Security Assessment Report

Marginfi V2 PR248

January 28, 2025

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Marginfi V2 PR248.

The artifact of the audit was the source code of the following programs, excluding tests, in https://github.com/mrgnlabs/marginfi-v2/pull/248.

The initial audit focused on the following versions and revealed 4 issues or questions.

program	type	commit
Marginfi V2 PR248	Solana	2fd6e2c → 60e02ae

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

Table of Contents

Result Overview

Issue	Impact	Status
MARGINFI V2 PR248		
[H-01] Unvalidated "bank_asset_tag" in liquidation	High	Resolved
[M-01] The "StakedWithPythPush" is not handled in the "validate_oracle_setup"	Medium	Resolved
[I-01] Validate "staked_setting" parameters	Info	Resolved
[I-02] Validate incoming oracle parameters	Info	Resolved

Findings in Detail

MARGINFI V2 PR248

[H-01] Unvalidated "bank_asset_tag" in liquidation

During the liquidation process, the liquidator pays assets of type "liab_bank" and receives assets of type "asset_bank". Conversely, the liquidatee pays assets of type "asset_bank" and receives assets of type "liab_bank".

In the liquidation process, the "find_or_create" function is used to construct a "bank_account". This function searches the "lending_account" for an existing balance corresponding to the given bank. If no such balance exists, a new one will be created.

However, during the creation of a new balance, there is no check to ensure that the "bank_asset_tag" of the new balance matches the "bank_asset_tag" of the existing balances.

```
/* programs/marginfi/src/instructions/marginfi_account/liquidate.rs */
250 | let mut bank_account = BankAccountWrapper::find_or_create(
251 |     &ctx.accounts.asset_bank.key(),
252 |     &mut asset_bank,
253 |     &mut liquidator_marginfi_account.lending_account,
254 | )?;

/* programs/marginfi/src/instructions/marginfi_account/liquidate.rs */
280 | let mut liquidatee_liab_bank_account = BankAccountWrapper::find_or_create(
281 |     &ctx.accounts.liab_bank.key(),
282 |     &mut liab_bank,
283 |     &mut liquidatee_marginfi_account.lending_account,
284 | )?;

/* programs/marginfi/src/state/marginfi_account.rs */
884 | pub fn find_or_create(
885 |     bank_pk: &Pubkey,
886 |     bank: &'a mut Bank,
887 |     lending_account: &'a mut LendingAccount,
888 | ) -> MarginfiResult<BankAccountWrapper<'a>> {
889 |     let balance_index = lending_account
890 |         .balances
891 |         .iter()
892 |         .position(|balance| balance.active && balance.bank_pk.eq(bank_pk));
893 |
894 |     match balance_index {
895 |         Some(balance_index) => {
896 |             let balance = lending_account
```

```
897 |                 .balances
898 |                 .get_mut(balance_index)
899 |                 .ok_or_else(|| error!(MarginfiError::BankAccountNotFound))?;
900 |
901 |             Ok(Self { balance, bank })
902 |         }
903 |         None => {
904 |             let empty_index = lending_account
905 |                 .get_first_empty_balance()
906 |                 .ok_or_else(|| error!(MarginfiError::LendingAccountBalanceSlotsFull))?;
907 |
908 |             lending_account.balances[empty_index] = Balance {
909 |                 active: true,
910 |                 bank_pk: *bank_pk,
911 |                 bank_asset_tag: bank.config.asset_tag,
912 |                 _pad0: [0; 6],
913 |                 asset_shares: I80F48::ZERO.into(),
914 |                 liability_shares: I80F48::ZERO.into(),
915 |                 emissions_outstanding: I80F48::ZERO.into(),
916 |                 last_update: Clock::get()?.unix_timestamp as u64,
917 |                 _padding: [0; 1],
918 |             };
919 |
920 |             Ok(Self {
921 |                 balance: lending_account.balances.get_mut(empty_index).unwrap(),
922 |                 bank,
923 |             })
924 |         }
925 |     }
926 | }
```

While borrow and deposit operations validate "`bank_asset_tag`" to prevent "`DEFAULT`" and "`STAKED`" assets from coexisting in the same "`marginfi_account`", this validation is missing in the liquidation process.

```
/* programs/marginfi/src/utils.rs */
202 | pub fn validate_asset_tags(bank: &Bank, marginfi_account: &MarginfiAccount) -> MarginfiResult {
203 |     let mut has_default_asset = false;
204 |     let mut has_staked_asset = false;
205 |
206 |     for balance in marginfi_account.lending_account.balances.iter() {
207 |         if balance.active {
208 |             match balance.bank_asset_tag {
209 |                 ASSET_TAG_DEFAULT => has_default_asset = true,
210 |                 ASSET_TAG_SOL => { /* Do nothing, SOL can mix with any asset type */ }
211 |                 ASSET_TAG_STAKED => has_staked_asset = true,
212 |                 _ => panic!("unsupported asset tag"),
213 |             }
214 |         }
215 |     }
216 |
```

```
217 |     // 1. Regular assets (DEFAULT) cannot mix with Staked assets
218 |     if bank.config.asset_tag == ASSET_TAG_DEFAULT && has_staked_asset {
219 |         return err!(MarginfiError::AssetTagMismatch);
220 |     }
221 |
222 |     // 2. Staked SOL cannot mix with Regular asset (DEFAULT)
223 |     if bank.config.asset_tag == ASSET_TAG_STAKED && has_default_asset {
224 |         return err!(MarginfiError::AssetTagMismatch);
225 |     }
```

For example, suppose the liquidator's asset (`"liab_bank"`) has the `"DEFAULT"` tag, while the liquidatee's asset (`"asset_bank"`) has the `"STAKED"` tag.

