Scallop

Audit

Presented by:

OtterSec contact@osec.io

Akash Gurugunti sud0u53r.ak@osec.io

Ilardi Marco goten@osec.io

Robert Chen r@osec.io

Sangsoo Kang sangsoo@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Scallop engaged OtterSec to perform an assessment of the `sui-lending-protocol` program. This assessment was conducted between July 3rd and July 15th, 2023. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 14 findings in total.

In particular, we identified the absence of version validation (OS-SCA-ADV-00) and incorrect key verification during the obligation lock process (OS-SCA-ADV-01).

We also made numerous suggestions around avoiding unnecessary operations (OS-SCA-SUG-00), directly accessing fields for updating delay attributes (OS-SCA-SUG-01), and eliminating obsolete constants in the codebase (OS-SCA-SUG-02).

02 | Scope

The source code was delivered to us in a git repository at github.com/scallop-io/sui-lending-protocol. This audit was performed against commit 128ffbd.

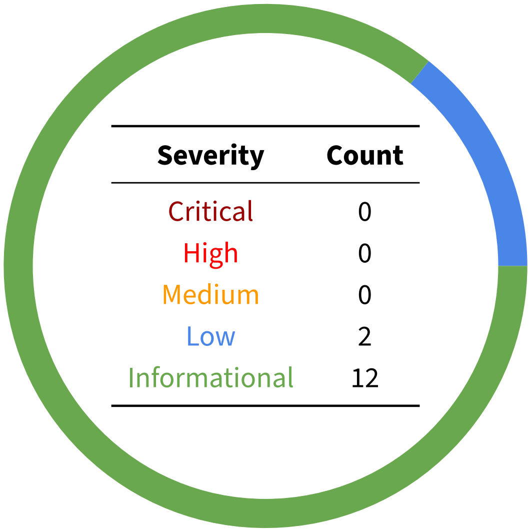
A brief description of the programs is as follows.

Name	Description
sui-lending-protocol	A money market designed specifically for the Sui ecosystem with a dynamic money market that offers high-interest lending, low-fee borrowing, an Automated Market Maker (AMM), and an asset management tool.

03 | Findings

Overall, we reported 14 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 help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	0
Low	2
Informational	12

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-SCA-ADV-00	Low	Resolved	The functions in `accrue_interest.move` omit the version check that prevents execution in the previous version.
OS-SCA-ADV-01	Low	Resolved	`obligation::lock` invokes an incorrect function for validating the lock key.

OS-SCA-ADV-00 [low] | Lack Of Version Check

Description

All user-callable functions perform a version check to ensure they utilize the most recent module whenever the protocol undergoes an upgrade. However, the functions within `accrue_interest.move` do not include this version validation, which may allow them to execute in their previous versions even after a protocol upgrade.

Remediation

Insert a validation step to confirm the current version by calling `assert_current_version`.

accrue_interest.move DIFF

```diff
@@ -3,12 +3,16 @@ module protocol::accrue_interest {
+ use protocol::version::{Self, Version};

  public fun accrue_interest_for_market(
+   version: &Version,
    market: &mut Market,
    clock: &Clock,
  ) {
+   version::assert_current_version(version);
+
    let now = clock::timestamp_ms(clock) / 1000;
    market::accrue_all_interests(market, now);
  }
@@ -19,11 +23,14 @@ module protocol::accrue_interest {
  public fun accrue_interest_for_market_and_obligation(
+   version: &Version,
    market: &mut Market,
    obligation: &mut Obligation,
    clock: &Clock,
  ) {
-   accrue_interest_for_market(market, clock);
+   version::assert_current_version(version);
+
+   accrue_interest_for_market(version, market, clock);
    obligation::accrue_interests_and_rewards(obligation, market);
  }
}
```

Patch

Fixed in f090a72.

OS-SCA-ADV-01 [low] | Incorrect Key Check

Description

`obligation::lock` is designed to lock the obligation functionality. Currently, the function invokes `assert_reward_key_in_store`, which is inconsistent with its intended purpose. Instead, `assert_lock_key_in_store` should be invoked, as the function is specifically designed to handle the locking of the obligation, not the management of rewards.

