Sec3™

Security Assessment Report

Sanctum S

February 8, 2024

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Sanctum S smart contracts.

The artifact of the audit was the source code of the following programs, excluding tests, in a private repository.

The initial audit focused on the following versions and revealed 16 issues or questions.

#	program	type	commit
P1	S Controller	Solana	`dd0768d496fdba80a019ccce86d187fbc2355223`
P2	Pricing Program	Solana	`dd0768d496fdba80a019ccce86d187fbc2355223`
P3	SOL Value Calculator	Solana	`dd0768d496fdba80a019ccce86d187fbc2355223`

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

Table of Contents

Result Overview

Issue	Impact	Status
S CONTROLLER		
[P1-L-01] Missing PoolState validation when removing LST	Low	Resolved
[P1-I-01] Missing equivalence check for incoming and outgoing LSTs	Info	Resolved
[P1-I-02] Inconsistent rent receiver related checks	Info	Resolved
[P1-I-03] Arbitrage opportunities	Info	Acknowledged
[P1-I-04] Create ATAs only if needed when adding LST	Info	Resolved
[P1-I-05] Incomplete dst_lst check during rebalancing	Info	Resolved
[P1-I-06] Incomplete lst_mint check when adding LST	Info	Resolved
[P1-I-07] Missing lst_mint check when withdrawing protocol fee	Info	Acknowledged
[P1-I-08] Incorrect in_sol_value calculation in a corner case	Info	Resolved
[P1-I-09] Missing redundancy check when adding new disable pool authority	Info	Resolved
[P1-I-10] Missing healthy check during rebalancing	Info	Resolved
[P1-I-11] Documentation and IDL nitpicks	Info	Resolved
PRICING PROGRAM		
[P2-L-01] Arbitrary account closure when removing LST	Low	Resolved
[P2-I-01] Better signed fee bps check	Info	Acknowledged
[P2-I-02] Missing lst_mint check when adding LST	Info	Resolved
SOL VALUE CALCULATOR		
[P3-I-01] Round up withdraw fee for SPL stake pool	Info	Resolved

Findings in Detail

S CONTROLLER

[P1-L-01] Missing PoolState validation when removing LST

In the "`RemoveLst`" instruction, as per the documentation, the initial step involves verifying whether the pool is in a rebalancing state or disabled. However, the implemented code lacks the invocation of the "`verify_not_rebalancing_and_not_disabled`" function, resulting in the omission of pertinent checks.

It is noteworthy that this instruction can only be invoked by the pool administrator, and the requisite checks are conducted in the preceding instruction, "`DisableLstInput`", mitigating potential risks to a considerable extent.

Recommendations

Consider adding the "`verify_not_rebalancing_and_not_disabled`" check.

Resolution

This issue has been resolved by commit 47b0fc1c1d238db6a931ed19ab5f27c3bef91cb3.

S CONTROLLER

[P1-I-01] Missing equivalence check for incoming and outgoing LSTs

Within the instructions "`swap_exact_in`" and "`swap_exact_out`", both employed for swapping operations, there is presently an absence of verification regarding the equivalence of the incoming and outgoing LST tokens.

In the context of swapping, confirming the identity of these tokens is a customary safeguard to protect users, and in certain protocols, it may prevent complications in the accounting process.

It is recommended to incorporate the check to comply with the common practice.

Resolution

This issue has been resolved by commit 4c6775fc06c07fa54cf6b994ef2764ac6a481d2e.

S CONTROLLER
[P1-I-02] Inconsistent rent receiver related checks

Within the project, three instructions, apart from "`end_rebalance`", necessitate users to provide a rent receiver for the restitution of rent upon the closure of an account. However, there exists inconsistency in the behavior across these three instances.

- "`remove_lst`" in S Controller. The documentation specifies that "`refund_rent_to`" should be mutable only. The implementation aligns with this specification.
- "`remove_disable_pool_authority`" in S Controller. Account "`refund_rent_to`" is mutable only according to the documentation. However, the implementation additionally requires it to be a signer.
- "`remove_lst`" in Pricing Program. The documentation specifies that "`refund_rent_to`" should be a signer only. However, the implementation additionally requires it to be mutable.

