Eternal Finance

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen r@osec.io

Akash Gurugunti sud0u53r.ak@osec.io

Contents

01 | Executive Summary

Overview

Eternal Finance engaged OtterSec to perform an assessment of the `eternal-contracts` program. This assessment was conducted between January 9th and January 20th, 2023. For more information on our auditing methodology, see Appendix B.

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 to streamline patches and confirm remediation. We delivered final confirmation of the patches **[not yet delivered]**.

Key Findings

Over the course of this audit engagement, we produced 9 findings total.

In particular, we identified numerous loss of funds issues including an inconsistency in position implementation (OS-ETN-ADV-00), a lack of asset manipulation checks (OS-ETN-ADV-02), and improper proper access control on internal functions (OS-ETN-ADV-01).

We also made recommendations around unnecessary mutable borrows (OS-ETN-SUG-00), unused bank information fields (OS-ETN-SUG-01), and an incorrect bitmap constant (OS-ETN-SUG-02).

Overall, we appreciate the Eternal Finance team's responsiveness throughout our engagement.

02 | Scope

The source code was delivered to us in a git repository at github.com/eternalfinanceio/eternal-contracts. This audit was performed against commit `5bc32ce`.

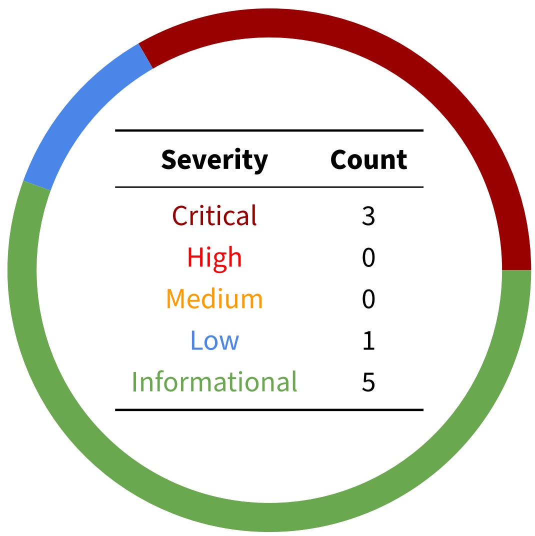
A brief description of the programs is as follows.

Name	Description
eternal-contracts	`eternal-contracts` implements leveraged yield farming using PancakeSwap protocol. • `bank` maintains vaults and user positions. • `vault` maintains user deposited collateral and withdrawn debts. Interest is accrued over time and added to debt value. • `pancake_dex_worker` contains entry functions for the user to create and reduce positions. It also enables some whitelisted users to reinvest rewards and liquidate positions. • `pancake_wmasterchef` contains functions used by worker module to deposit lp coins into pancake masterchef and earn cake rewards. It also contains functions to harvest rewards and stake it to gain extra rewards.

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 don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	3
High	0
Medium	0
Low	1
Informational	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-ETN-ADV-00	Critical	Resolved	Improper implementation of positions leads to inconsistency in `coll_token.share`.
OS-ETN-ADV-01	Critical	Resolved	Improper access control checks lead to loss of funds.
OS-ETN-ADV-02	Critical	Resolved	Improper slippage checks in swaps allow for the theft of reinvested assets.
OS-ETN-ADV-03	Low	Resolved	Improper setting and usage of paths lead to the failure of cake rewards reinvestment.

OS-ETN-ADV-00 [crit] | Improper Implementation Of Positions

Description

In the bank.move module, the Position struct is utilized to store collateral share amounts and debts on a position. However, there exists an inconsistency in the implementation of positions between the bank.move and pancake_dex_worker.move modules. The bank module assumes that a position can manage the collateral and debts of multiple coins, whereas the worker module assumes that a position can only hold the collateral and debts for one coin pair.

bank.move RUST

```rust
struct Position<CollToken> has store {
    coll_token: Option<CollToken>, // Stores only one type of collateral
    debt_bitmap: u128, // Handles multiple collaterals
    debt_share_of: SimpleMap<String, u64>, // Handles multiple
    ↪  collaterals
    user: address,
    pos_id: u64,
}
```

This is inconsistent because if a position is expected to manage the collateral of multiple coin pairs, the position struct in the bank should have a map of CollToken structs with coin name as key instead of storing just one CollToken field. Alternatively, if it is expected to handle only one coin pair, then the create_position, reduce_position and liquidate functions should have checks to validate whether the coins that have passed in the generics are actually associated with the position.

This inconsistency could result in the loss of funds. For example, adding to an existing position with a different coin pair would increase the position share, regardless of the initial coin pair. Additionally, the health checks assume that the debts vector returned from the bank::get_position_debts function has a length of two, which can result in improper health checks and under-collateralized loans.

Remediation

If a position is expected to handle multiple coin pairs, it is recommended to add a map of CollTokens to the Position struct and update the CollTokens accordingly. If the intended behaviour of a position is to handle only one coin pair, it is recommended to validate that the coin pairs passed in the generics align with the coin type of the LP shares stored in the position.coll_token.