Remediation

Check if `ObligationAccessStore` contains `lock_key` instead of `reward_key`.

obligation.move DIFF

```diff
@@ -257,13 +257,13 @@ module protocol::obligation {
public fun lock<T: drop>(
    // ...
-   obligation_access::assert_reward_key_in_store(obligation_access_store, key);
+   obligation_access::assert_lock_key_in_store(obligation_access_store, key);
```

Patch

Fixed in f090a72.

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 lead to security issues in the future.

ID	Description
OS-SCA-SUG-00	Adding an `else` statement may remove the occurrence of unnecessary operations.
OS-SCA-SUG-01	The initialization of `new_delay` may be avoided by directly updating the value of the structure.
OS-SCA-SUG-02	Remove obsolete constants in the codebase for maintenance and clarity.
OS-SCA-SUG-03	`lock_deposit_collateral` and `lock_withdraw_collateral` are assigned incorrectly in `obligation::lock`.
OS-SCA-SUG-04	The coin type does not need to be stored.
OS-SCA-SUG-05	`fixed_point32::zero` invokes `create_from_rational` instead of `create_from_raw_value`.
OS-SCA-SUG-06	Unnecessary fields in `WitTable` and `AcTable`.
OS-SCA-SUG-07	`balance_bag` holds empty balances without removing them.
OS-SCA-SUG-08	Optimize the process of liquidation of an obligation by avoiding repeated calls to `&get<DebtType>()`.
OS-SCA-SUG-09	`pyth_rule::set_price` does not check the confidence value returned from `pyth_adaptor::get_pyth_price`.
OS-SCA-SUG-10	Add checks to avoid reverts in the future.
OS-SCA-SUG-11	Incorrect naming of variables in `supra_registry::init`.

OS-SCA-SUG-00 | Avoid Unnecessary Operations

Description

In `incentive_rewards.move`, `set_reward_factor` executes an unnecessary mutable borrow operation on the `reward_factors` table, where a new `coin_type` has been introduced. This procedure is redundant, considering that the recently incorporated entry already possesses the correct `reward_factor` value.

Remediation

Insert an `else` statement to avoid redundancy.

incentive_rewards.move DIFF

```
@@ -30,9 +30,9 @@ module protocol::incentive_rewards {
        reward_factor: factor,
      };
      wit_table::add(RewardFactors{}, reward_factors, coin_type, reward_factor);
+   } else {
+     let reward_factor = wit_table::borrow_mut(RewardFactors{}, reward_factors,
    ↪   coin_type);
+     reward_factor.reward_factor = factor;
    };
-
-   let reward_factor = wit_table::borrow_mut(RewardFactors{}, reward_factors,
    ↪   coin_type);
-   reward_factor.reward_factor = factor;
  }
}
```

Patch

Fixed in d6d2de5.

OS-SCA-SUG-01 | Direct Field Access

Description

In app.move, when updating the delay attributes of admin_cap, the current implementation involves initializing a new variable new_delay and duplicating its value to change_delay. This occurs in three functions:

1. extend_interest_model_change_delay.
2. extend_risk_model_change_delay.
3. extend_limiter_change_delay.

However, this step is unnecessary and may be optimized by directly increasing the delay values in admin_cap, eliminating the requirement of the new_delay variable.

Remediation

Directly increase delay values in admin_cap without utilizing new_delay.

app.move DIFF

```diff
@@ -56,24 +56,21 @@ module protocol::app {
      admin_cap: &mut AdminCap,
      delay: u64,
    ) {
-     let new_delay = admin_cap.interest_model_change_delay + delay;
-     admin_cap.interest_model_change_delay = new_delay;
+     admin_cap.interest_model_change_delay = admin_cap.interest_model_change_delay +
      ↪ delay;

    public fun extend_risk_model_change_delay(
      admin_cap: &mut AdminCap,
      delay: u64,
    ) {
-     let new_delay = admin_cap.risk_model_change_delay + delay;
-     admin_cap.risk_model_change_delay = new_delay;
+     admin_cap.risk_model_change_delay = admin_cap.risk_model_change_delay + delay;

    public fun extend_limiter_change_delay(
      admin_cap: &mut AdminCap,
      delay: u64,
    ) {
-     let new_delay = admin_cap.limiter_change_delay + delay;
-     admin_cap.limiter_change_delay = new_delay;
+     admin_cap.limiter_change_delay = admin_cap.limiter_change_delay + delay;
```

Patch

Fixed in 020bcae.

