Aries Markets
Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io
Akash Gurugunti Sud0u53r.ak@osec.io
Shiva Shankar Genji sh1v@osec.io

Contents

01 | Executive Summary

Overview

Aries Markets engaged OtterSec to perform an assessment of the `aries` program. This assessment was conducted between October 3rd and October 21st, 2022.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team over to streamline patches and confirm remediation. We delivered final confirmation of the patches November 11th, 2022.

Key Findings

Over the course of this audit engagement, we produced 10 findings total.

In particular, we reported a number of issues around oracle pricing (OS-ARS-ADV-00), liquidation calculations (OS-ARS-ADV-01), and denial of service vectors (OS-ARS-ADV-02).

We also made suggestions around improved admin validation, code style, and utilization optimization (OS-ARS-SUG-00, OS-ARS-SUG-02, OS-ARS-SUG-04).

Finally, we presented a discussion of formal verification and explored ideas for possible data invariants as well as miscellaneous function specifications (OS-ARS-VER-00, OS-ARS-VER-01).

Overall the code quality was extremely high. The Aries team was responsive to feedback and a pleasure to work with.

02 | Scope

The source code was delivered to us in a git repository at github.com/Aries-Markets/aries-markets. This audit was performed against commit `add3db2`.

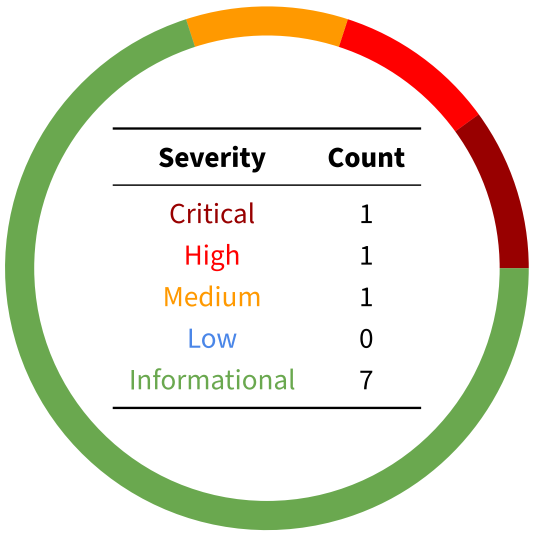
A brief description of the programs is as follows.

Name	Description
Aries Markets	Lending & borrowing protocol built on Aptos

03 | Findings

Overall, we report 10 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 don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	1
High	1
Medium	1
Low	0
Informational	7

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-ARS-ADV-00	Critical	Resolved	Miscalculation in Pyth oracle feed leads to incorrect oracle values
OS-ARS-ADV-01	High	Resolved	The settle share amount is improperly calculated while performing liquidation.
OS-ARS-ADV-02	Medium	Resolved	Denial of Service while removing shares from the reserve.

OS-ARS-ADV-00 [crit] [resolved] | Improper Oracle Calculations

Description

Pyth price calculations in `oracle.move` are performed incorrectly.

oracle/sources/oracle.move RUST

```
pyth_price = option::some(decimal::div(

     ↪  decimal::from_u64(pyth::i64::get_magnitude_if_positive(&pyth::price::get_price(&
    decimal::from_u128(math128::pow(10,
     ↪  (pyth::i64::get_magnitude_if_positive(&pyth::price::get_expo(&price)))
     ↪  as u128)))
))
```

The price should be multiplied by, not divided by, the magnitude. This code also errors if the magnitude is negative.

Remediation

Multiply by the magnitude.

Patch

Resolved in 2a62fc8.

OS-ARS-ADV-01 [high] [resolved] | Improper Calculation in Liquidation

Description

In `profile.move`, the amount of shares that need to be settled is calculated incorrectly. In the `else` case of the liquidation function, the `settle_share_amount` should be calculated from the `repay_amount` using the `get_share_amount_from_borrow_amount` function. Instead, the repay amount is directly returned as the settle share amount.

