OtterSec

Cetus

Security Assessment

March 30th, 2024 — Prepared by OtterSec

Michał Bochnak embe221ed@osec.io

Andreas Mantzoutas andreas@osec.io

Robert Chen r@osec.io

Table of Contents

01 — Executive Summary

Overview

Cetus Protocol engaged OtterSec to assess the `vaults` program. This assessment was conducted between March 1st and March 20th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 8 findings throughout this audit engagement.

In particular, we identified a vulnerability, where the function neglects to verify if the member possesses the specified role before executing a subtraction operation, which may result in unexpected outcomes (OS-CPV-ADV-00). Furthermore, migrating liquidity with a substantially larger tick range compared to the initial one may result in zero liquidity, rendering the vault unusable (OS-CPV-ADV-01).

We also made recommendations around adherence to coding best practices (OS-CPV-SUG-03) and suggested modifying the code base to enhance its security and efficiency (OS-CPV-SUG-02). Additionally, we advised rearranging the order of multiplication and division operations to minimize rounding errors (OS-CPV-SUG-00).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/CetusProtocol/vaults. This audit was performed against commit c23ab34.

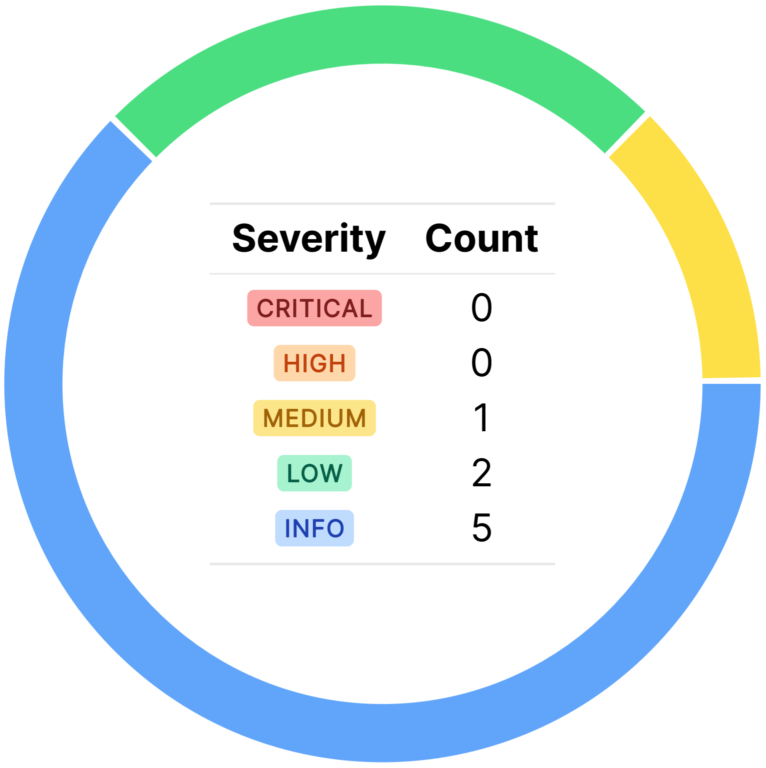
A brief description of the programs is as follows:

Name	Description
vaults	A system specifically designed to automatically manage user liquidity, encompassing the timely reinvestment of fees and rewards and rebalancing when necessary.

03 — Findings

Overall, we reported 8 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	1
LOW	2
INFO	5

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-CPV-ADV-00	MEDIUM	RESOLVED ✓	`remove_role` neglects to verify if the member possesses the specified role before executing a subtraction operation, which may result in unexpected outcomes.
OS-CPV-ADV-01	LOW	RESOLVED ✓	Calling `migrate_liquidity` with a substantially larger tick range compared to the initial one may result in zero liquidity, rendering the vault unusable.
OS-CPV-ADV-02	LOW	RESOLVED ✓	It is possible to exploit the flash loan feature to repeatedly borrow funds, thereby reducing the repayment amount due to slippage adjustments, resulting in the unauthorized withdrawal of funds from the protocol.

Unchecked Role Removal MEDIUM

OS-CPV-ADV-00

Description

The current implementation of `acl::remove_role` directly subtracts the specified role from the member's permissions without initially verifying if the role is set for the member.

>_ *sources/acl.move* rust

```rust
/// @notice Revoke a role for a member in the ACL.
public fun remove_role(acl: &mut ACL, member: address, role: u8) {
    assert!(role < 128, ERoleNumberTooLarge);
    if (linked_table::contains(&acl.permissions, member)) {
        let perms = linked_table::borrow_mut(&mut acl.permissions, member);
        *perms = *perms - (1 << role);
    }
}
```

If the member's specified role is not set, the subtraction operation removes bits from the member's permissions that may correspond to other roles or unrelated data. This may result in unpredictable behavior and may inadvertently grant or revoke other roles for the member. Alternatively, if the subtraction operation attempts to subtract a role that is not set, it will set the role instead due to the two's complement arithmetic used in integer subtraction.

