TORTUGA

Tortuga

Audit

Presented by:

OtterSec contact@osec.io

Harrison Green hgarrereyn@osec.io

Fineas Silaghi fedex@osec.io

Contents

01 | Executive Summary

Overview

Tortuga Finance engaged OtterSec to perform an assessment of the `liquid-staking-ottersec` program and TIP-1. The audit of the program was conducted between September 17th and October 7th, 2022. The evaluation of the proposal was conducted between June 1st and June 7th, 2023.

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 the final confirmation of the patches on October 16th, 2022 and June 20th, 2023.

Key Findings

Over the course of this audit engagement, we produced 11 findings in total.

In particular, we identified several logical inconsistencies affecting staking limits (OS-TOR-ADV-03, OS-TOR-PRO-01) and one impacting commission rates (OS-TOR-ADV-01).

We also made recommendations around formal verification and the use of capability resources and other cases where information follows a “chain of security” to improve security and prevent inaccurate verification (OS-TOR-SUG-02, OS-TOR-SUG-03).

02 | Scope

The source code of the program was delivered to us in a git repository at github.com/MoveLabsXYZ/liquid-staking-ottersec. This audit was performed against commit f2b49ac

The source code of the proposal was shared with us through a pull request at github.com/Tortuga-Finance/tortuga-protocol/pull/9. This audit was performed against commit 6f05232.

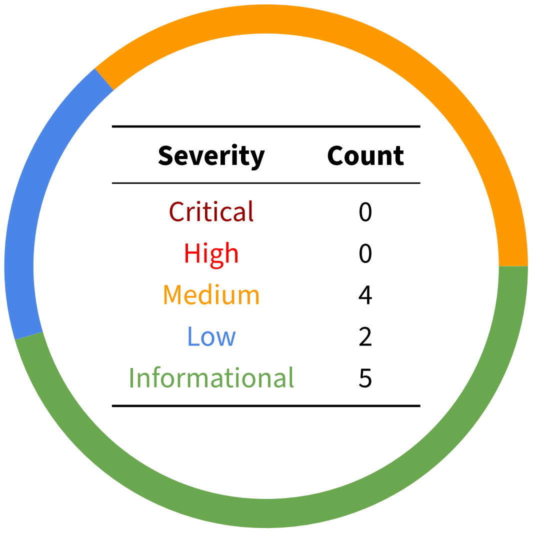
A brief description of the programs is as follows:

Name	Description
liquid-staking	Liquid staking protocol which allows users to optimally delegate APT to validators
TIP-1	The first Tortuga Improvement Proposal introduces support for the `delegation_pool` module and addresses several bugs and their fixes.

03 | Findings

Overall, we report 11 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	4
Low	2
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-TOR-ADV-00	Medium	Resolved	Users may bypass the minimum stake requirement in `delegation_service`.
OS-TOR-ADV-01	Medium	Resolved	Validators may manipulate their commission rate.
OS-TOR-ADV-02	Medium	Resolved	`unreserved_shares` pool in `delegation_service` may be filled with fake delegators in certain instances.
OS-TOR-ADV-03	Low	Resolved	Users may bypass the minimum stake amount limit by invoking `stake_coins` directly.

OS-TOR-ADV-00 [med] [resolved] | Minimum Stake Bypass

Description

The Tortuga protocol operates on top of the `delegation_service` module, which controls the pools and computes rewards for validators and delegators. While most users will delegate indirectly through Tortuga (`stake_router`), validators may also receive direct delegations through the `delegation_service` API.

Users who wish to delegate directly may invoke `delegation_service::delegate` and provide an amount. Internally, this function ensures that the amount provided meets a minimum delegation amount, which is configurable by the pool owner:

```move
fun certify_delegation(
    managed_pool_address: address,
    delegator_address: address,
    amount: u64
) ... {
    ...
    assert!(
        amount >= managed_stake_pool.min_delegation_amount,
        error::invalid_argument(EDELEGATION_AMOUNT_TOO_SMALL)
    );
    ...
}
```

While this check ensures that the instantaneous delegation amount is above the required minimum, withdrawals do not impose this limit. Therefore, a user may delegate some amount of stake higher than `min_delegation_amount`, then immediately withdraw a large portion to effectively bypass this limit.

