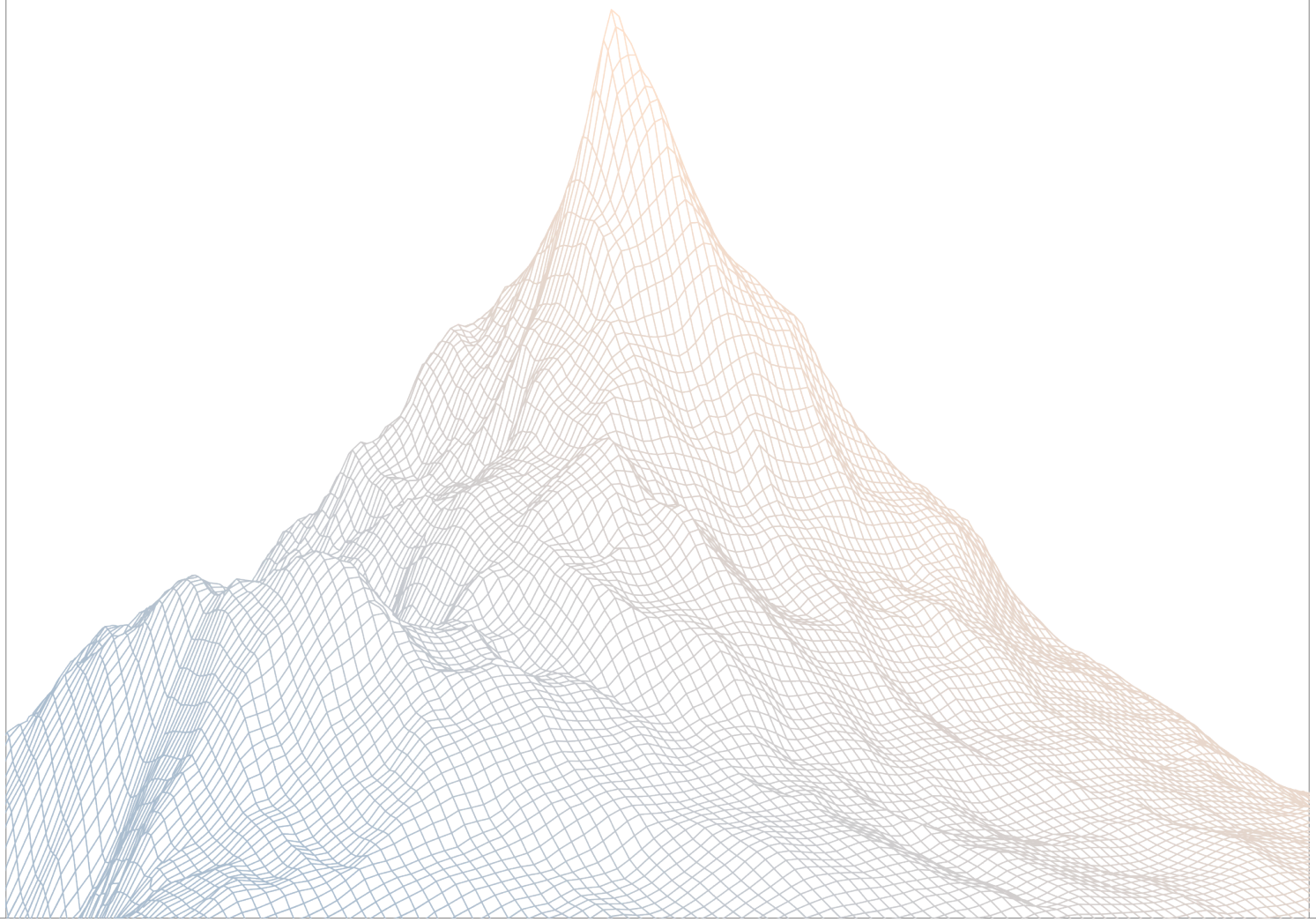
Zenith

Kofi Finance

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: March 3rd to March 10th, 2025

AUDITED BY: 1nc0gn170
jayfromthe13th

Contents

1 Introduction 2

1.1 About Zenith 3

1.2 Disclaimer 3

1.3 Risk Classification 3

2 Executive Summary 3

2.1 About Kofi Finance 4

2.2 Scope 4

2.3 Audit Timeline 5

2.4 Issues Found 5

3 Findings Summary 5

4 Findings 6

4.1 Critical Risk 7

4.2 Medium Risk 8

4.3 Low Risk 11

4.4 Informational 13

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Kofi Finance

Kofi is a next-generation liquid staking platform on Aptos, allowing Aptos users to earn boosted staking rewards through Kofi's dual liquid staking token model — kAPT & stkAPT.