Recommendations

It is recommended to standardize these three instances by specifying that "`refund_rent_to`" should be mutable and no signer requirement is necessary.

Resolution

This issue has been resolved by commits 301c92c86048b8300d73a4765ae10bbd8da51398 and 4c6775fc06c07fa54cf6b994ef2764ac6a481d2e.

S CONTROLLER

[P1-I-03] Arbitrage opportunities

In the “`add_liquidity`” instruction, only the value of “`lst_index`” asset is updated. As a result, the calculated “`total_sol_value`” may deviate from the actual value, leading to an over- or under-issuance of “`lp_tokens`”. The same issue also exists in “`remove_liquidity`”.

Consider the following scenario: the pool contains n types of assets, and their values increase on epoch transitions. The amount in the pool for LST_i is x_i. When transitioning from epoch n to epoch $n+1$, the unit price of the assets changes, denoted as δ_i.

If, at this point, LST A in the pool has not been used in this epoch (whether through swap or add/remove liquidity), then “`sync_sol_value_unchecked`” has not been called to update its real value in “`pool_state.total_sol_value`”. At this point, arbitrageurs can use “`add_liquidity`” to add another LST B with a total value v_{arb}. The proportion of “`lp_tokens`” obtained will be $\frac{v_{arb}}{v_{arb}+v_{pool}}$. Subsequently, the arbitrageur, in the same transaction, adds a negligible amount of A to the pool using “`add_liquidity`”. Since “`sync_sol_value_unchecked(A)`” is called, the pool's value will return to normal, increasing by $\delta_A \cdot x_A$. The arbitrageur then uses “`remove_liquidity`” to burn these “`lp_tokens`” and take back to A, profiting $\frac{v_{arb}}{v_{arb}+v_{pool}} \cdot (v_{arb} + v_{pool} + \delta_A \cdot x_A) - v_{arb} = \frac{v_{arb} \cdot \delta_A \cdot x_A}{v_{arb}+v_{pool}}$.

If a particular asset has not been used by anyone for multiple epochs (whether through swap or add/remove liquidity), δ_A will be even larger, giving the arbitrageurs a greater profit potential, even taking LP withdrawal fees into consideration.

To avoid such scenarios, consider to update the total value of the assets in the S Controller when an LST price change is observed or make sure to update the total value once every epoch.

Resolution

The team acknowledged this issue and plans to address it by updating the total value in an off-chain program.

S CONTROLLER

[P1-I-04] Create ATAs only if needed when adding LST

```
/* programs/s-controller/src/processor/add_lst.rs */
// 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,
})?;
```

In the "`add_lst`" instruction, the program assesses whether the respective LST has been added previously by creating the corresponding ATA and determining its existence based on the success or failure of this operation. However, given that ATAs can be generated by any user, malicious actors may intentionally create the relevant ATA in advance to induce a failure in the "`add_lst`" instruction.

Recommendations

It is advisable to adopt a more robust approach for ascertaining whether the LST has been previously added, and creating the associated ATA only when the corresponding ATA does not already exist.

Resolution

This issue has been resolved by commit 4c6775fc06c07fa54cf6b994ef2764ac6a481d2e.

S CONTROLLER

[P1-I-05] Incomplete dst_lst check during rebalancing

"start_rebalance" and "end_rebalance" should be called in pairs.