In this scenario, the liquidator creates a `"STAKED"` balance in the `"liquidator_marginfi_account"`, while the liquidatee creates a balance with the `"DEFAULT"` tag in the `"liquidatee_marginfi_account"`.

As a result, both the liquidator and the liquidatee end up with balances containing conflicting `"bank_asset_tag"` values (`"DEFAULT"` and `"STAKED"`) in their respective accounts. This inconsistency causes any future `"borrow"` or `"deposit"` operation to fail during the `"validate_asset_tags"` check, unless the input asset is `"SOL"`.

To prevent such conflicts, consider adding a validation step in the `"liquidate"` function to ensure that `"liab_bank"` and `"asset_bank"` share the same `"bank_asset_tag"`.

Resolution

Fixed by commit [989f21c](989f21c).

MARGINFI V2 PR248

[M-01] The "StakedWithPythPush" is not handled in the "validate_oracle_setup"

In the "validate_oracle_setup" function, if "oracle_setup" is set to "StakedWithPythPush", the function "bank_config.get_pyth_push_oracle_feed_id()" is called at line 304 to fetch the "FeedId".

```
/* programs/marginfi/src/state/price.rs */
296 | OracleSetup::StakedWithPythPush => {
297 |     if lst_mint.is_some() && stake_pool.is_some() && sol_pool.is_some() {
298 |         check!(oracle_ais.len() == 3, MarginfiError::InvalidOracleAccount);
299 |
300 |         // Note: mainnet/staging/devnet use "push" oracles, localnet uses legacy
301 |         if live!() {
302 |             PythPushOraclePriceFeed::check_ai_and_feed_id(
303 |                 &oracle_ais[0],
304 |                 bank_config.get_pyth_push_oracle_feed_id().unwrap(),
305 |             )?;
306 |         } else {
307 |             // Localnet only
308 |             check!(
309 |                 oracle_ais[0].key == &bank_config.oracle_keys[0],
310 |                 MarginfiError::InvalidOracleAccount
311 |             );
312 |
313 |             PythLegacyPriceFeed::check_ais(&oracle_ais[0])?;
314 |         }
```

However, as shown below, the "get_pyth_push_oracle_feed_id()" function does not handle the "OracleSetup::StakedWithPythPush" case. Therefore, it returns "None", leading to a runtime error at line 304.

```
/* programs/marginfi/src/state/marginfi_group.rs */
1466 | pub fn get_pyth_push_oracle_feed_id(&self) -> Option<&FeedId> {
1467 |     if matches!(self.oracle_setup, OracleSetup::PythPushOracle) {
1468 |         let bytes: &[u8; 32] = self.oracle_keys[0].as_ref().try_into().unwrap();
1469 |         Some(bytes)
1470 |     } else {
1471 |         None
1472 |     }
1473 | }
```

As a result, the "lending_pool_add_bank_permissionless" will fail in the "validate_oracle_setup" process, preventing the creation of a bank account.

To address this issue, it is recommended to extend the "get_pyth_push_oracle_feed_id()" func-

tion to include handling for the "`StakedWithPythPush`" case.

Resolution

Fixed by commit [989f21c](989f21c).

MARGINFI V2 PR248

[I-01] Validate "staked_setting" parameters

The `"staked_settings"` account is used during the initialization of a `"bank"` account to configure the bank's settings. Once the bank config is set, the `"bank.config.validate()"` method is called to verify the parameters.

To ensure the validation process passes successfully, the parameters within the `"staked_settings"` account should satisfy the following conditions:

```
check!(
    asset_init_w >= I80F48::ZERO && asset_init_w <= I80F48::ONE,
    MarginfiError::InvalidConfig
);
check!(asset_maint_w >= asset_init_w, MarginfiError::InvalidConfig);

if self.risk_tier == RiskTier::Isolated {
    check!(asset_init_w == I80F48::ZERO, MarginfiError::InvalidConfig);
    check!(asset_maint_w == I80F48::ZERO, MarginfiError::InvalidConfig);
}
```

Consider validating the parameters when creating and updating the `"staked_settings"` account.

Resolution

Fixed by commit 989f21c.

MARGINFI V2 PR248

[I-02] Validate incoming oracle parameters

The "`validate_oracle_setup()`" in line 15 validates the "`bank.config.oracle_max_age`" and "`bank.config.oracle_keys`". However, since they are not updated before "`validate_oracle_setup()`", the incoming "`settings.oracle`" and "`settings.oracle_max_age`" are not validated.

```
/* programs/marginfi/src/instructions/marginfi_group/propagate_staked_settings.rs */
008 | pub fn propagate_staked_settings(ctx: Context<PropagateStakedSettings>) -> Result<()> {
012 |     // Only validate the oracle if it has changed
013 |     if settings.oracle != bank.config.oracle_keys[0] {
014 |         bank.config
015 |             .validate_oracle_setup(ctx.remaining_accounts, None, None, None)?;
016 |     }
018 |     bank.config.oracle_keys[0] = settings.oracle;
023 |     bank.config.oracle_max_age = settings.oracle_max_age;
```

The updates to "`bank.config.oracle_keys[0]`" and "`bank.config.oracle_max_age`" should be made before calling "`validate_oracle_setup()`" to ensure they are validated.

Resolution

Fixed by commit 06151a0.

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.