Kofi offers boosted liquid staking, whereby Kofi's yield bearing staked asset provides staking yields as the base layer yield with additional boosted rewards accrued on top of it.

2.2 Scope

The engagement involved a review of the following targets:

Target	kofi-finance-contracts
Repository	https://github.com/wagmitt/kofi-finance-contracts
Commit Hash	`2a8929d22445a038919f579ffa04499ed8817436`
Files	`kofi-finance-contracts/modules/*`

2.3 Audit Timeline

March 3, 2025	Audit start
March 10, 2025	Audit end
March 17, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	1
High Risk	0
Medium Risk	2
Low Risk	1
Informational	2
Total Issues	**6**

3

Findings Summary

ID	Description	Status
C-1	Locked Fees in Vault Due to Missing Withdrawal Function	Resolved
M-1	Incorrect Metadata Used in deposit_stkapt Function	Resolved
M-2	Possible DOS Due to Inefficient Validator Looping	Acknowledged
L-1	Unbounded values can be set in config.	Resolved
I-1	Unused Code and Constants	Resolved
I-2	Incorrect Token Burn Amount in ensure_minimum_pending_inactive Function	Resolved

4

Findings

4.1 Critical Risk

A total of 1 critical risk findings were identified.

[C-1] Locked Fees in Vault Due to Missing Withdrawal Function

SEVERITY: Critical	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target

- vault.move#L448
- withdrawal_manager.move#L135

Description:

The protocol deposits withdrawal fees into the fees vault using `deposit_fees_apt()`, but there is no corresponding function to withdraw these fees. This creates a permanent lock of APT tokens in the fees vault, as there is no mechanism for protocol administrators to access these accumulated fees. The `withdrawal_manager` module deposits fees during withdrawal finalization, but these fees become permanently locked in the fees vault with no way to retrieve them.

Recommendations:

Implement a withdrawal function to withdraw fees .

Kofi: Resolved with @ea17636d210...

Zenith: Verified.

4.2 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] Incorrect Metadata Used in `deposit_stkapt` Function

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- vault.move#L237

Description:

In the `deposit_stkapt` function, the code incorrectly uses `kAPT_coin::metadata()` instead of `stkAPT_coin::metadata()` when creating/accessing the fungible store for `stkAPT` tokens. This mismatch between token type and metadata will cause deposits to fail. The function is supposed to deposit `stkAPT` tokens but is using kAPT metadata to identify the store.

Line of code:

```
public(friend) fun deposit_stkapt(fa: FungibleAsset)
    acquires VaultCapabilities {
    let vault_addr = get_stkapt_vault_address();
    let metadata = kAPT_coin::metadata();   //@audit issue
    let store =
        primary_fungible_store::ensure_primary_store_exists(vault_addr,
    metadata);
    fungible_asset::deposit(store, fa);
}
```

Recommendations:

Change the metadata reference to use the correct token type:

```
let metadata = stkAPT_coin::metadata();
```

Kofi: Resolved with @74a6eab8b1b...

Zenith: Verified.

[M-2] Possible DOS Due to Inefficient Validator Looping

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- delegation_manager.move#L82-L118
- delegation_manager.move#L125-L163
- delegation_manager.move#L167-L219
- delegation_manager.move#L303-L579

Description:

The system is designed to handle operations across multiple validators, each with a single associated pool. As the protocol scales to support thousands of validators, certain functions that loop through all validators to perform operations become potential bottlenecks. Functions such as delegate_apt, undelegate_apt, and withdraw_stake in the delegation_manager.move module iterate over all validators, leading to high computational resource usage. This inefficiency can result in increased execution time and gas costs, potentially causing transactions to fail due to exceeding gas limits or timeouts. The situation is exacerbated when these functions are automatically triggered at regular intervals or in response to specific events, leading to a potential DOS scenario where legitimate transactions are delayed or fail due to resource exhaustion.

Recommendations:

Consider using table-based lookups instead of iterating through lists, which can improve efficiency by allowing for faster key-based access. Implementing rate limiting can also help prevent abuse and manage resource consumption. If this implementation is kept, then at a bare minimum, it would be advisable to monitor the system, possibly perform validator cleanup, and consider implementing an additional pause mechanism as a safeguard when adding validators to the system.

Kofi: Acknowledged

4.3 Low Risk

A total of 1 low risk findings were identified.

[L-1] Unbounded values can be set in config.