In "start_rebalance", the presence of a subsequent "end_rebalance" operation in the same transaction is verified through "verify_has_succeeding_end_rebalance_ix". However, this check only ensures the existence of "end_rebalance" without verifying the consistency of "dst_lst_mint".

```
/* programs/s-controller/src/processor/start_rebalance.rs */
pub fn process_start_rebalance(
    accounts: &[AccountInfo],
    args: StartRebalanceIxArgs,
) -> ProgramResult {
...
    let mut rebalance_record_data = accounts.rebalance_record.try_borrow_mut_data()?;
    let rebalance_record = try_rebalance_record_mut(&mut rebalance_record_data)?;
    rebalance_record.dst_lst_index = args.dst_lst_index;
    rebalance_record.old_total_sol_value = old_total_sol_value;

/* libs/s-controller-lib/src/accounts_resolvers/end_rebalance.rs */
pub fn resolve(self) -> Result<(EndRebalanceKeys, usize), SControllerError> {
...
        let rebalance_record_acc_data = self.rebalance_record.data();
        let RebalanceRecord { dst_lst_index, .. } =
            try_rebalance_record(&rebalance_record_acc_data)?;
        let dst_lst_index = index_to_usize(*dst_lst_index)?;

        let dst_lst_state =
            try_match_lst_mint_on_list(*self.dst_lst_mint.pubkey(), list, dst_lst_index)?;
```

Currently, the "start_rebalance" writes "dst_lst_index" to the "rebalance_record" account, and the subsequent "end_rebalance" checks if the "dst_lst_index" in the "rebalance_record" matches the "dst_lst_mint" passed in the accounts.

However, a malicious user may insert a "remove_lst" in between so that the index points to a different "dst_lst_mint". By doing this, the disabled check in "start_rebalance" can be bypassed.

Although this instruction can only be invoked by the rebalance authority, consider validating if the "dst_lst" is consistent during rebalancing. In fact, we believe that "remove_lst" should not be allowed during rebalancing, as stated in issue [P1-L-01] Missing PoolState validation when removing LST.

Resolution

This issue has been resolved by commit 4c6775fc06c07fa54cf6b994ef2764ac6a481d2e.

S CONTROLLER

[P1-I-06] Incomplete lst_mint check when adding LST

```
/* libs/s-controller-lib/src/accounts_resolvers/add_lst.rs */
    pub fn resolve(self) -> Result<(AddLstKeys, LstStateBumps), SControllerError> {
        let AddLstFreeArgs {
            payer,
            sol_value_calculator,
            pool_state: pool_state_acc,
            lst_mint,
        } = self;
...
        Ok((
            AddLstKeys {
                payer,
                sol_value_calculator,
                lst_mint: *lst_mint.pubkey(),
                admin: pool_state.admin,
                pool_reserves,
                protocol_fee_accumulator,
                protocol_fee_accumulator_auth: PROTOCOL_FEE_ID,
                pool_state: POOL_STATE_ID,
                lst_state_list: LST_STATE_LIST_ID,
                associated_token_program: spl_associated_token_account::ID,
                system_program: system_program::ID,
                lst_token_program: *lst_mint.owner(),
            },
            LstStateBumps {
                protocol_fee_accumulator: protocol_fee_accumulator_bump,
                pool_reserves: pool_reserves_bump,
            },
        ))
    }
```

In the "`add_lst`" instruction, the program solely verifies the owner of the user-provided "`lst_mint`" without conducting a comprehensive examination of whether it is a valid mint account.

Although this instruction can only be invoked by the admin, it is advisable to add a verification step ensuring that "`lst_mint`" is a valid mint account.

Resolution

This issue has been resolved by commit 4c6775fc06c07fa54cf6b994ef2764ac6a481d2e.

S CONTROLLER

[P1-I-07] Missing lst_mint check when withdrawing protocol fee

```
/* libs/s-controller-lib/src/accounts_resolvers/withdraw_protocol_fees.rs */
    pub fn resolve(self) -> Result<WithdrawProtocolFeesKeys, ProgramError> {
        let WithdrawProtocolFeesFreeArgs {
            pool_state: pool_state_acc,
            withdraw_to,
        } = self;
...
        let lst_mint = token_account_mint(&withdraw_to)?;
...
        Ok(WithdrawProtocolFeesKeys {
            pool_state: POOL_STATE_ID,
            protocol_fee_accumulator,
            protocol_fee_accumulator_auth: PROTOCOL_FEE_ID,
            protocol_fee_beneficiary: pool_state.protocol_fee_beneficiary,
            token_program: *withdraw_to.owner(),
            withdraw_to: *withdraw_to.pubkey(),
            lst_mint,
        })
    }
```

In the "`withdraw_protocol_fees`" instruction, the protocol fee beneficiary can withdraw protocol fees associated with the specified "`lst_mint`" from the protocol fee account.