OS-SCA-SUG-02 | Eliminate Obsolete Constants

Description

Several constants in the codebase are declared without being utilized. These unutilized constants may confuse developers and make the codebase harder to maintain. The constants in question are:

1. `u64::DIVIDE_BY_ZERO`.
2. `pyth_rule::rule::U8_MAX`.
3. `cetus_adaptor::cetus_flash_loan::ERepayTypeIncorrect`.
4. `supra_rule::rule::U8_MAX`.
5. `supra_rule::rule::U64_MAX`.

Remediation

Remove the aforementioned unutilized constants.

OS-SCA-SUG-03 | Incorrect Assignment Of Function Parameters

Description

In `obligation.move`, `lock` assigns:

1. `lock_deposit_collateral` to `self.withdraw_collateral_locked`.
2. `lock_withdraw_collateral` to `self.deposit_collateral_locked`.

This assignment is incorrect and may result in unexpected consequences in the program's execution.

Remediation

Update the assignments of `lock_deposit_collateral` and `lock_withdraw_collateral` in `lock`.

obligation.move DIFF

```diff
@@ -20,9 +20,9 @@ public fun lock<T: drop>(
    obligation_access::assert_reward_key_in_store(obligation_access_store, key);

    self.lock_key = option::some(type_name::get<T>());
    self.borrow_locked = lock_borrow;
    self.repay_locked = lock_repay;
-   self.withdraw_collateral_locked = lock_deposit_collateral;
-   self.deposit_collateral_locked = lock_withdraw_collateral;
+   self.deposit_collateral_locked = lock_deposit_collateral;
+   self.withdraw_collateral_locked = lock_withdraw_collateral;
    self.liquidate_locked = lock_liquidate;
```

Patch

Fixed in f090a72.

OS-SCA-SUG-04 | Coin Type Not Required

Description

Storing the coin type in `incentive_rewards::RewardFactor` is redundant, as the coin type already serves as the key for retrieving the reward factor from `WitTable`.

interest_model.move RUST

```rust
struct RewardFactor has store {
coin_type: TypeName,
reward_factor: FixedPoint32,
}
```

Remediation

Store the `reward_factor` directly in `WitTable` and remove the redundant coin type storage.

OS-SCA-SUG-05 | Optimize Zero Fixed Point

Description

In `fixed_point32.move`, `zero` creates a fixed point object representing zero utilizing `fixed_point32::create_from_rational(0, 1)`. However, a simpler approach exists. `fixed_point32::create_from_raw_value(0)` may directly generate a fixed point object that represents zero.

Remediation

Replace the `fixed_point32::create_from_rational(0, 1)` inside `zero` with `fixed_point32::create_from_raw_value(0)`.

fixed_point32.move DIFF

```diff
@@ -10,7 +10,7 @@ public fun zero(): FixedPoint32 {
-   fixed_point32::create_from_rational(0, 1)
+   fixed_point32::create_from_raw_value(0)
  }
```

OS-SCA-SUG-06 | Unnecessary Fields

Description

`wit_table::WitTable` and `ac_table::AcTable` contain the unnecessary field `with_keys`. The existence of keys may be verified by checking whether the optional keys field is Some or None.

wit_table.move RUST

```rust
struct WitTable<phantom T: drop, K: copy + drop + store, phantom V: store> has
  ↪  key, store {
  id: UID,
  table: Table<K, V>,
  keys: option::Option<VecSet<K>>,
  with_keys: bool
}
```

ac_table.move RUST

```rust
struct AcTable<phantom T: drop, K: copy + drop + store, phantom V: store> has key,
  ↪  store {
  id: UID,
  table: Table<K, V>,
  keys: option::Option<VecSet<K>>,
  with_keys: bool
}
```

Also, the `effective_epoches` field present in the following events may be omitted since it is derivable from the `current_epoch` and `delay_epoches` fields:

- `InterestModelChangeCreated`
- `LimiterUpdateLimitChangeCreatedEvent`
- `LimiterUpdateParamsChangeCreatedEvent`
- `RiskModelChangeCreated`

Remediation

Eliminate the `with_keys` field from `wit_table::WitTable` and `ac_table::AcTable`. Instead, check for the existence of keys by evaluating the state of the optional keys field. Furthermore, remove the `effective_epoches` field from:

- `InterestModelChangeCreated`
- `LimiterUpdateLimitChangeCreatedEvent`

- `LimiterUpdateParamsChangeCreatedEvent`
- `RiskModelChangeCreated`

Instead, derive it by utilizing the `current_epoch` and `delay_epoches` fields.