Remediation

Include a check to verify that the specified role is set for the member before executing the subtraction operation. Alternatively, modify the operation to utilize bit-wise `AND` with a bit mask that clears the bit corresponding to the specified role (`perms = *perms & (MAX_U128 - (1 << role))`). This guarantees that only the bit corresponding to the specified role is cleared, regardless of whether the role is currently set or not.

Patch

Fixed in 69ecf6d.

Incorrect Tick Range Expansion LOW OS-CPV-ADV-01

Description

The vulnerability arises when `vaults::migrate_liquidity` is invoked with a significantly larger tick range (`[tick_lower; tick_upper]`) than the initial one. Here, `tick_lower` and `tick_upper` set the range of tick values for the new position, affecting the range of asset prices in the liquidity pool.

>_ *sources/vaults.move* rust

```rust
public fun migrate_liquidity<CoinTypeA, CoinTypeB, T>(
    manager: &VaultsManager,
    vault: &mut Vault<T>,
    [...]
    tick_lower: u32,
    tick_upper: u32,
    [...]
) {
    checked_package_version(manager);
    check_rebalance_role(manager, tx_context::sender(ctx));
    assert!(!vault.is_pause, EPoolIsPause);
    assert!(vault.status == STATUS_RUNNING, EVaultNotRunning);
    assert!(vector::length(&vault.positions) == 1, EPositionSizeError);

    // change status to `STATUS_REBALANCING`; After add remaining tokens into position will
    set the status to `STATUS_RUNNING`.
    vault.status = STATUS_REBALANCING;
    [...]
}
```

Expanding the tick range significantly may result in a scenario where the migrated liquidity results in zero liquidity. This occurs because the widened tick range may fail to effectively capture any trading or liquidity provision opportunities. Zero liquidity implies no assets are available for trading or providing liquidity in the pool. As a result, interacting with the vault or the liquidity pool becomes impossible or highly inefficient, rendering the vault unusable.

Remediation

Implement checks and limits to prevent excessive expansion beyond reasonable bounds.

Patch

Fixed in fb1586a.

Flash Loan Exploitation LOW OS-CPV-ADV-02

Description

The vulnerability stems from the interaction between `flash_loan` and `repay_flash_loan` within `vaults`, specifically in how the `repay_amount` is calculated and adjusted for slippage. Initially, the `repay_amount` is determined based on the original flash loan amount, with adjustments made for slippage. Consequently, if the operator borrows funds repeatedly to repay the previous loan, they may repay a reduced amount each time due to the slippage adjustment.

>_ *sources/vaults.move* rust

```rust
public fun flash_loan<CoinTypeA, CoinTypeB, T>(
    manager: &VaultsManager,
    vault: &mut Vault<T>,
    oracle_pool: &Pool<CoinTypeA, CoinTypeB>,
    amount: u64,
    loan_coin_a: bool,
    ctx: &mut TxContext,
): (Coin<CoinTypeA>, Coin<CoinTypeB>, FlashLoanReceipt) {
    [...]
    let repay_amount = repay_amount - repay_amount * (oracle_info.slippage as u64) /
        ↪  SLIPPAGE_DENOMINATOR;
    [...]
}
```

As the operator may borrow funds repeatedly, they may borrow the necessary amount to repay the previous loan. This establishes a loop where the operator continually borrows and repays loans, essentially siphoning funds from the protocol without contributing any external value. This presents a substantial risk to the protocol's financial integrity. Permitting operators to borrow and repay loans with reduced repayment amounts repeatedly will deplete the protocol's reserves over time.

Remediation

Ensure that loans are limited to only one coin type at a time to reduce complexity and potential abuse. By restricting loans to a single coin type, the protocol will have better control over the flow of funds and mitigate the risk of manipulative behavior.

Patch

Fixed in b7c15c9.

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-CPV-SUG-00	The multiplication and division operations order in `get_tvl_of_based_coin` results in inaccuracies due to rounding down the final result multiple times.
OS-CPV-SUG-01	`take_protocol_assets` aborts if the type name is not found, whereas `harvest_assets_amount` returns zero in the same scenario.
OS-CPV-SUG-02	Recommendations for modifying the code base to enhance its security and efficiency.
OS-CPV-SUG-03	Suggestions regarding ensuring adherence to best coding practices.
OS-CPV-SUG-04	The setters and `create_vault` do not verify the values of `max_quota` and `finish_rebalance_threshold`, allowing the pool manager to set arbitrary values.