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- config.move#L491-L500
- config.move#L437-L446
- staking_manager.move#L78-L107

Description:

- The `set_min_withdrawal_amount_admin` function allows setting the minimum withdrawal amount without any upper bound validation. This could lead to a denial of service if the admin accidentally sets an extremely high minimum withdrawal amount, effectively preventing users from making withdrawals.
- The `set_withdrawal_fee_admin` function allows setting the withdrawal fee without any upper bound validation. Unlike other fee-setting functions, which validate that the fee doesn't exceed DEFAULT_BASIS_POINTS, this function allows setting arbitrarily high fees. This could lead to users losing funds if the admin accidentally or maliciously sets an extremely high fee .
- The `set_withdrawal_period_admin` function only checks that the withdrawal period is greater than zero, but doesn't set an upper bound. This could lead to a denial of service if the admin sets an extremely long withdrawal period.
- The `set_management_fee_rate_admin` function ensures that the management fee rate is less than or equal to DEFAULT_BASIS_POINTS. However, it should also enforce a lower bound using something like DEFAULT_MANAGEMENT_FEE_RATE. If the management fee rate is set to zero by default accidentally or a very low value, it could lead to a division by zero when calculating rewards in the update_rewards function. This could result in invalid and zero amounts for rewards, affecting the intended reward distribution in the system.
- In the `set_min_stake_amount_admin` function, there is a check to ensure that the

minimum stake amount is at least the default value (DEFAULT_MIN_STAKE_AMOUNT). However, if this check is not enforced properly, it could lead to a misconfiguration. If the minimum stake amount is set to a value lower than the default, it may cause a dos in the staking function.

Recommendations:

Enforce upper & lower bound checks accordingly.

Kofi: Resolved with @19f6a33c860...

Zenith: Verified.

4.4 Informational

A total of 2 informational findings were identified.

[I-1] Unused Code and Constants

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- Config.move#L32
- Vault.move#L235
- Vault.move#L220
- Vault.move#L248
- Vault.move#L387
- Vault.move#L402
- delegation_manager.move#L248
- delegation_manager.move#L295
- Config.move#L30

Description:

Several functions and constants are defined but never used in the production code.

- DEFAULT_MAX_CLAIM_AMOUNT - Defined but never referenced
- deposit_stkapt - Function defined but never called
- withdraw_stkapt - Function defined but never called
- burn_kapt - Function defined but never called
- withdraw_buffer_kapt - Function defined but never used
- deposit_buffer_kapt - Function defined but never used
- ensure_minimum_pending_inactive - Function defined but never used
- DEFAULT_CLAIM_INTERVAL - Defined but never referenced

Recommendations:

Try removing unused code.

Kofi: Resolved with @7a2a6ce83e4...

Zenith: Verified.

[I-2] Incorrect Token Burn Amount in `ensure_minimum_pending_inactive` Function

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- delegation_manager.move

Description:

The `ensure_minimum_pending_inactive` function burns `min_pending_inactive` tokens instead of `amount_to_unlock` tokens. This creates an accounting mismatch between the amount of APT being unlocked from active stake (which is amount_to_unlock) and the amount of kAPT being burned.

Line of code:

```
public(friend) fun ensure_minimum_pending_inactive(
    admin: &signer, pool_address: address
) {
    let vault_address = vault::get_apt_vault_address();
    let vault_signer = vault::get_apt_vault_signer();
    let min_pending_inactive = config::get_min_pending_inactive();

    assert!(
        delegation_pool::delegation_pool_exists(pool_address),
        errors::no_delegation_pool()
    );

    let (active, _inactive, pending_inactive) =
        delegation_pool::get_stake(pool_address, vault_address);

    if (pending_inactive < min_pending_inactive) {
        let amount_to_unlock = min_pending_inactive - pending_inactive;
        assert!(active ≥ amount_to_unlock, errors::insufficient_stake());

        delegation_pool::unlock(&vault_signer, pool_address,
    amount_to_unlock);
```

```
        kAPT_coin::burn(admin, min_pending_inactive);   //@audit
    };

}
```

Recommendations:

```
public(friend) fun ensure_minimum_pending_inactive(
    admin: &signer, pool_address: address
) {
    let vault_address = vault::get_apt_vault_address();
    let vault_signer = vault::get_apt_vault_signer();
    let min_pending_inactive = config::get_min_pending_inactive();

    assert!(
        delegation_pool::delegation_pool_exists(pool_address),
        errors::no_delegation_pool()
    );

    let (active, _inactive, pending_inactive) =
        delegation_pool::get_stake(pool_address, vault_address);

    if (pending_inactive < min_pending_inactive) {
        let amount_to_unlock = min_pending_inactive - pending_inactive;
        assert!(active >= amount_to_unlock,
errors::insufficient_stake());

        delegation_pool::unlock(&vault_signer, pool_address,
amount_to_unlock);

        kAPT_coin::burn(admin, min_pending_inactive);
        kAPT_coin::burn(admin, amount_to_unlock);

    };

}
```

Kofi: Resolved by removing `ensure_minimum_amounts_from_buffer`

Zenith: Verified.