OFFSIDE LABS

Exponent

Marginfi Standard

Smart Contract Security Assessment

September 2024

Prepared for:

Exponent Finance

Prepared by:

Offside Labs

Ripples Wen

Siji Feng

Contents

1 About Offside Labs

Offside Labs is a leading security research team, composed of top talented hackers from both academia and industry.

We possess a wide range of expertise in modern software systems, including, but not limited to, *browsers*, *operating systems*, *IoT devices*, and *hypervisors*. We are also at the forefront of innovative areas like *cryptocurrencies* and *blockchain technologies*. Among our notable accomplishments are remote jailbreaks of devices such as the **iPhone** and **PlayStation 4**, and addressing critical vulnerabilities in the **Tron Network**.

Our team actively engages with and contributes to the security community. Having won and also co-organized *DEFCON CTF*, the most famous CTF competition in the Web2 era, we also triumphed in the **Paradigm CTF 2023** within the Web3 space. In addition, our efforts in responsibly disclosing numerous vulnerabilities to leading tech companies, such as *Apple*, *Google*, and *Microsoft*, have protected digital assets valued at over **$300 million**.

In the transition towards Web3, Offside Labs has achieved remarkable success. We have earned over **$9 million** in bug bounties, and **three** of our innovative techniques were recognized among the **top 10 blockchain hacking techniques of 2022** by the Web3 security community.


https://offside.io/


https://github.com/offsidelabs


https://twitter.com/offside_labs

2 Executive Summary

Introduction

Offside Labs completed a security audit of *Exponent* smart contracts, starting on *September 23rd, 2024*, and concluding on *September 30th, 2024*.

Project Overview

This program enables users to mint standard yield tokens that represent deposits in the Marginfi lending protocol, with each token corresponding to one asset share in the Marginfi pool. These tokens enhance DeFi composability by allowing interaction with Marginfi deposits without needing Cross-Program Invocations. The program provides functions to access important asset information, such as:

1. Share-to-underlying asset ratio.
2. Accrued emission indexes for the depositor account.
3. Current state of staked emission positions.

Audit Scope

The assessment scope contains mainly the smart contracts of the *Marginfi Standard* program for the *Exponent* project.

The audit is based on the following specific branches and commit hashes of the codebase repositories:

- *Marginfi Standard*
  - Code Base: https://github.com/exponent-finance/exponent-core
  - Commit Hash: ae790305659aca41dee8ec3346045377be73971a

We listed the files we have audited below:

- *Marginfi Standard*
  - solana/programs/marginfi_standard
  - solana/libraries/marginfi_cpi
  - solana/libraries/i80_converter

Findings

The security audit revealed:

- 0 critical issue
- 0 high issue
- 1 medium issues
- 3 low issues
- 2 informational issues

Further details, including the nature of these issues and recommendations for their remediation, are detailed in the subsequent sections of this report.

3 Summary of Findings

ID	Title	Severity	Status
01	Missing refresh_emissions in mint_sy/redeem_sy on Emission Token Index and Distribution	Medium	Fixed
02	Incorrect Assertion in marginfi_-standard::redeem_sy Instruction	Low	Fixed
03	Missing Remaining Accounts in marginfi_-standard::mint_sy Instruction	Low	Fixed
04	Potential Outdated Exchange Rate Retrieval Due to Missing Bank Refresh in get_state Instruction	Low	Acknowledged
05	mint_sy Account Not Reloaded After cpi_mint_sy	Informational	Fixed
06	Inconsistent Payer for Position Reallocation Fees	Informational	Acknowledged

4 Key Findings and Recommendations

4.1 Missing refresh_emissions in mint_sy/redeem_sy on Emission Token Index and Distribution

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `refresh_emissions` function, the increase in the emission token index is calculated by dividing the increase in token amount by the `sy_supply` .

```
200 fn increase_from_token_credit(&mut self, sy_supply: u64, token_amount:
    ↪  u64) {
201     let index_delta = Number::from_ratio(token_amount.into(),
        ↪  sy_supply.into());
202
203     self.index += index_delta;
204     self.last_seen_total_accrued_emissions += token_amount as u128;
205 }
```

solana/programs/marginfi_standard/src/state/sy_meta.rs#L200-L205

Both the `mint_sy` and `redeem_sy` functions update the `sy_supply` without invoking `refresh_emissions` . As a result, the emissions accrued prior to calling `mint_sy` will be divided by the updated `sy_supply` . Since the newly minted tokens have not yet been deposited, the unallocated emissions corresponding to these tokens will be directed to the treasury account.

Recommendation

Invoke `refresh_emissions` within both the `mint_sy` and `redeem_sy` instructions before updating the `sy_supply` .

