to add the program owner check for the `maybe_mint` account, which is supposed to be owned by the `token_2022` program.

```
/* programs/marginfi/src/utils.rs */
116 | pub fn maybe_take_bank_mint<'c: 'info, 'info>(
117 |     remaining_accounts: &mut &'c [AccountInfo<'info>],
118 |     bank_mint: &Pubkey,
119 |     token_program: &Pubkey,
120 | ) -> MarginfiResult<Option<InterfaceAccount<'info, Mint>>> {
121 |     match *token_program {
122 |         anchor_spl::token::ID => Ok(None),
123 |         anchor_spl::token_2022::ID => {
124 |             let (maybe_mint, remaining) = remaining_accounts
125 |                 .split_first()
126 |                 .ok_or(MarginfiError::T22MintRequired)?;
127 |             *remaining_accounts = remaining;
128 |
129 |             if *bank_mint != *maybe_mint.key {
130 |                 return err!(MarginfiError::T22MintRequired);
131 |             }
132 |
133 |             InterfaceAccount::try_from(maybe_mint)
134 |                 .map(Option::Some)
135 |                 .map_err(|e| {
136 |                     msg!("failed to parse mint account: {:?}", e);
137 |                     MarginfiError::T22MintRequired.into()
138 |                 })
139 |         }
140 |
141 |         _ => panic!("unsupported token program"),
142 |     }
143 | }
```

Resolution

The team clarified that as long as the `mint` provided by the user remains consistent with the bank, there is no security risk, even if an incorrect token program is supplied. Therefore, adding this check is deemed unnecessary.

MARGINFI PR#238

[I-07] Unused "last_update" in "liquid_insurance_fund"

The `last_update` is defined and initialized in `create_liquid_insurance_fund`, but it is never used.

```
/* programs/marginfi/src/state/liquid_insurance_fund.rs */
045 | impl LiquidInsuranceFund {
046 |     pub fn initialize(
047 |         &mut self,
055 |     ) {
057 |         *self = LiquidInsuranceFund {
064 |             last_update: i64::MIN,
069 |         };
070 |     }
```

Resolution

Fixed by commit `3d707b9`.

Appendix: Methodology and Scope of Work

Assisted by the Sec3 Scanner developed in-house, the manual audit particularly focused on the following work items:

- Check common security issues.
- 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 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 MRGN, Inc. (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 sole purpose of the Report 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

The Sec3 audit team comprises a group of computer science professors, researchers, and industry veterans with extensive experience in smart contract security, program analysis, testing, and formal verification. We are also building automated security tools that incorporate 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.