Marginfi Integration

Security Assessment

October 16th, 2024 — Prepared by OtterSec

Ajay Shankar Kunapareddy d1r3wolf@osec.io

Akash Gurugunti sud0u53r.ak@osec.io

Robert Chen r@osec.io

Table of Contents

01 — Executive Summary

Overview

Exponent Finance engaged OtterSec to assess the `marginfi-integration` program. This assessment was conducted between August 24th and October 9th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 7 findings throughout this audit engagement.

In particular, we identified a vulnerability in the functionality for withdrawing from a lending account, where the withdrawal parameter is incorrectly encoded as a single byte, while the Marginfi program expects a two-byte parameter (OS-MGI-ADV-02). Furthermore, due to a potential discrepancy while redeeming SY, the shares are inaccurately converted to the base asset, especially if the asset share value is updated in the Marginfi withdraw instruction (OS-MGI-ADV-03). Additionally, the add emission instruction in the Marginfi-standard program may designate any account as the escrow account, even if it is not an associated token account (OS-MGI-SUG-00).

We also made recommendations to incorporate additional checks within the codebase for improved robustness and security (OS-MGI-SUG-01) and suggested the removal of unutilized and redundant code within the system for increased readability (OS-MGI-SUG-02).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/exponent-finance/exponent-core. This audit was performed against commit a9d3a6b.

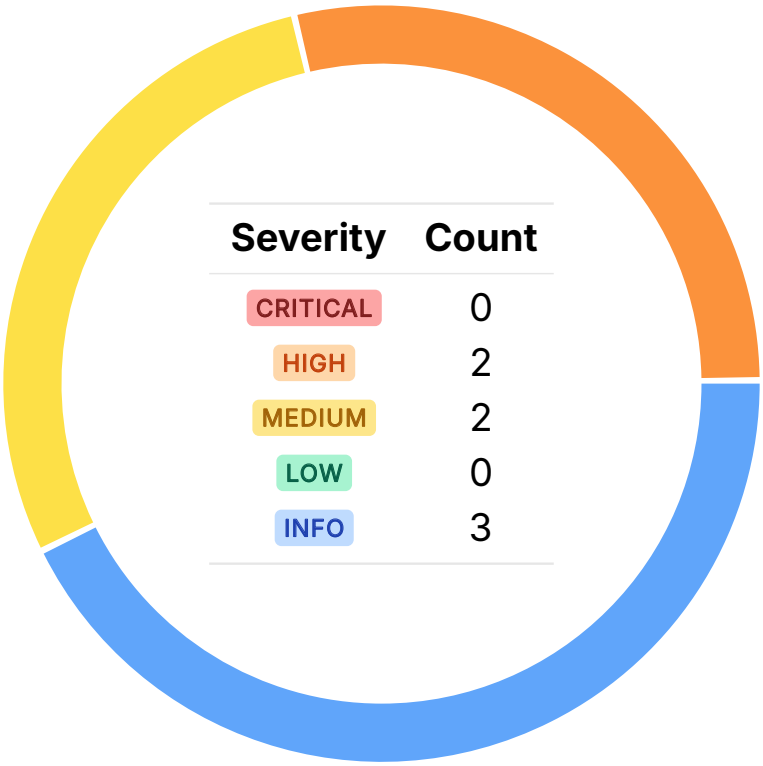
A brief description of the program is as follows:

Name	Description
marginfi-integration	It allows users to mint a receipt token that represents a deposit into the Marginfi lending protocol.

03 — Findings

Overall, we reported 7 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	0
HIGH	2
MEDIUM	2
LOW	0
INFO	3

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-MGI-ADV-00	HIGH	RESOLVED	The reward distribution in the Marginfi standard program is inconsistent because it is based on `SY` amounts, while the vault calculates emissions based on depreciating `YT` amounts. Also, the earned emissions are calculated based on an outdated `YT` balance, resulting in an inaccurate `lambo_fund` calculation.
OS-MGI-ADV-01	HIGH	RESOLVED	The `Balance` structure in the Marginfi-cpi library lacks necessary padding bytes after the `bank_pk` field, resulting in improper memory alignment and deserialization issues.
OS-MGI-ADV-02	MEDIUM	RESOLVED	The `withdraw_all` parameter in `withdraw_from_lending_account` is incorrectly encoded as a single byte, whereas the Marginfi program expects a two-byte `Option<bool>`.
OS-MGI-ADV-03	MEDIUM	RESOLVED	`RedeemSy` may inaccurately convert shares to the base asset due to a potential discrepancy between the `asset_share_value` and `mint_sy.supply`, especially if the asset share value is updated in the Marginfi withdraw instruction.