Since the value of shares increases with the accumulation of borrow interest, the actual `settle_share_amount` would be less than the `repay_amount`. Directly subtracting the repay amount from borrowed shares will decrease the overall value of a borrowed share and affect the health of the protocol.

Proof of Concept

Consider the following scenario:

1. A user borrows X amount from the protocol. This is recorded as X number of borrowed shares.
2. After some time, borrow interest accumulates and the total borrowed amount on the reserve increases, thus increasing the borrow share value.
3. Now, if the user's account loses health and the liquidator liquidates X amount of the loan amount, only X amount of shares are subtracted from the user's borrowed shares in spite of the increase borrowed share value.

Remediation

A possible remediation is converting `repay_amount` to shares using the `get_share_amount_from_borrow_amount` function in the else case of the `liquidate_profile` function.

aries/sources/profile.move DIFF

```diff
     if (decimal::gte(bonus_liquidation_value, collateral_value)) {
         let repay_percentage = decimal::div(collateral_value,
      ↪  bonus_liquidation_value);
         let settle_amount = decimal::mul(max_liquidation_amount,
      ↪  repay_percentage);
         let repay_amount = decimal::ceil_u64(settle_amount);
         let withdraw_amount = withdraw_reserve.collateral_amount;
-        (repay_amount, withdraw_amount, settle_amount)
 +       (repay_amount, withdraw_amount,
      ↪   reserve::get_share_amount_from_borrow_amount_dec(settle_amount))
     } else {
         let withdraw_percentage = decimal::div(bonus_liquidation_value,
      ↪   collateral_value);
         let settle_amount = max_liquidation_amount;
         let repay_amount = decimal::ceil_u64(settle_amount);
         let withdraw_amount = decimal::floor_u64(
             decimal::mul_u64(withdraw_percentage,
      ↪   withdraw_reserve.collateral_amount)
         );
 -       (repay_amount, withdraw_amount, settle_amount)
 +       (repay_amount, withdraw_amount,
      ↪   reserve::get_share_amount_from_borrow_amount_dec(settle_amount))
     }
```

Patch

Resolved in ba3c164.

OS-ARS-ADV-02 [med] [resolved] | DOS While Removing Shares From Reserve

Description

In `profile.move`, the `try_subtract_profile_reward_share` function checks whether the profile has a farm for that specific reserve type while subtracting shares from a profile farm. If a farm doesn't exist, the reward for that reserve is created after the profile is created, so the function skips the subtraction of shares.

This case is not handled while subtracting shares from the reserve farm in the `reserve::try_remove_reserve_reward_share` function. This leads to the subtraction of shares that were previously absent in the reserve.

Proof of Concept

Consider the following scenario:

1. A reserve is created.
2. A user (user1) mints and deposits collateral to the reserve. Since the farm doesn't have any rewards, no shares are added to `reserve_farm` or `profile_farm`.
3. A reward is added to the `reserve_farm` for that reserve.
4. Another user (user2) mints and deposits collateral to the reserve. Since there is a `reserve_farm`, shares are added to the `reserve_farm` and `profile_farm`.
5. If user1 removes their collateral while they have no shares in their `profile_farm`, no shares are subtracted from their profile farm. Shares are still removed from their `reserve_farm`.
6. Now if user2 tries to withdraw their collateral, the reserve farm gives an error since user1 has subtracted shares from the reserve farm.

Remediation

A possible method of remediation is ensuring the amount of shares subtracted from the reserve farm is equal to the amount of shares subtracted from the profile farm.

Patch

Resolved in 7f0519d.

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 antipatterns and could lead to security issues in the future.

ID	Description
OS-ARS-SUG-00	Unlimited deposit and borrow limits are not implemented.
OS-ARS-SUG-01	Struct fields should be validated before updating them.
OS-ARS-SUG-02	Optimal utilization should be properly constrained.
OS-ARS-SUG-03	Assert constraints should generate the proper error codes.
OS-ARS-SUG-04	There is no function implemented that allows the admin to withdraw the protocol's fees.