However, it doesn't check if the "`lst_mint`" is a legitimate mint within the "`lst_state_list`". Nevertheless, since this function is exclusively callable by a specific user designated by the admin, the lack of this check does not pose a security threat.

Resolution

The team acknowledged this issue and decided to keep it as is. The reason is that if the check is added, the protocol fee beneficiary will not be able to withdraw accumulated protocol fees for a LST that was previously on the list but removed.

S CONTROLLER

[P1-I-08] Incorrect in_sol_value calculation in a corner case

Within the "`SwapExactOut`" instruction, the program computes the value of the specified outgoing LST as per user instruction, and then determines the requisite amount of input LST. However, during the calculation process, the "`in_sol_value`" calculation may be incorrect in a corner case.

```
/* programs/s-controller/src/processor/swap_exact_out.rs */
let out_sol_value = dst_lst_cpi.invoke_lst_to_sol(amount)?.max;
let in_sol_value = pricing_cpi.invoke_price_exact_out(PricingProgramIxArgs {
    amount,
    sol_value: out_sol_value,
})?;

/* libs/pricing-programs/flat-fee-lib/src/calc/price_exact_out.rs */
pub fn calculate_price_exact_out(
    CalculatePriceExactOut {
        input_fee_bps,
        output_fee_bps,
        sol_value,
    }: CalculatePriceExactOut,
) -> Result<u64, FlatFeeError> {
    let fee_bps = input_fee_bps
        .checked_add(output_fee_bps)
        .ok_or(FlatFeeError::MathError)?;
    let post_fee_bps: u64 = BPS_DENOMINATOR_I16
        .checked_sub(fee_bps)
        .and_then(|v| u64::try_from(v).ok())
        .ok_or(FlatFeeError::MathError)?;
    let post_fee_bps = U64RatioFloor {
        num: post_fee_bps,
        denom: BPS_DENOMINATOR,
    };
    let result = post_fee_bps
        .reverse(sol_value)
        .map_err(|_e| FlatFeeError::MathError)?;
    Ok(result.max)
}

/* sanctum-solana-utils-e0a9b6e99d4ff9ca/2d1718f/sanctum-token-ratio/src/u64_ratio_floor.rs */
pub fn reverse(&self, amt_after_apply: u64) -> Result<U64ValueRange, MathError> {
    let d: u128 = self.denom.into();
    let n: u128 = self.num.into();
    let is_zero = d == 0 || n == 0;
    if is_zero {
        if amt_after_apply == 0 {
            return Ok(U64ValueRange::full());
        } else {
            return Err(MathError);
        };
    }
```

As shown in the above code snippet, if the user intends to exchange a quantity of some LST resulting in a computed SOL value, represented by "`out_sol_value`" of 0, and the combined fee rate for input and output LSTs is 100%, invoking the "`reverse`" function would have both "`n`" and "`amt_after_apply`" set to 0. Consequently, this would return a complete range from 0 to "`u64::MAX`", resulting in the computed "`in_sol_value`" being set to "`u64::MAX`".

Although such a scenario is highly improbable and, due to the excessively large numerical value, may only lead to a revert without causing any loss for both the user and the protocol, it is advisable to introduce relevant checks to ensure "`out_sol_value`" is not equal to 0.

Resolution

This issue has been resolved by commit f1c6ff18fe004d22f3ab662a0d39b10739f21f1e.

S CONTROLLER

[P1-I-09] Missing redundancy check when adding new disable pool authority

The admin can augment the list of authorities authorized to disable a pool by invoking the "`add_disable_pool_authority`" instruction.

However, in "`add_disable_pool_authority`", it doesn't check whether the authority being added is already present in the list. Consequently, inappropriate usage may result in the presence of multiple records for a single authority in the list.

