Security Assessment Report

Marginfi v2

December 14, 2023

Summary

The sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Marginfi v2 on-chain smart contract program at https://github.com/mrgnlabs/marginfi-v2. The initial audit was done on the source code of the following version:

- commit a4a7182f741857bdffe0e7338bc584e1f8b1e390
- PR #121

The review revealed 10 issues and questions. 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 516637404c2db54d41022b7deb9cfd627aa2a824, which is the version with all fixes applied.

Table of Contents

Result Overview

MARGINFI V2 PROGRAM		
Issue	**Impact**	**Status**
[L-1] Inconsistent prices used in liquidation assessment and settlement	Low	Resolved
[L-2] Whether bank is paused is not checked in withdraw_all and repay_all	Low	Resolved
[L-3] Bad debts may be not fully covered in some cases	Low	Resolved
[I-1] No easy ways to close balance account with low balance	Info	Resolved
[I-2] Interest_rate may be larger than max_ir	Info	Resolved
[I-3] Price oracle consistency	Info	Resolved
[I-4] Actual APY may be larger than expected	Info	Resolved
[I-5] Optimization opportunity for lending_account_liquidate(ctx, 0)	Info	Resolved
[Q-1] Is it intentional to ignore RiskTier check during flashloan?	Question	Resolved
[Q-2] Implicitly avoid asset_bank == liab_bank in the liquidation?	Question	Resolved

Findings in Detail

[L-1] Inconsistent prices used in liquidation assessment and settlement

In the liquidation assessment check_pre_liquidation_condition_and_get_account_health(), the lower-bound EMA price (say A1) is used to calculate the asset value, and the higher-bound EMA price (say L1) is used to calculate the liability value. A position can be liquidated when the total asset value is lower than the total liability value.

```
/* programs/marginfi/src/state/marginfi_account.rs */
375 | pub fn check_pre_liquidation_condition_and_get_account_health(
376 |     &self,
377 |     bank_pk: &Pubkey,
378 | ) -> MarginfiResult<I80F48> {
397 |     let (assets, liabs) =
398 |         self.get_account_health_components(RiskRequirementType::Maintenance)?;
400 |     let account_health = assets.checked_sub(liabs).ok_or_else(math_error!())?;

/* programs/marginfi/src/state/marginfi_account.rs */
320 | pub fn get_account_health_components(
321 |     &self,
322 |     requirement_type: RiskRequirementType,
323 | ) -> MarginfiResult<(I80F48, I80F48)> {
327 |     for a in &self.bank_accounts_with_price {
328 |         let (assets, liabilities) =
329 |             a.calc_weighted_assets_and_liabilities_values(
                        requirement_type.to_weight_type())?;
335 |     }
338 | }

/* programs/marginfi/src/state/marginfi_account.rs */
166 | pub fn calc_weighted_assets_and_liabilities_values(
167 |     &self,
168 |     weight_type: WeightType,
169 | ) -> MarginfiResult<(I80F48, I80F48)> {
170 |     let (worst_price, best_price) = self.price_feed.get_price_range()?;
216 |     Ok((
217 |         calc_asset_value(asset_amount, worst_price, mint_decimals,
                                Some(asset_weight))?,
218 |         calc_asset_value(
```

```
219 |             liability_amount,
220 |             best_price,
221 |             mint_decimals,
222 |             Some(liability_weight),
223 |         )?,
224 |     ))
225 | }
```

However, during the settlement, the EMA prices are used for assets (say A2) and liabilities (say L2). Therefore, compared to the prices used in the liquidation assessment, the asset price is higher (A2 > A1), and the liability price is lower (L2 < L1).

```
/* programs/marginfi/src/instructions/marginfi_account/liquidate.rs */
112 | let mut asset_bank = ctx.accounts.asset_bank.load_mut()?;
113 | let asset_price = {
114 |     let oracle_ais = &ctx.remaining_accounts[0..1];
115 |     let asset_pf = OraclePriceFeedAdapter::try_from_bank_config(
116 |         &asset_bank.config,
117 |         oracle_ais,
118 |         current_timestamp,
119 |         MAX_PRICE_AGE_SEC,
120 |     )?;
121 |     asset_pf.get_price()?
122 | };
123 |
124 | let mut liab_bank = ctx.accounts.liab_bank.load_mut()?;
125 | let liab_price = {
126 |     let oracle_ais = &ctx.remaining_accounts[1..2];
127 |     let liab_pf = OraclePriceFeedAdapter::try_from_bank_config(
128 |         &liab_bank.config,
129 |         oracle_ais,
130 |         current_timestamp,
131 |         MAX_PRICE_AGE_SEC,
132 |     )?;
133 |
134 |     liab_pf.get_price()?
135 | };
```