OS-SCA-SUG-07 | Remove Empty Balances

Description

`obligation_collaterals::decrease` removes collaterals that have a zero amount from `WitTable`. On the other hand, `balance_bag` of `obligation` does not eliminate empty balances. To maintain code consistency and ensure clarity, remove empty balances.

Remediation

Remove the empty balance of `balance_bag` when withdrawing the collateral from `obligation`.

OS-SCA-SUG-08 | Avoid Repeated Calls

Description

In `liquidator.move`, `liquidate_obligation_with_cetus_pool_only_a` and `liquidate_obligation_with_cetus_pool_only_b` each invoke `&get<DebtType>()` twice. Repeated function calls may impact performance. Storing the result of `&get<DebtType>()` in a variable and reusing it would enhance the efficiency of the code.

liquidator.move RUST

```rust
public fun liquidate_obligation_with_cetus_pool_only_a<DebtType, B>(
  [...]
) {
  /// Make sure the obligation has DebtType, and CollateralType
  if (
    vector::contains(&obligation::debt_types(obligation), &get<DebtType>()) ==
  ↪   false ||
      vector::contains(&obligation::collateral_types(obligation),
  ↪   &get<DebtType>()) == false
  )
  [...]

public fun liquidate_obligation_with_cetus_pool_only_b<A, DebtType>(
  [...]
) {
  /// Make sure the obligation has DebtType, and CollateralType
  if (
    vector::contains(&obligation::debt_types(obligation), &get<DebtType>()) ==
  ↪   false ||
      vector::contains(&obligation::collateral_types(obligation),
  ↪   &get<DebtType>()) == false
  ) return;
```

Remediation

Store the result of `&get<DebtType>()` in a variable to avoid repeated function calls.

OS-SCA-SUG-09 | Missing Confidence Check

Description

In `pyth_rule::set_price`, the confidence value returned from `get_pyth_price` is not validated. While there is a check to confirm the primary oracle's price against at least half of the secondary oracles' prices, it is advisable to verify that the confidence level is not excessively high.

Remediation

Check that the returned value of confidence is not too high in `pyth_rule::set_price`.

OS-SCA-SUG-10 | Additional Checks To Avoid Reverts

Description

In interest_model::create_interest_model_change, it is advisable to include additional checks to prevent potential reverts in the future, particularly those created by dividing by zero and underflow errors.

Remediation

Integrate the following checks to prevent dividing by zero and underflow errors, thus avoiding potential reverts of this nature in the future.

interest_model.move RUST

```rust
  public(friend) fun create_interest_model_change<T>(
  _: &AcTableCap<InterestModels>,
  base_rate_per_sec: u64,
  interest_rate_scale: u64,
  borrow_rate_on_mid_kink: u64,
  mid_kink: u64,
  borrow_rate_on_high_kink: u64,
  high_kink: u64,
  max_borrow_rate: u64,
  revenue_factor: u64,
  borrow_weight: u64,
  scale: u64,
  min_borrow_amount: u64,
  change_delay: u64,
  ctx: &mut TxContext,
): OneTimeLockValue<InterestModel> {
      [...]
      assert!(mid_kink != 0 && high_kink < 1 && base_rate <=
  ↪   borrow_rate_on_mid_kink <= borrow_rate_on_high_kink <= max_borrow_rate);
      [...]
  }
```

OS-SCA-SUG-11 | Incorrect Variable Names

Description

In `supra_registry::init`, the variables `pyth_registry` and `pyth_registry_cap` should be named `supra_registry` and `supra_registry_cap` respectively for better code clarity.

Remediation

Rename `pyth_registry` and `pyth_registry_ca` to `supra_registry` and `supra_registry_cap` respectively.

A | Vulnerability Rating Scale

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

Critical

Vulnerabilities that immediately lead to 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 could lead to 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 could lead to denial of service scenarios or degraded usability.

Examples:

- Malicious input that causes computational limit exhaustion
- Forced exceptions in normal user flow

Low

Low probability vulnerabilities which could still be 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 implementation of the program 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 sum, 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 get a comprehensive understanding of 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.