OS-ARS-SUG-00 | Enforcing Unlimited Deposit and Borrow Limits

Description

In `reserve_config.move`, the struct `ReserveConfig` stores deposit and borrow limits that need to be enforced on the relevant reserve. It is mentioned in the comments that if the deposit and borrow limits values of the struct are 0, then no limits will be enforced on the deposit and borrow functions in that reserve. But the behavior described is not implemented in the functions that return those values.

Below is the code snippet that contains the said comments:

aries-config/sources/reserve_config.move RUST

```rust
    // 0 represents no limit
    deposit_limit: u64,
    // 0 represents no limit
    borrow_limit: u64,
```

Remediation

A possible method for remediation is returning `u64_MAX` if those values are `0`. The below code snippet shows the changes that could be made:

aries-config/sources/reserve_config.move DIFF

```diff
    public fun deposit_limit(reserve_config: &ReserveConfig): u64 {
+       if (reserve_config.deposit_limit == 0) {
+           18446744073709551615
+       } else {
            reserve_config.deposit_limit
+       }
    }
```

aries-config/sources/reserve_config.move DIFF

```diff
    public fun borrow_limit(reserve_config: &ReserveConfig): u64 {
+       if (reserve_config.borrow_limit == 0) {
+           18446744073709551615
+       } else {
            reserve_config.borrow_limit
+       }
    }
```

OS-ARS-SUG-01 | Validating Struct Fields Before Updation

Description

In `reserve_config.move` and `interest_rate_config.move`, struct fields are accessed and modified using getter and setter functions generated by a TypeScript script. Conditions enforced while instancing these structs should also be enforced while updating fields of the structs using the setter functions outside of instantiation.

Remediation

A possible method for remediation is asserting the necessary conditions on the fields in their respective setter functions. The below code snippet shows a possible implementation:

aries-config/sources/reserve_config.move DIFF

```
public fun update_loan_to_value(reserve_config: &ReserveConfig,
    ↪  loan_to_value: u8): ReserveConfig {
+   assert!(0 <= loan_to_value && loan_to_value <= 100, 0);
    let new_reserve_config = *reserve_config;
    new_reserve_config.loan_to_value = loan_to_value;
    new_reserve_config
}
```

OS-ARS-SUG-02 | Improper Constraint on Optimal Utilization

Description

In `interest_rate_config.move`, the `optimal_utilization` field in the `InterestRateConfig` struct is constrained to be less than or equal to 100. But this leads to `get_borrow_rate` throwing an error in the case where `utilization` is greater than `optimal_utilization`, since `1 - optimal_utilization`, which is equal to 0, is used in the denominator.

Remediation

A possible method for remediation is constraining the value of `optimal_utilization` to be strictly "less than 100" instead of "less than or equal to 100". The below code snippet shows a possible implementation:

aries-config/sources/interest_rate_config.move DIFF

```diff
public fun new_interest_rate_config(
    min_borrow_rate: u64,
    optimal_borrow_rate: u64,
    max_borrow_rate: u64,
    optimal_utilization: u64
): InterestRateConfig {
-   assert!(0 <= optimal_utilization && optimal_utilization <= 100, 0);
+   assert!(0 <= optimal_utilization && optimal_utilization < 100, 0);
    assert!(min_borrow_rate <= optimal_borrow_rate && optimal_borrow_rate
     ↪  <= max_borrow_rate, 0);

    InterestRateConfig {
        min_borrow_rate,
        optimal_borrow_rate,
        max_borrow_rate,
        optimal_utilization
    }
}
```

OS-ARS-SUG-03 | Usage of Proper Error Codes

Description

While most of the assert constrains generate the proper error codes, there are still some constraints that do not. For example, in `profile::new` function, if the profile doesn't exist, the error code used in the assert is `EPROFILE_ALREADY_EXIST` instead of `EPROFILE_NOT_EXIST`. Similarly some of the constraints simply generate error code 0, instead of relevant error codes.