The price inconsistency could lead to two issues:

1. The implementation that underestimates the assets and overestimates the liabilities in the liquidation eligibility check is consistent with the documentation for a lower

risk. However, this design might potentially result in the liquidation of certain marginally healthy accounts, i.e., the account is healthy if evaluated using EMA prices (A2 and L2) but unhealthy when evaluated using EMA confidence-internal based prices (A1 and L1), especially when there are drastic price changes. If using the same EMA confidence-interval prices in the liquidation pre-check and settlement will eliminate this gap.

2. Using the EMA prices (A2 and L2) in the liquidation settlement will benefit the liquidatees but hurt liquidators. In particular, according to how the amount of liability the liquidator has to pay (`liab_amount_liquidator`), `asset_amount * A2 / L2` is always larger than `asset_amount * A1 / L1`, which means the liquidator has to pay more.

```
/* programs/marginfi/src/instructions/marginfi_account/liquidate.rs */
140 | // Quantity of liability to be paid off by liquidator
141 | let liab_amount_liquidator = calc_asset_amount(
142 |     calc_asset_value(
143 |         asset_amount,
144 |         asset_price,
145 |         asset_bank.mint_decimals,
146 |         Some(liquidator_discount),
147 |     )?,
148 |     liab_price,
149 |     liab_bank.mint_decimals,
150 | )?;
```

It may be a good idea to use the EMA confidence-interval prices in the settlement, too. In this way, it is more appealing to the liquidators and also lowers the risk for the protocol. At the same time, although it's not friendly to the liquidatees, it discourages operations that may lead to liquidations.

Resolution

This issue has been addressed by the commit `d9eeec9`.

[L-2] Whether bank is paused is not checked in withdraw_all and repay_all

```
* programs/marginfi/src/state/marginfi_group.rs */
733 | pub enum BankOperationalState {
734 |     Paused,
735 |     Operational,
736 |     ReduceOnly,
737 | }

/* programs/marginfi/src/instructions/marginfi_account/withdraw.rs */
025 | pub fn lending_account_withdraw(
029 | ) -> MarginfiResult {
064 |         let spl_withdraw_amount = if withdraw_all {
065 |             bank_account.withdraw_all()?
066 |         } else {
067 |             bank_account.withdraw(I80F48::from_num(amount))?;
070 |         };

/* programs/marginfi/src/state/marginfi_account.rs */
718 | pub fn withdraw(&mut self, amount: I80F48) -> MarginfiResult {
719 |     self.decrease_balance_internal(amount, BalanceDecreaseType::WithdrawOnly)
720 | }

/* programs/marginfi/src/state/marginfi_account.rs */
914 | fn decrease_balance_internal(
918 | ) -> MarginfiResult {
961 |     {
962 |         let is_liability_amount_increasing =
963 |          liability_amount_increase.is_positive_with_tolerance(ZERO_AMOUNT_THRESHOLD);
964 |         bank.assert_operational_mode(Some(is_liability_amount_increasing))?;
965 |     }
981 | }
```

In withdraw(), the bank operation mode, especially Paused, is checked at marginfi_account.rs:L964. However, in withdraw_all(), the bank operation mode is not checked.

Similarly, in repay_all(), the bank operation mode is not checked, either.

Resolution

This issue has been addressed by the commit ed86fbe.