Mitigation Review Log

Exponent Team: Fixed in relevant code implementation

Offside Labs: `Fixed`

4.2 Incorrect Assertion in marginfi_standard::redeem_sy Instruction

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Redundant Check

Description

In the `redeem_sy` instruction, we require that the second remaining account match the predefined oracle key of the bank. This assertion is too strict and should not be enforced, as it can break the program in certain cases.

```
179     let bank = ctx.accounts.marginfi_bank.load()?;
180     let bank_pyth_feed = bank.config.oracle_keys.get(0).unwrap();
181     // compare bank pyth feed with remaining accounts at index 1
182     let bank_pyth_feed_2 = ctx.remaining_accounts.get(1).unwrap();
183     msg!("bank_pyth_feed: {}", bank_pyth_feed);
184
185     assert_eq!(bank_pyth_feed, &bank_pyth_feed_2.key(), "does not
        ↪  match");
```

solana/programs/marginfi_standard/src/instructions/redeem_sy.rs#L179-L184

For instance, if the `token_program` account in `margifin` 's `LendingAccountWithdraw` instruction (which corresponds to the `token_base_program` account in `margifin_standard` 's `RedeemSy` instruction) is set to token2022, it will attempt to interpret the first remaining account as the bank's mint and then advance the pointer to the `remaining_accounts` . In this scenario, the `pyth oracle` is no longer the second account in the original `remaining_accounts` .

```rust
/// Checks if first account is a mint account. If so, updates
    ↪ remaining_account -> &remaining_account[1..]
///
/// Ok(None) if Tokenkeg
pub fn maybe_take_bank_mint<'info>(
    remaining_accounts: &mut &'info [AccountInfo<'info>],
    bank: &Bank,
    token_program: &Pubkey,
) -> MarginfiResult<Option<InterfaceAccount<'info, Mint>>> {
    match *token_program {
        anchor_spl::token::ID => Ok(None),
        anchor_spl::token_2022::ID => {
            let (maybe_mint, remaining) = remaining_accounts
                .split_first()
                .ok_or(MarginfiError::T22MintRequired)?;
            *remaining_accounts = remaining;

            if bank.mint != *maybe_mint.key {
                return err!(MarginfiError::T22MintRequired);
            }
```

programs/marginfi/src/utils.rs#L114-L132

Furthermore, the newly introduced oracle type also breaks the assumptions about oracle accounts.

Recommendation

Do not implement a mandatory check in `marginfi_standard`. The constraints will be validated later within `marginfi`, making it unnecessary to assert them in `marginfi_standard`.

Mitigation Review Log

Exponent Team: change already addressed. Check not present anymore.

4.3 Missing Remaining Accounts in marginfi_standard::mint_sy Instruction

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Compatibility Issue

Description

In the `mint_sy` instruction, the deposit into `marginfi` does not utilize the `remaining_accounts`. However, the deposit instruction in `marginfi` also manages token2022 and attempts to parse the remaining accounts using `maybe_take_bank_mint`. As a result, the current implementation breaks due to this inconsistency.

```
24 pub fn lending_account_deposit<'info>(
25     mut ctx: Context<'_, '_, 'info, 'info, LendingAccountDeposit<'info>>,
26     amount: u64,
27 ) -> MarginfiResult {
28     let LendingAccountDeposit {
29         marginfi_account: marginfi_account_loader,
30         signer,
31         signer_token_account,
32         bank_liquidity_vault,
33         token_program,
34         bank: bank_loader,
35         ..
36     } = ctx.accounts;
37     let clock = Clock::get()?;
38     let maybe_bank_mint = utils::maybe_take_bank_mint(
39         &mut ctx.remaining_accounts,
40         &*bank_loader.load()?,
41         token_program.key,
42     )?;
```

programs/marginfi/src/instructions/marginfi_account/deposit.rs#L24-L42

Recommendation

Ensure that the `mint_sy` instruction properly passes the `remaining_accounts` to the `LendingAccountDeposit` instruction in `marginfi` to align with its expected behavior.

Mitigation Review Log

Exponent Team: Fixed in relevant code implementation

Offside Labs: `Fixed`

4.4 Potential Outdated Exchange Rate Retrieval Due to Missing Bank Refresh in get_state Instruction

Severity: Low	Status: Acknowledged
Target: Smart Contract	Category: Inconsistent State

Description

In the `get_state` instruction of the `marginfi_standard` program, the `exchange_rate` is retrieved from the `asset_share_value` field of the `marginfi_bank` account. However, the bank is not refreshed prior to fetching this value, which means the `asset_share_value` may be outdated or stale at the time of retrieval.

Recommendation

Add a call to `LendingPoolAccrueBankInterest` within the `get_state` instruction. This will refresh the bank account before fetching the `asset_share_value`, ensuring that the retrieved exchange rate accurately reflects the current state of the bank.