Patch

Fixed by adding type validation in 6afe287.

OS-ETN-ADV-01 [crit] | Critical Access Control Check

Description

In `common_config.move`, the `resource_signer` function is utilized to obtain the signer from the signer capability that is stored in the resource based on the provided seed. This function is employed by other modules to generate, save, and retrieve resource accounts. Since the created resource account is used to store tokens in other modules, it is crucial that only the protocol modules can access this function.

common_config.move RUST

```rust
public fun resource_signer(seed: vector<u8>): signer acquires
    ↪  ResourceOwnershipCapStore{
    assert!(resource_acc_exists_at(seed), ERROR_SEED);
    let resources =
    ↪  &borrow_global<ResourceOwnershipCapStore>(@LYF).resources;
    account::create_signer_with_capability(&table::borrow(resources,
    ↪  seed).signer_cap)
}
```

To prevent unauthorized access to the `resource_signer` function while protecting the funds in the `wmasterchef` module, it is recommended to restrict the access of this function to only the protocol modules.

Remediation

This issue can be fixed by changing the access level of the function to `friend` and providing access to the function only to the trusted protocol modules.

Patch

Fixed in e69fea5.

OS-ETN-ADV-02 [crit] | Missing Slippage Checks

Description

`get_lp_by_cake` does not properly perform slippage checks against an oracle price when swapping assets around. This mainly occurs in `swap_exact_x_to_y_direct_external` and `add_liquidity_from_1_token`.

pancake_dex_worker.move RUST

```rust
    if( path == 0) {
        let coin_x_swap = router::swap_exact_x_to_y_direct_external<Cake,
    ↪   X>(cake);
        coin::merge(&mut coin_x, coin_x_swap);
        coin::merge(&mut coin_x, coin::withdraw<X>(resource_signer,
    ↪   get_my_balance<X>()));
    } else if (path == 1){
        // ...
    // 5. add liquidity from only coin x or coin y
    add_liquidity_from_1_token<X, Y>(coin_x, coin_y)
```

`get_lp_by_cake` is used when reinvesting to the exchange earned cake for LP tokens.

pancake_dex_worker.move RUST

```rust
fun reinvest_sorted<X, Y>(cake: Coin<Cake>) {
    // 1.0.1 send cake to treasury?

    // 1.1 trade cake via reinvest path
    // ...
    }else{
        wmc::stake<LPToken<X, Y>>(dex_helper::get_lp_by_cake<X,
    ↪   Y>(cake));
    };
```

It is possible for an attacker to use flash loans to manipulate the underlying pool, causing the swap to execute at a poor price. The attacker would then be able to swap back through the pool, frontrunning the reinvestment swap. Because reinvesting is done in a permissionless manner, this could allow for the theft of any accrued cake.

Remediation

One potential solution would be to make reinvesting permissioned, similar to the design for liquidations. We note that unpermissioned liquidations are also similarly vulnerable to the manipulation of protocol health.

Another possible solution would be to enforce slippage requirements on swap, comparing the output with the expected output provided by an oracle. This model would be more decentralized and could be more sustainable in the long run.

Patch

Fixed in aebdeb3.

OS-ETN-ADV-03 [low] | Improper Path Validation And Usage

Description

In `pancake_dex_helper.move`, the `set_paths` function sets a path to be used when reinvesting Cake rewards, verifying whether the path exists or not. When `idx = 2`, the function checks whether the path `Cake -> AptosCoin -> X` exists or not. However, in the `get_lp_by_cake` function, when `path = 2`, it uses the `Cake -> AptosCoin -> Y` path to convert the Cake rewards to Y token, which is inconsistent with the validations in the `set_paths` function.

lyf/pancake_dex_helper.move RUST

```rust
    } else if (idx == 2) {
        assert!(swap::is_pair_created<Cake, MIDToken1>() ||
     ↪  swap::is_pair_created<MIDToken1, Cake>(), ERROR_WRONG_PATH);
        assert!(swap::is_pair_created<X, MIDToken1>() ||
     ↪  swap::is_pair_created<MIDToken1, X>(), ERROR_WRONG_PATH);
    }else {
```

lyf/pancake_dex_helper.move RUST

```rust
    } else if (path == 2){
        let coin_mid = router::swap_exact_x_to_y_direct_external<Cake,
     ↪  MIDToken1>(cake);
        let coin_y_swap =
     ↪  router::swap_exact_x_to_y_direct_external<MIDToken1,
     ↪  Y>(coin_mid);
        coin::merge(&mut coin_y, coin_y_swap);
        coin::merge(&mut coin_y, coin::withdraw<Y>(resource_signer,
     ↪  get_my_balance<Y>()));
    } else {
```

This inconsistency could potentially cause the `get_lp_by_cake` function to fail. When `path = 2`, the function attempts to convert Cake rewards from `Cake -> AptosCoin -> Y`, but this path may not exist.