However, when invoking "`remove_disable_pool_authority`" for deletion, only the initial record is removed, potentially leading to unintended consequences.

Resolution

This issue has been resolved by commits e2d1e2c474884b39defd4fd324d015fc6122ca4f and ad28d0c1d57a23642cee1829a40a498a4f506527.

S CONTROLLER

[P1-I-10] Missing healthy check during rebalancing

In the rebalance operation, the current implementation solely verifies that the total value in the pool post-rebalancing is not less than pre-rebalancing. However, it lacks the incorporation of heuristics to ensure that the state of the pool is genuinely more balanced after the rebalance.

It is advisable to introduce relevant checks to prevent intentional or unintentional deterioration of the pool's health by the caller through the rebalance operation.

Resolution

This issue has been mitigated by commit 4c6775fc06c07fa54cf6b994ef2764ac6a481d2e by ensuring the pool isn't in an unexpected state before rebalance.

S CONTROLLER

[P1-I-11] Documentation and IDL nitpicks

In addition to the issues mentioned in P1-I-02, there are several minor issues pertaining to the documentation and IDL, as delineated below:

- The flat fee pricing program documentation for "`remove_lst`" requires passing in system program which is not necessary.
- The flat fee pricing program documentation for "`add_lst`" does not specify the requirement of "`payer`" to be mutable.
- The S Controller IDL for "`disable_pool`" requires "`disable_pool_authority_list`" to be mutable, which is not necessary.
- Incorrect description of "`add_lst`" and "`remove_lst`" in the flat fee pricing program documentation.

Resolution

This issue has been resolved by commit 66e1e123aec42175099ddcd777198adf1620d732.

PRICING PROGRAM

[P2-L-01] Arbitrary account closure when removing LST

In the "RemoveLst" instruction, subsequent to conducting basic checks on the user-provided accounts, the instruction proceeds to close the "fee_acc" account and transfer the rent to the specified "refund_rent_to" account.

```
/* programs/pricing-programs/flat-fee/src/processor/remove_lst.rs */
fn verify_remove_lst<'me, 'info>(
    accounts: &'me [AccountInfo<'info>],
) -> Result<RemoveLstAccounts<'me, 'info>, ProgramError> {
    let actual: RemoveLstAccounts = load_accounts(accounts)?;

    let free_args = RemoveLstFreeArgs {
        refund_rent_to: *actual.refund_rent_to.key,
        state_acc: actual.state,
        fee_acc: *actual.fee_acc.key,
    };
    let expected: RemoveLstKeys = free_args.resolve()?;

    remove_lst_verify_account_keys(actual, expected).map_err(log_and_return_wrong_acc_err)?;
    remove_lst_verify_account_privileges(actual).map_err(log_and_return_acc_privilege_err)?;

    Ok(actual)
}

close_account(CloseAccountAccounts {
    refund_rent_to,
    close: fee_acc,
})?;
```

Notably, this directive lacks checks on the "fee_acc" account, enabling the closure of any account, including the program state singleton for the current program.

However, given that only the manager possesses the authority to invoke this instruction, the potential impact is considered minimal.

Recommendations

It is recommended to implement validation checks on the seeds of "fee_acc" to ensure it is a legitimate "FeeAccount" PDA.

Resolution

This issue has been resolved by commit 4c6775fc06c07fa54cf6b994ef2764ac6a481d2e.

PRICING PROGRAM

[P2-I-01] Better signed fee bps check

```
/* programs/pricing-programs/flat-fee/src/processor/add_lst.rs */
verify_signed_fee_bps_bound(args.input_fee_bps)?;
verify_signed_fee_bps_bound(args.output_fee_bps)?;

/* libs/pricing-programs/flat-fee-lib/src/fee_bound.rs */
pub fn verify_signed_fee_bps_bound(fee_bps_i16: i16) -> Result<(), FlatFeeError> {
    if MAX_FEE_BPS < fee_bps_i16 {
        return Err(FlatFeeError::SignedFeeOutOfBound);
    }
    Ok(())
}
```

When adding a new LST, the "`AddLst`" instruction should be called to record the fee rates applicable when the specified LST is utilized as input or output. Please note that the fee rates employed can assume negative values, indicative of rewarding users.