The below code snippets shows some examples of the usage of improper error codes (highlighted):

aries/sources/profile.move RUST

```rust
public fun new(account: &signer, profile_name: string::String) acquires
    ↪  Profiles {
    let addr = signer::address_of(account);
    assert!(exists<Profiles>(addr), EPROFILE_ALREADY_EXIST);
    let profiles = borrow_global_mut<Profiles>(addr);
```

aries/sources/reserve.move RUST

```rust
public(friend) fun add_collateral<Coin0>(
    lp_coin: Coin<LP<Coin0>>
) acquires Reserves, ReserveCoinContainer {
    let reserve_type_info = type_info<Coin0>();
    assert!(
        reserve_details::allow_collateral(
            &reserve_details(reserve_type_info)
        ),
        0
    );
```

Remediation

To remediate this issue, error codes should be defined and used at relevant places for easier debugging.

OS-ARS-SUG-04 | Fee Withdraw Function For Admin

Description

The fee for the protocol is collected and stored in `ReserveCoinContainer.fee`. This fee is collected while converting liquidity provider tokens to underlying tokens and also while liquidating a profile. Either the deployer (`@aries`) or the admin should be able to withdraw this fee into their account.

Another fee on accumulated borrower's interest is stored in `ReserveDetails.reserve_amount`. This fee is collected on the interest accrued on the borrowed shares. The admin should have the ability to collect this fee amount to his account.

Remediation

A function should be implemented which can only be accessed by the admin to withdraw protocol and accrued interest fees to the admin's account.

06 | Formal Verification

Here we present a discussion about the formal verification of smart contracts. We include example specifications, recommendations, and general ideas to formalize critical invariants.

ID	Description
OS-ARS-VER-00	Specify internal properties of `ReserveConfig` and `InterestRateConfig`.
OS-ARS-VER-01	Function invariant specifications.

OS-ARS-VER-00 | Data Invariant Specifications

1. The conditions asserted while creating a new reserve configuration can be easily enforced via data invariant.

aries-config/sources/reserve_config.move RUST

```
spec ReserveConfig {
    invariant 0 <= loan_to_value && loan_to_value <= 100;
    invariant 0 <= liquidation_bonus_bips && liquidation_bonus_bips
     ↪  <= 10000;
    invariant 0 <= liquidation_threshold && liquidation_threshold <=
     ↪  100;
    invariant 0 <= liquidation_fee_hundredth_bips &&
     ↪  liquidation_fee_hundredth_bips <= 100;
    invariant 0 <= reserve_ratio && reserve_ratio <= 100;
    invariant 0 <= borrow_fee_hundredth_bips &&
     ↪  borrow_fee_hundredth_bips <= 1000000;
    invariant borrow_limit <= deposit_limit;
}
```

2. Similarly, for the interest rate configuration, the conditions can be enforced using invariants.

aries/sources/reward_container.move RUST

```
spec InterestRateConfig {
    invariant 0 <= optimal_utilization && optimal_utilization < 100;
    invariant min_borrow_rate <= optimal_borrow_rate;
    invariant optimal_borrow_rate <= max_borrow_rate;
}
```

OS-ARS-VER-01 | Miscellaneous Function Specifications

1. Explicate when key functions can abort. For example,

   (a) Utility function for depositing coins should abort only if the receiver `CoinStore` is frozen and function for burning coins should never abort.

   aries/sources/utils.move RUST

```
    spec deposit_coin {
        aborts_if global<coin::CoinStore<Coin0>>(
            signer::address_of(recipient)
        ).frozen;
    }

    spec burn_coin {
        aborts_if false;
    }
```

   (b) Queries for checking the existence of a resource should never abort.

   aries/sources/reward_container.move RUST

```
    spec exists_container {
        aborts_if false;
    }
```

2. Decimal module can be formally verified by writing specifications for the functions such as:

decimal/sources/decimal.move RUST

```
    const P64: u128 = 0x10000000000000000;
    spec add {
        aborts_if a.val + b.val >= P64 * P64;
        ensures result.val == a.val + b.val;
    }

    spec sub {
        aborts_if b.val > a.val;
        ensures result.val == a.val - b.val;
    }
```

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