Remediation

To address the inconsistency between the validations in the `set_paths` function and the `get_lp_by_cake` function, we recommend adding a `idx = 3` case to the `set_paths` function, where it checks if the path `Cake -> AptosCoin -> Y` exists or not. Then, in the `get_lp_by_cake` function, we suggest using the `path = 2` case to convert cake rewards in the `Cake -> AptosCoin -> X` path, and using the `path = 3` case to convert cake rewards in the `Cake -> AptosCoin -> Y` path.

Patch

Fixed in 797a45c.

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-ETN-SUG-00	Mutable borrows are used when immutable borrows could suffice.
OS-ETN-SUG-01	`BankInfo::is_listed` is unused in the current implementation but provides important functionality.
OS-ETN-SUG-02	An incorrect bitmap constant is used in `bank::repay`.
OS-ETN-SUG-03	Cake rewards that are collected in WMC can only be reinvested and cannot be withdrawn.
OS-ETN-SUG-04	Function check used by health check makes it impossible to close a position.

OS-ETN-SUG-00 | Overuse Of Mutable Borrows

Description

Throughout the codebase, many examples of resources are mutably borrowed when the values are not mutated. This makes it more difficult to reason about changes in data.

For example, in the `debt_share_to_val` function, no changes to the state are done. However, the vaults are borrowed mutably.

pancake_dex_worker.move RUST

```
/// @dev Return the Token debt value given the debt share. Be careful of
    ↪  unaccrued interests.
public fun debt_share_to_val(coin_name: String, debt_share: u64) :u64
    ↪  acquires Vaults {
    let vaults = borrow_global_mut<Vaults>(@LYF);
    let vault_info = simple_map::borrow_mut<String, VaultInfo>(&mut
    ↪  vaults.vaults, &coin_name);

    if (vault_info.vault_debt_share == 0) return debt_share; // When
    ↪  there's no share, 1 share = 1 val.
    safe_math::mul_div_round_up_u64(debt_share, vault_info.vault_debt_val,
    ↪  vault_info.vault_debt_share)
}
```

Remediation

Refactor the code to use `borrow_global` instead of `borrow_global_mut`.

OS-ETN-SUG-01 | Unused BankInfo Field

Description

The `is_listed` field on `BankInfo` determines whether or not borrowing is allowed from the bank.

bank.move RUST

```
    struct BankInfo has store {
        is_listed: bool, // whether borrowing is allowed.
        index: u8, // Reverse look up index for this bank.
        total_share: u64 // The total debt share count across all open
     ↪  positions.
    }
```

However, the current bank module provides no way to change it from the default value of the true set in `init_bank`.

bank.move RUST

```
/// @dev Add a new bank to the ecosystem.
public entry fun init_bank<CoinType>(acc: &signer) acquires MetaData {
    // ...

    // 1. add bank info
    let bank = BankInfo {
        is_listed: true,
        index: (index as u8),
        // total_debt: 0,
        total_share: 0
    };
```

Remediation

Create admin-gated functions to modify `BankInfo::is_listed`.

OS-ETN-SUG-02 | Inaccurate Bitmap Constant

Description

In `bank::repay`, 255 is used as the bitmap constant. However, this only works if the bank index is less than 8.

bank.move RUST

```rust
*debt_share_of =
    if(*debt_share_of > share_reduced) *debt_share_of - share_reduced
    else {
        *&mut pos.debt_bitmap = *&mut pos.debt_bitmap & (255 - (1 <<
    ↪   *&bank.index)); // remove from bit map
        0
    };
```

Remediation

Use `u128::MAX` instead of 255.

OS-ETN-SUG-03 | Unable To Withdraw Cake From WMC

Description

In the `wmasterchef.move` module, cake rewards are collected from the pancake protocol into the resource account for every deposit and withdraw. The cake rewards are harvested and reinvested again in the protocol using the `harvest_reward` and `stake` functions by the whitelisted users, but there is no way for the protocol to withdraw the cake rewards accumulated into the resource account.

Remediation

It is recommended to add a permissioned function that transfers the cake rewards collected into the resource account to the treasury account.

OS-ETN-SUG-04 | Unable To Close Position

Description

In the `pancake_dex_helper.move` module, the function `calculate_debt_bps` is used to calculate the debt ratio of a position. This is used for checking a position's health while depositing and withdrawing shares and during liquidation. However, the current implementation of this function returns the maximum possible value (10000) when the `lp_amount` parameter is zero. This creates an issue when attempting to close a position and makes the share balance zero, as the debt ratio would be at maximum regardless of the actual debts, causing the health check to fail.

pancake_dex_helper.move RUST

```rust
public fun calculate_debt_bps<X, Y>(
    lp_amount: u64,
    tokens: &vector<String>,
    debts: &vector<u64>,
): u64 {
    // if no more collateral, debt is 10000.
    if (lp_amount == 0) {
        return 10000
    };
```

Remediation

It is recommended to check if the debts are empty before returning 10000 when the `lp_amount` is zero.

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.