[L-3] Bad debts may be not fully covered in some cases

```
/* programs/marginfi/src/instructions/marginfi_group/handle_bankruptcy.rs */
026 | pub fn lending_pool_handle_bankruptcy(ctx: Context<LendingPoolHandleBankruptcy>)
            -> MarginfiResult {
060 |     let bad_debt = bank.get_liability_amount(
                    lending_account_balance.liability_shares.into())?;
067 |     let (covered_by_insurance, socialized_loss) = {
068 |         let available_insurance_funds = I80F48::from_num(insurance_vault.amount);
070 |         let covered_by_insurance = min(bad_debt, available_insurance_funds);
071 |         let socialized_loss = max(bad_debt - covered_by_insurance, I80F48::ZERO);
073 |         (covered_by_insurance, socialized_loss)
074 |     };
077 |     bank.withdraw_spl_transfer(
078 |         covered_by_insurance
079 |             .checked_to_num()
080 |             .ok_or_else(math_error!())?,
081 |         Transfer {
082 |             from: ctx.accounts.insurance_vault.to_account_info(),
083 |             to: ctx.accounts.liquidity_vault.to_account_info(),
084 |             authority: ctx.accounts.insurance_vault_authority.to_account_info(),
085 |         },
086 |         token_program.to_account_info(),
087 |         bank_signer!(
088 |             BankVaultType::Insurance,
089 |             bank_loader.key(),
090 |             bank.insurance_vault_authority_bump
091 |         ),
092 |     )?;
095 |     bank.socialize_loss(socialized_loss)?;
099 |     BankAccountWrapper::find_or_create(
100 |         &bank_loader.key(),
101 |         &mut bank,
102 |         &mut marginfi_account.lending_account,
103 |     )?
104 |     .repay(bad_debt)?;
```

When handling the bankruptcy, if the bad debt can be entirely covered by insurance, due to precision limitations, the `checked_to_num` operation on line 79 results in the transfer amount to the liquidity vault being less than the actual bad debt amount.

Resolution

The team acknowledged this issue and clarified that the conversion loss should be trivial. Consequently, the team believes it is not a concern.

[I-1] No easy ways to close balance account with low balance

```
/* programs/marginfi/src/state/marginfi_account.rs */
751 | pub fn withdraw_all(&mut self) -> MarginfiResult<u64> {
767 |     check!(
768 |         current_asset_amount.is_positive_with_tolerance(ZERO_AMOUNT_THRESHOLD),
769 |         MarginfiError::NoAssetFound
770 |     );
771 |
772 |     check!(
773 |         current_liability_amount.is_zero_with_tolerance(ZERO_AMOUNT_THRESHOLD),
774 |         MarginfiError::NoAssetFound
775 |     );
776 |
777 |     balance.close()?;

/* programs/marginfi/src/state/marginfi_account.rs */
801 | pub fn repay_all(&mut self) -> MarginfiResult<u64> {
816 |     check!(
817 |         current_liability_amount.is_positive_with_tolerance(ZERO_AMOUNT_THRESHOLD),
818 |         MarginfiError::NoLiabilityFound
819 |     );
820 |
821 |     check!(
822 |         current_asset_amount.is_zero_with_tolerance(ZERO_AMOUNT_THRESHOLD),
823 |         MarginfiError::NoLiabilityFound
824 |     );
825 |
826 |     balance.close()?;
```

In the current implementation, if one wishes to close a `balance` account, it can only be achieved through the methods of `repay_all` or `withdraw_all`.

However, both of these methods require that the corresponding `liability_amount` and `asset_amount` be greater than the "`ZERO_AMOUNT_THRESHOLD`".

Once a user manually reduces the respective amount to below this threshold by invoking `repay` or `withdraw`, they will not have a direct means to close the `balance` account.

Furthermore, if the corresponding bank is in a `ReduceOnly` state, the user may even be unable to close the balance account.

Resolution

This issue has been addressed by the commit `1529657`.

[I-2] Interest_rate may be larger than max_ir

The interest rates are updated by `accrue_interest()` in most business-related operations. In particular, they are calculated by `calc_interest_rate_accrual_state_changes()`.

```
/* programs/marginfi/src/state/marginfi_group.rs */
461 | pub fn accrue_interest(
462 |     &mut self,
463 |     current_timestamp: i64,
464 |     #[cfg(not(feature = "client"))] bank: Pubkey,
465 | ) -> MarginfiResult<()> {
475 |     let total_assets = self.get_asset_amount(self.total_asset_shares.into())?;
476 |     let total_liabilities = self.get_liability_amount(
                                    self.total_liability_shares.into())?;
478 |     self.last_update = current_timestamp;

497 |     let (asset_share_value, liability_share_value,
              fees_collected, insurance_collected) =
498 |         calc_interest_rate_accrual_state_changes(
499 |             time_delta,
500 |             total_assets,
501 |             total_liabilities,
502 |             &self.config.interest_rate_config,
503 |             self.asset_share_value.into(),
504 |             self.liability_share_value.into(),
505 |         )
506 |         .ok_or_else(math_error!())?;
511 |     self.asset_share_value = asset_share_value.into();
512 |     self.liability_share_value = liability_share_value.into();
545 |     Ok(())
546 | }
```

When calculating the interest rates, by design, the `borrowing_rate` (`borrowing_apr`) is larger than the `lending_rate` (`lending_apr`).