Inconsistent Reward Allocation HIGH OS-MGI-ADV-00

Description

There is a discrepancy between the utilization of `SY` and `YT` for reward distribution in the Marginfi-standard program and the vault program. In the Marginfi-standard program, the reward distribution is based on the amount of `SY` tokens held. This implies that the rewards are allocated proportionally to the `SY` tokens held by the users. However, the emissions calculated in the vault are based on the amount of `YT` tokens. The value of `YT` tokens depreciates as the `SY` exchange rate increases, resulting in a decrease in their value. Consequently, emissions are only allocated for the `sy_for_pt` amount, while rewards for `uncollected_sy` are not distributed to any user.

Furthermore, as the `SY` exchange rate increases, the amount of `YT` tokens decreases. This depreciation affects the emissions staging, resulting in a mismatch between the emissions staged and the actual rewards users should receive, depending on the timing of emissions staging relative to exchange rate changes.

>_ exponent_core/src/state/personal_yield_tracker.rs RUST

```rust
fn calc_earned_emissions(&self, current_index: Number, lp_amount_user: u64) -> u64 {
    let delta = current_index - self.last_seen_index;
    let earned = delta * Number::from_natural_u64(lp_amount_user);
    earned.floor_u64()
}
pub fn dec_staged(&mut self, amount: u64) {
    self.staged = self
        .staged
        .checked_sub(amount)
        .expect("insufficient staged balance");
}
```

Additionally, `YieldTokenTracker::calc_earned_emissions` (shown above) calculates the `lambo_fund` using the `current_index` and `sy_balance`, derived from the `yt_balance`. Since `merge` does not account for `YT` tokens, these tokens remain in the yield position, resulting in an imbalance where `YT` tokens continue to contribute to the `lambo_fund` calculation, even though they should have been merged. Consequently, the emissions calculated based on these tokens will be inaccurate, resulting in incorrect emissions.

Remediation

Align the reward calculation methods between the vault and the Marginfi standard program. Specifically, verify that both systems utilize a consistent metric for emission calculations. Also, ensure that `YT` tokens are correctly accounted for and converted to their `SY` equivalent before calculating emissions. `merge` should handle `YT` tokens properly to prevent them from remaining in the yield position.

Patch

Resolved in PR#606.

Missing Padding Bytes HIGH OS-MGI-ADV-01

Description

The issue pertains to the difference between the `Balance` structure in the Marginfi-CPI library within `marginfi_account` and the original implementation of the `Balance` structure in the Marginfi library. Specifically, the `Balance` structure in `marginfi_account` is missing padding bytes that are present in the original structure.

>_ marginfi_cpi/src/state/marginfi_account.rs RUST

```rust
pub struct Balance {
    pub active: bool,
    pub bank_pk: Pubkey,
    pub asset_shares: WrappedI80F48,
    pub liability_shares: WrappedI80F48,
    pub emissions_outstanding: WrappedI80F48,
    pub last_update: u64,
    pub _padding: [u64; 1],
}
```

Specifically, it does not include the 7-byte `_pad0` padding. This omission may result in improper deserialization of data. Consequently, incorrect values may be fetched from the structure fields such as `asset_shares` and `asset_share_value`.

Remediation

Update the Marginfi-CPI `Balance` structure to include the missing padding bytes (`_pad0`). This will ensure that both the original and copied structures have identical memory layouts.

Patch

Resolved in PR#569.