Remediation

Impose a restriction on direct delegations such that delegators maintain a minimum delegation amount at all times or withdraw all their funds.

Patch

Fixed in af0d61f.

OS-TOR-ADV-01 [med] [resolved] | Validators Manipulating Commission Rates

Description

Registered validators may receive stake from the protocol or directly from individual delegators. Either way, paying a commission set by the validator is necessary. `protocol_commission` must be smaller than `current_commission`, and both must be less than or equal to `ManagedStakePool`'s `max_commission`, set by the protocol in `delegation_service::initialize`. Validators may drastically increase the commission percentage at any given time, which poses an issue.

RUST
```rust
public entry fun change_commission(
    pool_owner: &signer,
    new_default_commission: u64,
    new_protocol_commission: u64,
) acquires ManagedStakePool {
    [...]
    assert_pool_exists(managed_pool_address);

    [...]
    assert!(
        new_default_commission <= managed_stake_pool.max_commission,
        error::invalid_argument(ECOMMISSION_EXCEEDS_MAX)
    );
    // (Input Assert, keep)
    assert!(
        new_protocol_commission <= new_default_commission,
        error::invalid_argument(EPROTOCOL_COMMISSION_EXCEEDS_DEFAULT)
    );

    [...]
    delegation_state::change_commission_internal(
        managed_pool_address,
        new_default_commission,
        new_protocol_commission,
    );
}
```

This allows a malicious validator to set a very small commission and increase it by a large margin later on. Since 30-day lockup periods lock the stakes, the validator may profit from a large commission for a long time.

Remediation

Implement a similar lockup period for commission percentage as implemented for the stake; this avoids unexpected changes to the commission.

Patch

Fixed in 90cd93e.

OS-TOR-ADV-02 [med] [resolved] | Potential Denial Of Service In Pool

Description

In `delegation_service`, there is a hard limit on the number of direct delegators a pool may have: `MAX_NUMBER_OF_DELEGATIONS`, which currently equals 100.

A malicious user may fill the delegator list with fake delegators, staking small amounts to prevent real delegators from staking. In conjunction with OS-TOR-PRO-01, an attacker may bypass the `min_delegation_amount` and leave dust amounts in the pool, effectively making a free exploit.

Remediation

Ensure that all delegators wishing to stake may stake to maximize efficiency for the validators. A `min_delegation_amount` imposed for the lifetime of the delegator's funds helps ensure that all delegators are meaningfully contributing to the validator's total stake. Consider implementing a fix for OS-TOR-PRO-01 with this issue in mind.

Patch

Fixed in af0d61f.

OS-TOR-ADV-03 [low] [resolved] | Limit Bypass Through Stake Coins Invocation

Description

`stake_router` provides two entry points to stake coins:

- `stake_router::stake_coins` - A permissionless staking endpoint. Takes in `Coin<AptosCoin>` and returns `Coin<StakedAptosCoin>`.
- `stake_router::stake` - Wrapper over `stake_coins`. Withdraws coins from the signer, verifies a minimum deposit limit, invokes `stake_coins` and emits a `StakeEvent`.

Currently, there exists a minimum stake requirement imposed in `stake`:

tortuga/sources/stake_router.move MOVE

```move
public entry fun stake(
    delegator: &signer,
    amount: u64
) ... {
    ...
    assert!(
        amount >= staking_status.min_transaction_amount,
        error::invalid_argument(EAMOUNT_TOO_SMALL)
    );
    ...
}
```

However, users may invoke `stake_coins` directly to bypass this limit.

Patch

Fixed in 2b336d6 by making `stake_coins` private.