4.5 Informational and Undetermined Issues

mint_sy Account Not Reloaded After cpi_mint_sy

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Account Synchronization

In the `marginfi_standard::mint_sy` instruction, `mint_sy.supply` is checked without reloading the mint account after calling `cpi_mint_sy`.

```
190     ctx.accounts.cpi_mint_sy(new_shares)?;
191     assert!(ctx.accounts.mint_sy.supply <=
          ↪ ctx.accounts.meta.max_sy_supply);
192     ctx.accounts.invariant()?;
```

solana/programs/marginfi_standard/src/instructions/mint_sy.rs#L190-L194

Inconsistent Payer for Position Reallocation Fees

Severity: Informational	Status: Acknowledged
Target: Smart Contract	Category: Rent Inconsistency

We've observed that the reallocation of a Position can be triggered by multiple instructions, but the payer for the reallocation fees is inconsistent across these instructions: 1. The initialization fee is paid by a `fee_payer`, who is not necessarily the `owner`. 1. The reallocation fee in the deposit instruction is paid by the `depositor`, not the `position.owner`. 1. The reallocation fee in the withdraw instruction is paid by the `owner`. 1. The reallocation fee in the get_position instruction is paid by the `authority_mfi_account`, which is a PDA account. Note: Since there is only one emission in `marginfi_standard`, this does not seem to be a significant problem. The `fee_payer` selection is (almost) correctly handled in `kamino_lend_standard`. However, both programs are using a PDA account as the fee payer, which does not make much sense.

5 Disclaimer

This audit report is provided for informational purposes only and is not intended to be used as investment advice. While we strive to thoroughly review and analyze the smart contracts in question, we must clarify that our services do not encompass an exhaustive security examination. Our audit aims to identify potential security vulnerabilities to the best of our ability, but it does not serve as a guarantee that the smart contracts are completely free from security risks.

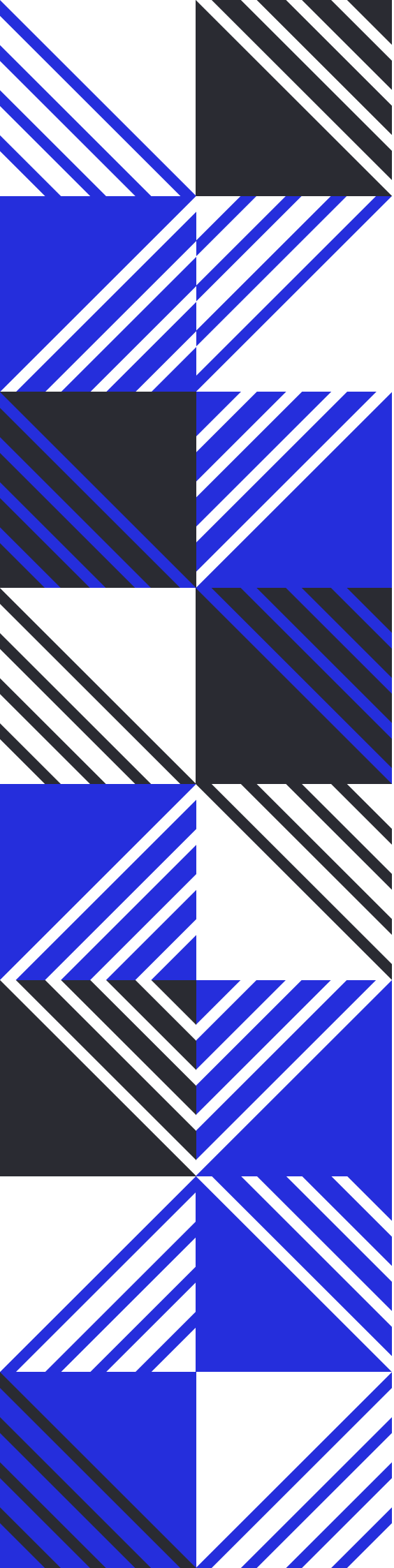
We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster the security of their smart contracts.

It is important to note that the scope of our audit is limited to the areas outlined within our engagement and does not include every possible risk or vulnerability. Continuous security practices, including regular audits and monitoring, are essential for maintaining the security of smart contracts over time.

Please note: we are not liable for any security issues stemming from developer errors or misconfigurations at the time of contract deployment; we do not assume responsibility for any centralized governance risks within the project; we are not accountable for any impact on the project's security or availability due to significant damage to the underlying blockchain infrastructure.

By using this report, the client acknowledges the inherent limitations of the audit process and agrees that our firm shall not be held liable for any incidents that may occur subsequent to our engagement.

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



OFFSIDE LABS

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs