Sanctum

Audit

Presented by:

OtterSec contact@osec.io

Thibault Marboud thibault@osec.io

Tamta Topuria tamta@osec.io

Robert Chen r@osec.io

Contents

01 Executive Summary **2**
Overview . . . 2
Key Findings . . . 2

02 Scope **3**

03 Findings **4**

04 Vulnerabilities **5**
OS-SCT-ADV-00 [med] | Rounding Error During Swaps . . . 6
OS-SCT-ADV-01 [med] | Missing Rebalance Checks . . . 7
OS-SCT-ADV-02 [med] | Potential Pool Unbalancing . . . 8
OS-SCT-ADV-03 [low] | Denial Of Service . . . 9
OS-SCT-ADV-04 [low] | Protocol Fee Modification Via Frontrunning . . . 10
OS-SCT-ADV-05 [low] | Account Type Validation . . . 11

05 General Findings **12**
OS-SCT-SUG-00 | Missing Minimum Protocol Fee Check . . . 13
OS-SCT-SUG-01 | Lack Of Account Verification . . . 14
OS-SCT-SUG-02 | Missing Checks . . . 15

Appendices

A Vulnerability Rating Scale **16**

B Procedure **17**

01 | Executive Summary

Overview

Igneous Labs engaged OtterSec to assess the S program. This assessment was conducted between January 2nd and January 15th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 9 findings throughout this audit engagement.

In particular, we identified several vulnerabilities, including the lack of checks in the rebalance functionality for confirmation of the destination liquidity pool's mint account (OS-SCT-ADV-01) and another issue concerning a potential denial of service scenario due to creating an excessive amount of associated token accounts (OS-SCT-ADV-03). We further highlighted the ability of the admin to front-run the liquidity provider deposits or withdrawals to manipulate the protocol fees (OS-SCT-ADV-04).

We provided recommendations regarding the risk of users executing swaps for free with minimal amounts (OS-SCT-SUG-00) and suggested implementing account validation checks in certain areas of the code base (OS-SCT-SUG-01). Additionally, we emphasized the importance of incorporating missing validations (OS-SCT-SUG-02) to ensure adherence to best practices.

02 | Scope

The source code was delivered to us in a Git repository at github.com/igneous-labs/S/tree/ottersec-231220. This audit was performed against commit dd0768d.

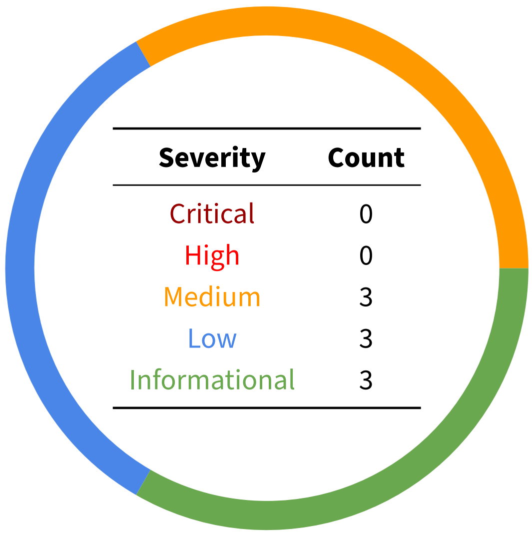
A brief description of the programs is as follows:

Name	Description
S	The Sanctum program collaboratively facilitates a multi-liquid staking token automated market maker, embodying the ”Curve of LSTs” concept by managing numerous liquid staking tokens. This setup enables capital-efficient swaps between liquid staking tokens, with all accounting and calculations conducted in SOL terms.

03 | Findings

Overall, we reported 9 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	0
Medium	3
Low	3
Informational	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-SCT-ADV-00	Medium	Resolved	Utilizing `U64RatioFloor:reverse` rounds up the `sol_to_lst` cross-program invocation, resulting in a rounding error, allowing users to receive one extra token than intended.
OS-SCT-ADV-01	Medium	Resolved	`start_rebalance` lacks checks to confirm the destination liquidity pool's mint account (`dst_lst_mint`) and the conditions stipulating re-balancing the reserves.
OS-SCT-ADV-02	Medium	Resolved	Re-balancing based on a fixed amount in the presence of network latency and arbitrage activities may result in unbalanced pools due to state changes.
OS-SCT-ADV-03	Low	Resolved	`add_lst` is vulnerable to a denial of service attack through associated token account creation.
OS-SCT-ADV-04	Low	Resolved	Deposits and withdrawals initiated by liquidity providers lack safeguards against potential front-running by an admin, allowing the fee to become 100% via `set_protocol_fee`.
OS-SCT-ADV-05	Low	Resolved	`add_lst` and `remove_lst` lack validation for the account type being added or removed.

OS-SCT-ADV-00 [med]| Rounding Error During Swaps

Description

Within swap_exact_in during the cross-program invocation call to sol_to_lst, U64RatioFloor:reverse rounds up the resulting value. This rounding behavior may allow users to receive one more lamport than they should. In the context of a token swap, this rounding behavior may result in a situation where the resulting dst_lst amount is slightly higher than it would be with rounding down. It should be noted that this issue becomes exploitable only if the flat-fee program is configured with zero fees since, due to the absence of fees, the exact amount of tokens received becomes critical for users looking to maximize their gains.

s-src/processor/swap_exact_in.rs RUST

```rust
pub fn process_swap_exact_in(accounts: &[AccountInfo], args: SwapExactInIxArgs) ->
    ↪  ProgramResult {
    [...]
    let out_sol_value = pricing_cpi.invoke_price_exact_in(PricingProgramIxArgs {
        amount,
        sol_value: in_sol_value,
    })?;
    let dst_lst_out = dst_lst_cpi.invoke_sol_to_lst(out_sol_value)?.min;
    [...]
}
```

Remediation

Round down the result of U64RatioFloor:reverse in the instructions.

Patch

Fixed in 87832e3.

OS-SCT-ADV-01 [med]| Missing Rebalance Checks

Description

`start_rebalance` initiates a re-balance operation between two liquidity pools. `end_rebalance` is expected to conclude the re-balance operation. However, `start_rebalance` fails to perform a thorough check on the destination liquidity pool's mint account within `process_start_rebalance`, specifically the `dst_lst_mint` account of `end_rebalance`. The absence of this verification of the destination mint account renders the system susceptible to fund loss if the address in `dst_lst_mint` is inadvertently set to an incorrect value.

s-src/processor/start_rebalance.rs RUST

```rust
pub fn process_start_rebalance(
    accounts: &[AccountInfo],
    args: StartRebalanceIxArgs,
) -> ProgramResult {
    let (
        accounts,
        SrcDstLstSolValueCalculatorCpis {
            src_lst: src_lst_cpi,
            dst_lst: dst_lst_cpi,
        },
        SrcDstLstIndexes {
            src_lst_index,
            dst_lst_index,
        },
    ) = verify_start_rebalance(accounts, &args)?;
    [...]
}
```

Remediation

Implement checks in `process_start_rebalance` to ensure the destination mint account aligns with the expected properties.

Patch

Fixed in 2dec5bb.

OS-SCT-ADV-02 [med]| Potential Pool Unbalancing

Description

pool adjusts the pool's composition based on a specified amount of tokens. However, if the state of the pool changes between the transaction submission and its execution due to factors such as arbitrage bots exploiting price differences, it may imbalance the pool. Arbitrage activities may result in rapid changes in token prices. If a re-balancing transaction is initiated but not executed immediately, the pool may re-balance based on outdated prices, unbalancing it.

Remediation

Ensure that rebalancing is necessary to prevent alterations to the reserve balances. Additionally, computation for the rebalancing amount should be be on-chain, providing increased accuracy and adaptability to changing market conditions.

Patch

Fixed in 0a258ce.

OS-SCT-ADV-03 [low] | Denial Of Service

Description

add_lst is vulnerable to a denial of service attack. process_add_lst creates two associated token accounts before executing the main logic of the add_lst. The program creates associated token accounts for pool_reserves and protocol_fee_accumulator associated with the lst_mint. It is possible to invoke create_ata_invoke repeatedly to create numerous associated token accounts for the same st_mint before executing the actual add_lst. This accumulates unnecessary associated token accounts, consuming additional storage space on the blockchain.

s-src/processor/set_protocol_fee.rs RUST

```rust
pub fn process_add_lst(accounts: &[AccountInfo]) -> ProgramResult {
    [...]
    // Attempting to create the ATAs verifies that the mint is not a duplicate
    create_ata_invoke(CreateAtaAccounts {
        ata_to_create: accounts.pool_reserves,
        wallet: accounts.pool_state,

        payer: accounts.payer,
        mint: accounts.lst_mint,
        system_program: accounts.system_program,
        token_program: accounts.lst_token_program,
    })?;

    create_ata_invoke(CreateAtaAccounts {
        ata_to_create: accounts.protocol_fee_accumulator,
        wallet: accounts.protocol_fee_accumulator_auth,

        payer: accounts.payer,
        mint: accounts.lst_mint,
        system_program: accounts.system_program,
        token_program: accounts.lst_token_program,
    })?;
    [...]
```

Remediation

The associated token account creation should ideally be part of the atomic operation performed by add_lst. Additionally, verify that the lst_mint has not already been added before proceeding with add_lst.

Patch

Fixed in 6cffae8.

OS-SCT-ADV-04 [low] | Protocol Fee Modification Via Frontrunning

Description

The current protocol design enables the admin to exploit `set_protocol_fee` by invoking it to front-run liquidity provider deposits or withdrawals to manipulate the protocol fees. This may result in a scenario in which the admin adjusts the protocol fees to an extremely high value (e.g., 100%).

s-controller/src/processor/set_protocol_fee.rs RUST

```rust
pub fn process_set_protocol_fee(
    accounts: &[AccountInfo],
    args: SetProtocolFeeIxArgs,
) -> ProgramResult {
    let (
        accounts,
        SetProtocolFeeIxArgs {
            new_trading_protocol_fee_bps,
            new_lp_protocol_fee_bps,
        },
    ) = verify_set_protocol_fee(accounts, args)?;
    [...]
    Ok(())
}
```

Consequently, users depositing into the liquidity pool may receive fewer liquidity pool tokens than expected, impacting their share of the pool and potential rewards, and users withdrawing from the pool may receive fewer underlying assets than expected due to the higher fee, reducing the value of their holdings.

Remediation

As a preventive measure, introduce a new parameter (example: `min_lp_token_out`), ensuring that users receive a minimum amount of liquidity pool tokens by calculating the tokens to distribute based on the new protocol fees and ensuring that the output satisfies the specified minimum.

Patch

Fixed in 4f71ef6.

OS-SCT-ADV-05 [low] | Account Type Validation

Description

add_lst in flat-fee and s-controller are expected to interact with a token program, which is crucial for handling associated token accounts. However, the instruction does not check whether the token program is either spl-2022 or spl-token. This lack of validation may yield unintended consequences when utilizing an unauthorized or incompatible token program.

Similarly, within remove_lst, while closing an account, it fails to ensure whether the account removed is a free account. This may inadvertently delete the program's main state by providing an arbitrary account as the fee_acc argument, and the function would proceed to close that account without verifying its type, affecting the overall functionality of the protocol.

Remediation

Add a validation step to check that the token program associated with the provided mint account is either spl-2022 or spl-token before proceeding with add_lst. Additionally, ensure the account passed to remove_lst is a fee account by d-serializing the account data.

Patch

Fixed in 83c0a5b and f0ee280.

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-SCT-SUG-00	`swap_exact_in` and `swap_exact_out` fails to ensure `to_protocol_fees_lst_amount` is non-zero, enabling users to execute swaps for free with minimal amounts.
OS-SCT-SUG-01	Suggestions regarding implementation of account validation checks.
OS-SCT-SUG-02	Recommendations regarding incorporating absent validations.