However, the "`verify_signed_fee_bps_bound`" function does not impose constraints on the maximum percentage for rewards. Furthermore, the apparent upper limit of 100% for fee rates might be too high, considering that practical computations often involve the summation of the respective fee rates for the two LSTs involved. Consequently, capping the maximum fee rate at 50% may be a better constraint.

Resolution

The team acknowledges the issue and has mitigated the missing negative bound issue in commit 4c6775fc06c07fa54cf6b994ef2764ac6a481d2e by limiting the fee range to [-100%, +100%].

PRICING PROGRAM

[P2-I-02] Missing lst_mint check when adding LST

```
/* libs/pricing-programs/flat-fee-lib/src/account_resolvers/add_lst.rs */
pub fn resolve(self) -> Result<(AddLstKeys, FeeAccountCreatePdaArgs), FlatFeeError> {
    let Self {
        payer,
        state_acc,
        lst_mint,
    } = self;

    if *state_acc.pubkey() != STATE_ID {
        return Err(FlatFeeError::IncorrectProgramState);
    }

    let bytes = &state_acc.data();
    let state: &ProgramState = try_program_state(bytes)?;

    let find_pda_args = FeeAccountFindPdaArgs { lst_mint };
    let (fee_acc, bump) = find_pda_args.get_fee_account_address_and_bump_seed();

    Ok((
        AddLstKeys {
            manager: state.manager,
            payer,
            fee_acc,
            lst_mint,
            state: STATE_ID,
            system_program: system_program::ID,
        },
        FeeAccountCreatePdaArgs {
            find_pda_args,
            bump,
        },
    ))
}
```

Within the "`add_lst`" instruction, the user-provided "`lst_mint`" serves as a free argument without undergoing much checks.

Although this instruction can only be invoked by the manager, it is advisable to add a verification step ensuring that "`lst_mint`" is a valid mint account.

Resolution

This issue has been resolved by commit 4c6775fc06c07fa54cf6b994ef2764ac6a481d2e.

SOL VALUE CALCULATOR

[P3-I-01] Round up withdraw fee for SPL stake pool

```
/* libs/sol-value-calculator-programs/spl-calculator-lib/src/calc.rs */
pub const fn stake_withdrawal_fee(&self) -> U64FeeFloor<u64, u64> {
    let Self {
        stake_withdrawal_fee_numerator,
        stake_withdrawal_fee_denominator,
        ..
    } = self;
    U64FeeFloor {
        fee_num: *stake_withdrawal_fee_numerator,
        fee_denom: *stake_withdrawal_fee_denominator,
    }
}
...
fn calc_lst_to_sol(&self, pool_tokens: u64) -> Result<U64ValueRange, ProgramError> {
    let AmtsAfterFee {
        amt_after_fee: pool_tokens_burnt,
        ..
    } = self.stake_withdrawal_fee().apply(pool_tokens)?;
    let withdraw_lamports = self.lst_to_lamports_ratio().apply(pool_tokens_burnt)?;
    Ok(U64ValueRange::single(withdraw_lamports))
}
```

In the calculation of "`lst_to_sol`" within the SPL stake pool, the program employs a rounding down mechanism for the withdrawal fee.

However, in the actual implementation of the SPL stake pool, rounding up is applied to the withdrawal fee. Although the disparity is a mere 1 lamport, it is recommended to align the implementation with the SPL stake pool for consistency.

Resolution

This issue has been resolved by commits 13933109110b0e430b48a19b47a15869d4ebdac1 and bc8a4aaf5b6e8465c9c19dbc5d3072fe410735cf.

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 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.
- 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

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, Sec3 (formerly Soteria) is a leading blockchain security company. We are also building sophisticated 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.

Sec3™