Rounding Error OS-CPV-SUG-00

Description

In `Vaults::get_tvl_of_based_coin`, the current order of execution of multiplication and division operations may result in inaccurate final results.

```rust
>_ sources/vaults.move                                                    rust

fun get_tvl_of_based_coin(sqrt_price: u128, amount_a: u64, amount_b: u64, base_a: bool): u128 {
    let sqrt_price_after_rs = (sqrt_price as u256) * PRICE_MULTIPER / UINT64_MAX;
    let price = sqrt_price_after_rs * sqrt_price_after_rs / PRICE_MULTIPER;
    let amount = if (base_a) {
        let converted_a = (amount_b as u256) * PRICE_MULTIPER / price;
        (amount_a as u128) + (converted_a as u128)
    }else {
        let converted_b = (amount_a as u256) * price / PRICE_MULTIPER;
        (amount_b as u128) + (converted_b as u128)
    };
    amount
}
```

The function begins by computing `sqrt_price_after_rs`, achieved through multiplying `sqrt_price` by `PRICE_MULTIPER` and then dividing by `UINT64_MAX`. Subsequently, it determines `price` by squaring `sqrt_price_after_rs` and dividing by `PRICE_MULTIPER`. Each multiplication and division operation may introduce rounding errors, particularly in integer arithmetic. These errors aggregate with multiple operations, possibly resulting in substantial deviations between the anticipated and realized outcomes.

Remediation

Rearrange the order of multiplication and division operations to minimize rounding errors.

Inconsistent Approach To Error Handling OS-CPV-SUG-01

Description

In `Vaults`, `take_protocol_assets` and `harvest_assets_amount` handle non-existent type names differently. `take_protocol_assets` employs an assertion to halt execution promptly with an error message (`EInvalidCoinType`), while `harvest_assets_amount` returns zero, indicating the absence of assets available for harvesting of the specified `CoinType`.

Remediation

Adjust one function's implementation to align with the other to maintain consistency in the code base.

Code Refactoring

OS-CPV-SUG-02

Description

1. `vaults::init` fails to utilize the `VERSION` constant when initializing the `VaultsManager`. The `VERSION` constant denotes the package version. Without referencing this version during the creation of `VaultsManager`, there exists a risk of inconsistency between the version indicated in the code and the actual version employed in the initialization process.

>_ *sources/vaults.move* rust

```rust
fun init(ctx: &mut TxContext) {
    [...]
    let manager = VaultsManager {
        id: object::new(ctx),
        index: 0,
        package_version: 1,
        vault_to_pool_maps: table::new(ctx),
        price_oracles: table::new(ctx),
        acl: acl::new(ctx)
    };
    [...]
}
```

2. The current `docstring` for `vaults::add_oracle_pool` lacks information regarding the `slippage` parameter, essential for adding an oracle pool. This omission may confuse developers or users who depend on the documentation.

Remediation

1. Revise `vaults::init` to utilize the `VERSION` constant when creating the `VaultsManager` object.
2. Update the `docstring` to accurately reflect the current implementation of the function, including details about the `slippage` parameter.

Code Maturity

OS-CPV-SUG-03

Description

1. In `vaults::deposit`, substitute the literal value (`18446744073709551615u128`) with the named constant (`UINT64_MAX`) to improve code readability.
2. Both `acl::remove_role` and `acl::remove_member` fail to raise errors if the specified `member` does not exist; instead, they silently return. As a result, `vaults::remove_role` and `vaults::remove_member` emit an event even if no such action was taken.
3. Within `vaults`, `threshold` is misspelled as `thresold` in multiple instances.

>_ *sources/vaults.move* rust

```rust
/// Emit when update finish_rebalance_thresold
struct UpdateRebalanceThresoldEvent has copy, drop {
    vault_id: ID,
    old_thresold: u64,
    new_thresold: u64
}
```

Remediation

Implement the above-mentioned suggestions.

Absence Of Parameter Validation OS-CPV-SUG-04

Description

The vulnerability stems from the absence of validation for the `max_quota` and `finish_rebalance_threshold` values in their setter functions or in `create_vault` within `vault`. Without proper validation, the pool manager may assign arbitrary and potentially unintended values to `max_quota` and `finish_rebalance_threshold`.

>_ *sources/vaults.move* rust

```rust
public fun create_vault<CoinTypeA, CoinTypeB, T>(
     [...]
     max_quota: u128,
     finish_rebalance_threshold: u64,
     clk: &Clock,
     ctx: &mut TxContext
) {
    checked_package_version(manager);
    check_pool_manager_role(manager, tx_context::sender(ctx));
    assert!(total_supply<T>(&lp_token_treasury) == 0, ETreausyCapIllegal);
    [...]
}
```

Assigning arbitrary values to these parameters may result in instability within the system. A high `max_quota` may cause unexpected resource allocation or imbalances in the vaults system. Similarly, setting inappropriate values for `finish_rebalance_threshold` may disrupt the re-balancing process and impact the system's overall performance.

Remediation

Implement validation checks for `max_quota` and `finish_rebalance_threshold`.

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.

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.

INFO

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.