Incorrect Parameter Encoding MEDIUM OS-MGI-ADV-02

Description

In `marginfi_cpi::withdraw_from_lending_account`, there is an incorrect encoding of the `withdraw_all` parameter in the instruction data. `withdraw_from_lending_account` allows a user to withdraw funds from their lending account. It constructs and sends a cross-program invocation (CPI) to the Marginfi program with the appropriate parameters to perform this withdrawal.

>_ *marginfi_cpi/src/lib.rs* RUST

```rust
pub fn withdraw_from_lending_account<'info>(
    ctx: CpiContext<'_, '_, '_, 'info, LendingAccountWithdraw<'info>>,
    amount: u64,
    withdraw_all: bool,
) -> Result<()> {
    [...]
}
```

One of these parameters, `withdraw_all`, is a boolean flag indicating whether to withdraw the entire balance from the lending account. However, the `withdraw_all` field in the instruction is incorrectly assigned as a single byte (1), while the Marginfi program expects the `withdraw_all` parameter to be an `Option<bool>`, which is a two-byte representation. Thus, the Marginfi program expects an `Option<bool>`, but it receives a single byte.

Remediation

Encode the `withdraw_all` parameter as `Option<bool>`.

Patch

Resolved in PR#508.

Discrepancy in Conversion of Synthetic Yield Tokens MEDIUM OS-MGI-ADV-03

Description

In the `RedeemSy` instruction, there is a potential discrepancy in the conversion of synthetic yield (`SY`) tokens (shares) to their corresponding base asset amounts. This discrepancy arises due to the way the share-to-asset conversion rate may change after the `SY` tokens are burned but before the base assets are redeemed. When a user initiates the `RedeemSy` instruction, the first step is to burn the specified amount of `SY` tokens (`amount`). Between the time the `SY` tokens are burned and the base assets are redeemed through the `FakeRewards` program, the asset share value may be updated.

>_ fake_rewards_sy/src/instructions/redeem_sy.rs RUST

```rust
pub fn handler(ctx: Context<RedeemSy>, amount: u64) -> Result<()> {
    let bump = ctx.bumps.authority;

    // Burn SY tokens
    token_2022::burn(
        CpiContext::new(
            ctx.accounts.token_2022_program.to_account_info(),
            anchor_spl::token_2022::Burn {
                mint: ctx.accounts.mint_sy.to_account_info(),
                from: ctx.accounts.sy_account.to_account_info(),
                authority: ctx.accounts.owner.to_account_info(),
            },
        ),
        amount,
    )?;

    // Redeem shares from FakeRewards and transfer the base asset
    let seeds: [&[u8]; 2] = [crate::GLOBAL_AUTH_SEED, &[bump]];
    [...]
```

As a result, the calculation utilized to determine the number of base assets corresponding to the burned `SY` tokens (`shares`) may no longer be accurate, as the base amount to be redeemed is calculated via the asset share value at the time the `SY` tokens are burned. If this value changes before the redemption is complete, the resulting base amount will not accurately reflect the user's intended redemption value. Consequently, the invariant check that ensures that the total supply of `SY` tokens (`mint_sy.supply`) and the corresponding base assets remain in balance will fail.

Remediation

Ensure that the burning of `SY` tokens and the redemption of base assets occur atomically within a single transaction, minimizing the window during which the asset share value may change.

Patch

Resolved in PR#569.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-MGI-SUG-00	In the `AddEmission` instruction, any account may be designated as the `escrow_account`, without restrictions on it being an ATA.
OS-MGI-SUG-01	There are several instances where proper validation is not done, resulting in potential security issues.
OS-MGI-SUG-02	The codebase contains multiple cases of unutilized and redundant code that should be removed for better maintainability and clarity.

Non-deterministic Destination Account OS-MGI-SUG-00

Description

The issue in `LendingAccountWithdrawEmissions` and `AddEmission` instructions pertains to how emission accounts and destination accounts are managed and validated. In `LendingAccountWithdrawEmissions`, the associated token account (ATA) of `emission_mint` and `mfi_authority` is utilized for the `destination_account`. However, in `AddEmission`, any account may be designated as the `escrow_account` for a new emission. The current implementation does not enforce that the `escrow_account` must be an ATA of the `emission_mint` and a specific authority (such as `mfi_authority`), nor does it enforce that it should be a Program Derived Address (PDA) with a specific seed.

>_ *marginfi_standard/src/instructions/admin/add_emission.rs* RUST

```rust
pub fn validate(&self) -> Result<()> {
    self.admin_state
        .principles
        .marginfi_standard
        .is_admin(self.signer.key)?;
    Ok(())
}

pub fn add_emission(&mut self) {
    self.meta.emissions.push(Emission {
        mint: self.mint_emission.key(),
        escrow_account: self.token_emission.key(),
        index: Number::ZERO,
        last_seen_total_accrued_emissions: 0,
        total_claimed_emissions: 0,
        treasury_emission: 0,
        last_seen_index: Number::ZERO,
    })
}
```

As a result, the generation of the `destination_account` is not deterministic currently. Furthermore, there are no checks to prevent adding the same `emission_mint` multiple times. Thus, the mint key may be duplicated in `SyMeta.emissions`. This implies the same token mint may be added more than once, resulting in redundant or conflicting emission records.

Remediation

Ensure that when adding a new emission in `AddEmission`, the `escrow_account` is a PDA derived via a specific seed that includes both the `marginfi_bank` and the `emission_mint`. This will ensure that the `destination_account` is uniquely and deterministically tied to each emission mint. Additionally, before adding a new emission in `AddEmission`, check if `emission_mint` already exists in `SyMeta.emissions`.

Patch

This issue was acknowledged.

Missing Validation Logic OS-MGI-SUG-01

Description

1. `MintSy::invariant` is intended to ensure that the state of the program remains consistent after certain operations. However, it does not ensure that the current supply of `SY` tokens (`mint_sy.supply`) is below or equal to the maximum allowable supply (`max_sy_supply`). This omission may result in minting more `SY` tokens than the cap allows.

>_ marginfi_standard/src/instructions/mint_sy.rs RUST

```rust
fn invariant(&mut self) -> Result<()> {
    self.mint_sy.reload()?;
    invariant(
        &self.marginfi_account.load()?.lending_account,
        &self.marginfi_bank.key(),
        self.mint_sy.supply,
    )
}
```

2. There is an absence of explicit checks for token authority and mint consistency in all the instructions. While some validation is performed implicitly through token CPI (Cross-Program Invocation) calls, there are security and efficiency benefits to performing these checks explicitly before these calls. The absence of these checks may introduce vulnerabilities and make error handling less transparent. Furthermore, the authority accounts are not verified against `SyMeta` .
3. `MintSy` currently permits any pre-existing token account to be utilized as the `token_base_account_authority` . Since any token account is allowed, users must manually create a token account if they do not already have one, which may be cumbersome. Additionally, this introduces the need to manage and verify multiple accounts in the program logic, increasing complexity and the potential for errors.

Remediation

1. Include a check to ensure that the current supply of `SY` tokens does not exceed `max_sy_supply` .
2. Explicitly check if the token accounts and mints are consistent with what is expected, verifying that the token mint is correct and that the authority matches the expected authority for the operation. If any of these checks fail, return a custom error. Also, verify the authority accounts against `SyMeta` .
3. Utilize an Associated Token Account (ATA) with `init_if_needed` , which is more efficient and secure.

Patch

1. Issue #1 was resolved in PR#604.
2. Issue #2 was resolved in PR#563.
3. Issue #3 was acknowledged.

Code Redundancy

OS-MGI-SUG-02

Description

1. In the calculation of `SyMeta::LEN_STATIC`, the size allocations for `treasury_sy` and `last_seen_share_index` are unnecessary and may be removed.
2. In the `Number` implementation, `is_positive` and `is_negative` appear redundant since they do not support negative values.

>_ solana/libraries/precise_number/src/lib.rs RUST

```rust
impl Number {
    [...]
    pub fn is_positive(&self) -> bool {
        self.gt(&Number::ZERO)
    }

    pub fn is_negative(&self) -> bool {
        !self.is_positive()
    }
    [...]
}
```

Remediation

Remove the redundant and unutilized code instances highlighted above.

Patch

1. This issue was already fixed by the Exponent team.
2. Issue #2 was resolved in PR#562.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.