OS-SCT-SUG-00 | Missing Minimum Protocol Fee Check

Description

`swap_exact_in` and `swap_exact_out` fails to explicitly check whether `to_protocol_fees_lst_amount` is zero after calculating the protocol fees. This omission allows users to effectively swap for free with small amounts, as the protocol fees may round down to zero. If a user initiates `swap_exact_in` with a small amount, the calculation of protocol fees (`to_protocol_fees_lst_amount`) is performed based on the small amount causing protocol fees to be negligible, resulting in zero after truncation, allowing users to exploit the system by swapping small amounts without paying the intended protocol fees.

Remediation

Include a check after calculating `to_protocol_fees_lst_amount` to ensure it is not zero.

OS-SCT-SUG-01 | Lack Of Account Verification

Description

1. add_lst and set_sol_value_calculator both require sol_value_calculator as an account parameter. However, they lack a verification mechanism to ensure that the provided sol_value_calculator is a valid Solana program account.
2. In process_set_pricing_program, the pricing program key is updated in the pool state. However, there is no explicit check to ensure that the provided new_pricing_program account is indeed a program account.

s-src/processor/set_pricing_program.rs RUST

```rust
pub fn process_set_pricing_program(accounts: &[AccountInfo]) -> ProgramResult
    ↪ {
    let SetPricingProgramAccounts {
        admin: _,
        new_pricing_program,
        pool_state,
    } = verify_set_pricing_program(accounts)?;
    let mut pool_state_bytes = pool_state.try_borrow_mut_data()?;
    let pool_state = try_pool_state_mut(&mut pool_state_bytes)?;
    pool_state.pricing_program = *new_pricing_program.key;
    Ok(())
}
```

3. It may be beneficial to explicitly validate the discriminant and account types of the marinade and lido state accounts in verify_lst_sol_common. This ensures that the function passes the correct accounts to the program.

Remediation

1. Introduce a validation step by comparing the account's program ID with the expected program ID to enhance the safety of these operations.
2. Include a check to verify that new_pricing_program is a valid program account before proceeding with the update in the pool state.
3. Ensure to validate the discriminant and account types of the marinade and lido state accounts.

OS-SCT-SUG-02 | Missing Checks

Description

1. Within set_lp_withdrawal_fee, there is a lack of validation for the lower bound of the lp_withdrawal_fee_bps parameter in process_set_lp_withdrawal_fee, aimed at ensuring that fees remain within practical and expected ranges. It is advisable to introduce a check at the beginning of process_set_lp_withdrawal_fee to guarantee that lp_withdrawal_fee_bps is greater than or equal to a defined lower bound.
2. initialize employs spl_token_2022::instruction::AuthorityType to establish authorities for the liquidity pool token mint. However, it is more appropriate to utilize spl_token::instruction::AuthorityType for the originally imported version of the SPL token program.

   initialise.rs RUST

   ```rust
   use spl_token::{native_mint, state::Mint};
   use spl_token_2022::instruction::AuthorityType;

   pub fn process_initialize(accounts: &[AccountInfo]) -> ProgramResult {
       [...]
       set_authority_invoke(
           set_authority_accounts,
           SetAuthorityArgs {
               authority_type: AuthorityType::MintTokens,
               new_authority: Some(*accounts.pool_state.key),
           },
       )?;
       set_authority_invoke(
           set_authority_accounts,
           SetAuthorityArgs {
               authority_type: AuthorityType::FreezeAccount,
               new_authority: Some(*accounts.pool_state.key),
           },
       )
   }
   ```

3. In swap_exact_in and swap_exact_out, it is crucial to verify that the source token (src_lst) differs from the destination token (dst_lst). Examine this fundamental invariant before proceeding with the swap operation.

Remediation

Ensure to incorporate the checks mentioned above.

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 section.

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.
Informational	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 the other 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.