Therefore, for a bank starting with `utilization_rate < 1` at `marginfi_group.rs:L673`, since the `lending_apr` is smaller than `borrowing_apr`, the `liability_share_value` at `marginfi_group.rs:L512` grows faster than `asset_share_value`. In marginal scenarios, the `utilization_rate` may be larger than 1 after the update.

```
/* programs/marginfi/src/state/marginfi_group.rs */
665 | fn calc_interest_rate_accrual_state_changes(
666 |     time_delta: u64,
667 |     total_assets_amount: I80F48,
668 |     total_liabilities_amount: I80F48,
669 |     interest_rate_config: &InterestRateConfig,
670 |     asset_share_value: I80F48,
671 |     liability_share_value: I80F48,
672 | ) -> Option<(I80F48, I80F48, I80F48, I80F48)> {
673 |     let utilization_rate = total_liabilities_amount.checked_div(total_assets_amount)?;
674 |     let (lending_apr, borrowing_apr, group_fee_apr, insurance_fee_apr) =
675 |         interest_rate_config.calc_interest_rate(utilization_rate)?;
687 |     Some((
688 |         calc_accrued_interest_payment_per_period(lending_apr, time_delta, asset_share_value)?,
689 |         calc_accrued_interest_payment_per_period(borrowing_apr, time_delta,
                                                     liability_share_value)?,
692 |     ))
693 | }

/* programs/marginfi/src/state/marginfi_group.rs */
102 | pub fn calc_interest_rate(
103 |     &self,
104 |     utilization_ratio: I80F48,
105 | ) -> Option<(I80F48, I80F48, I80F48, I80F48)> {
115 |     let base_rate = self.interest_rate_curve(utilization_ratio)?;
116 |
117 |     // Lending rate is adjusted for utilization ratio to symmetrize payments between
        // borrowers and depositors.
118 |     let lending_rate = base_rate.checked_mul(utilization_ratio)?;
119 |
120 |     // Borrowing rate is adjusted for fees.
121 |     // borrowing_rate = base_rate + base_rate * rate_fee + total_fixed_fee_apr
122 |     let borrowing_rate = base_rate
123 |         .checked_mul(I80F48::ONE.checked_add(rate_fee)?)?
124 |         .checked_add(total_fixed_fee_apr)?;
138 |     assert!(lending_rate >= I80F48::ZERO);
139 |     assert!(borrowing_rate >= I80F48::ZERO);
143 |     // TODO: Add liquidation discount check
145 |     Some((
146 |         lending_rate,
147 |         borrowing_rate,
148 |         group_fees_apr,
149 |         insurance_fees_apr,
150 |     ))
151 | }
```

As a result, when `ur > 1`, the base interest may be larger than `max_ir`.

```
/* programs/marginfi/src/state/marginfi_group.rs */
159 | fn interest_rate_curve(&self, ur: I80F48) -> Option<I80F48> {
160 |     let optimal_ur = self.optimal_utilization_rate.into();
161 |     let plateau_ir = self.plateau_interest_rate.into();
162 |     let max_ir: I80F48 = self.max_interest_rate.into();
163 |
164 |     if ur <= optimal_ur {
165 |         ur.checked_div(optimal_ur)?.checked_mul(plateau_ir)
166 |     } else {
167 |         (ur - optimal_ur)
168 |             .checked_div(I80F48::ONE - optimal_ur)?
169 |             .checked_mul(max_ir - plateau_ir)?
170 |             .checked_add(plateau_ir)
171 |     }
172 | }
```

However, since the interest rates are updated in each major operation, it doesn't make sense to abort. Otherwise, operations related to these banks will fail and get stuck.

Resolution

The team acknowledged this issue and clarified that, given the piecewise nature of the IR curve, the consequences should be minimal. Consequently, the team believes it is not a concern.

[I-3] Price oracle consistency

```
/* programs/marginfi/src/state/price.rs */
221 | fn get_confidence_interval(&self) -> MarginfiResult<I80F48> {
222 |     let std_div = self.aggregator_account.latest_confirmed_round_std_deviation;
223 |     let std_div = swithcboard_decimal_to_i80f48(std_div)
224 |         .ok_or(MarginfiError::InvalidSwitchboardDecimalConversion)?;
225 |
226 |     let conf_interval = std_div
227 |         .checked_mul(CONF_INTERVAL_MULTIPLE)
228 |         .ok_or_else(math_error!())?;
229 |
230 |     assert!(
231 |         conf_interval >= I80F48::ZERO,
232 |         "Negative confidence interval"
233 |     );
234 |
235 |     Ok(conf_interval)
236 | }
```

1. 95% confidence intervals

In the current code implementation, there are two available options for configuring price oracles: Pyth and Switchboard v2. However, it's essential to be attentive to certain differences in the implementation details of these two price oracles. In the case of Pyth, the returned price follows a Laplace distribution, and, therefore, multiplying the returned standard deviation by 2.12 can compute the 95% confidence interval range. Conversely, in Switchboard v2, the returned `std_deviation` represents a basic standard deviation calculated from various price sources in the price feed. Therefore, multiplying this `std_deviation` by 2.12 to define a confidence interval may not be as appropriate.

2. EMA/TWAP in Switchboard

Furthermore, the current Pyth implementation is consistent with the design outlined in the documentation, utilizing Exponential Moving Average (EMA) prices. However, for

Switchboard, the Time-Weighted Average Price (TWAP) needs to be configured within the feed independently.

3. Minimum number of price sources

To ensure the reliability of the price oracle and avoid price manipulation attacks (e.g. single price source in the feed). It's crucial to pay attentions to the minimum number of publishers in price feeds, token/price consistency, etc.

Resolution

The team acknowledged this issue.

[I-4] Actual APY may be larger than expected

```
/* programs/marginfi/src/state/marginfi_group.rs */
461 | pub fn accrue_interest(
462 |     &mut self,
463 |     current_timestamp: i64,
464 |     #[cfg(not(feature = "client"))] bank: Pubkey,
465 | ) -> MarginfiResult<()> {
466 |     #[cfg(not(feature = "client"))]
467 |     solana_program::log::sol_log_compute_units();
468 |
469 |     let time_delta: u64 = (current_timestamp - self.last_update).try_into().unwrap();
470 |
471 |     if time_delta == 0 {
472 |         return Ok(());
473 |     }
474 |
475 |     let total_assets = self.get_asset_amount(self.total_asset_shares.into())?;
476 |     let total_liabilities = self.get_liability_amount(self.total_liability_shares.into())?;
477 |
478 |     self.last_update = current_timestamp;
496 |
497 |     let (asset_share_value, liability_share_value, fees_collected, insurance_collected) =
498 |         calc_interest_rate_accrual_state_changes(
499 |             time_delta,
500 |             total_assets,
501 |             total_liabilities,
502 |             &self.config.interest_rate_config,
503 |             self.asset_share_value.into(),
504 |             self.liability_share_value.into(),
505 |         )
506 |         .ok_or_else(math_error!())?;
510 |
511 |     self.asset_share_value = asset_share_value.into();
512 |     self.liability_share_value = liability_share_value.into();
546 | }

/* programs/marginfi/src/state/marginfi_group.rs */
705 | fn calc_accrued_interest_payment_per_period(
706 |     apr: I80F48,
707 |     time_delta: u64,
708 |     value: I80F48,
709 | ) -> Option<I80F48> {
710 |     let ir_per_period = apr
711 |         .checked_mul(time_delta.into())?
712 |         .checked_div(SECONDS_PER_YEAR)?;
```

```
713 |
714 |     let new_value = value.checked_mul(I80F48::ONE.checked_add(ir_per_period)?)?;
715 |
716 |     Some(new_value)
717 | }
```

In actual interest calculations, for banks with significant trading volumes, their interest may be computed quite frequently, and each calculation involves an update to the share values. Subsequent calculations are based on the previous share values, and this compounding interest effect can result in the actual APY being greater than expected.

Taking an extreme example, suppose `accrue_interest` is called every second. In such a case, when the expected APY is 5% and 10%, the actual APY would be 5.13% and 10.52%.

```
In [1]: pow(1+0.05/31_536_000, 31_536_000)
Out[1]: 1.0512710936245873

In [2]: pow(1+0.1/31_536_000, 31_536_000)
Out[2]: 1.1051709199418713
```

Resolution

The team has acknowledged this issue and mentioned that they have a to-do item to implement a compounding accrual function using the Taylor series. However, it is currently of low priority.

[I-5] Optimization opportunity for lending_account_liquidate(ctx, 0)

Consider directly returning when `asset_amount = 0` instead of failing at the post-risk check.

```
/* programs/marginfi/src/instructions/marginfi_account/liquidate.rs */
070 | pub fn lending_account_liquidate(
071 |     ctx: Context<LendingAccountLiquidate>,
072 |     asset_amount: u64,
073 | ) -> MarginfiResult {
```

Resolution

This issue has been addressed by the commit `2a94625`.

[Q-1] Is it intentional to ignore RiskTier check during flashloan?

Just curious about the design. The only difference is how isolated banks can be used in the flash loan.

Because it will be checked at the end of the flash loan, there is no risk.

Resolution

The team clarified that this is an intentional design choice.

[Q-2] Implicitly avoid asset_bank == liab_bank in the liquidation?

```
/* programs/marginfi/src/instructions/marginfi_account/liquidate.rs */
112 | let mut asset_bank = ctx.accounts.asset_bank.load_mut()?;
124 | let mut liab_bank = ctx.accounts.liab_bank.load_mut()?;
365 |
366 | #[account(
367 |     mut,
368 |     constraint = asset_bank.load()?.group == marginfi_group.key()
369 | )]
370 | pub asset_bank: AccountLoader<'info, Bank>,
371 |
372 | #[account(
373 |     mut,
374 |     constraint = liab_bank.load()?.group == marginfi_group.key()
375 | )]
376 | pub liab_bank: AccountLoader<'info, Bank>,
```

In the current implementation of the `liquidate` instruction, there is no explicit check in place to prevent users from providing the same bank address for both `asset_bank` and `liab_bank`.

```
/* marginfi-v2/programs/marginfi/src/instructions/marginfi_account/liquidate.rs */
080 | let mut liquidator_marginfi_account = liquidator_marginfi_account_loader.load_mut()?;
081 | let mut liquidatee_marginfi_account = liquidatee_marginfi_account_loader.load_mut()?

/* marginfi-v2/programs/marginfi/src/instructions/marginfi_account/liquidate.rs */
085 | ctx.accounts.asset_bank.load_mut()?.accrue_interest()
090 | ctx.accounts.liab_bank.load_mut()?.accrue_interest()

/*  anchor-lang-0.26.0/src/accounts/account_loader.rs */
170 | pub fn load_mut(&self) -> Result<RefMut<T>> {
177 |     let data = self.acc_info.try_borrow_mut_data()?;
190 | }

/* .cargo/registry/src/index.crates.io-6f17d22bba15001f/solana-program-1.14.18/src/account_info.rs */
123 | pub fn try_borrow_mut_data(&self) -> Result<RefMut<&'a mut [u8]>, ProgramError> {
124 |     self.data
125 |         .try_borrow_mut()
126 |         .map_err(|_| ProgramError::AccountBorrowFailed)
127 | }

/* rustlib/src/rust/library/core/src/cell.rs */
1020 | /// Mutably borrows the wrapped value, returning an error if the value is currently borrowed.
1045 | pub fn try_borrow_mut(&self) -> Result<RefMut<'_, T>, BorrowMutError> {
```

However, at runtime, the borrow checker raises an error message stating, "Failed to borrow a reference to account data, already borrowed".

We were curious whether it is an expected behavior to rely on the borrow checker to prevent users from providing the same bank address for both `asset_bank` and `liab_bank`.

Similarly, `liquidator_marginfi_account` and `liquidatee_marginfi_account` cannot refer to the same account.

Resolution

The commit `1ace78e` has resolved this question by incorporating an explicit check.

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 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 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™