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-TOR-SUG-00	Validate signer against a hard-coded tortuga address in initialization routines.
OS-TOR-SUG-01	`get_total_value_locked` is unnecessary.
OS-TOR-SUG-02	Extract metadata from capability resources directly.
OS-TOR-SUG-03	Derive associated accounts directly instead of performing auxiliary checks.

OS-TOR-SUG-00 [resolved] | Validate Tortuga Address

Description

In `stake_router.move`, several initialization and setter methods take in `tortuga: &signer` as a function parameter. The lack of checks against this parameter may result in vulnerabilities.

tortuga/sources/stake_router.move MOVE

```
public entry fun set_reward_commission(
    tortuga: &signer,
    value: u64
) acquires StakingStatus {
    let staking_status =
    ↪  borrow_global_mut<StakingStatus>(signer::address_of(tortuga));
    staking_status.reward_commission = value;
}
```

Remediation

Include a check against the hard-coded `@tortuga` to ensure the addresses match.

MOVE

```
assert!(
    signer::address_of(tortuga) == @tortuga,
    error::unauthenticated(EPERMISSION_DENIED)
);
```

Patch

Resolved in 5ab8d72.

OS-TOR-SUG-01 [resolved] | Unnecessary Wrapper Function

Description

In `stake_router.move`, `get_total_value_locked` simply wraps `get_total_worth`, which is unnecessary.

tortuga/sources/stake_router.move MOVE

```
public fun get_total_value_locked(): u64 acquires StakingStatus {
    get_total_worth()
}
```

Remediation

Remove `get_total_value_locked`.

Patch

Resolved in ef89a88.

OS-TOR-SUG-02 [resolved] | Extract Metadata From Capabilities

Description

The Tortuga protocol uses several capability resources to provide access control to various functions. For example, `delegation_service` provides a `ManageCapability` to the protocol when registering new validators.

However, the current validation of capability resources occurs in a non-canonical way. For example:

delegation/sources/delegation_service.move MOVE

```
public fun pay_commission_to_owner(
    managed_pool_address: address,
    manage_cap: &ManageCapability,
): u64 {
    assert_pool_exists(managed_pool_address);
    assert!(manage_cap.managed_pool_address == managed_pool_address,
    ↪   error::unauthenticated(EPERMISSION_DENIED));
    delegation_state::pay_commission_to_owner(managed_pool_address)
}
```

In the snippet above, `manage_cap` contains a `managed_pool_address`, which it may operate on. Including this as a second argument and using an assertion to check equality is unnecessary. This pattern is dangerous as forgetting an assertion may lead to mishandling the capability (i.e. an attacker may be able to operate on the wrong pool).

Remediation

Derive any necessary data from the capability resource directly to simplify logic and ensure that the operation acts on the correct target for the given capability.

The code above could be refactored to:

delegation/sources/delegation_service.move MOVE

```
public fun pay_commission_to_owner(
    manage_cap: &ManageCapability,
): u64 {
    assert_pool_exists(manage_cap.managed_pool_address);
    delegation_state::pay_commission_to_owner(
        manage_cap.managed_pool_address
    )
}
```

Patch

Resolved in ef89a88.

OS-TOR-SUG-03 [resolved] | Simplify Associated Account Derivation

Description

Similar to OS-TOR-SUG-02, sometimes derivation of associated data from the given arguments is possible. In these cases, it is more canonical to use the derivation directly rather than requiring auxiliary arguments for the function, which are verified separately. For example:

delegation/sources/delegation_service.move MOVE

```move
public entry fun set_operator(
    pool_owner: &signer,
    stake_pool_address: address,
    new_operator: address
) acquires ManagedStakePool {
    let managed_pool_address = signer::address_of(pool_owner);
    // (Input Assert, keep)
    assert_pool_exists(managed_pool_address);
    // (Input Assert, keep)
    assert_is_stake_pool_owner(managed_pool_address, stake_pool_address);
    let managed_stake_pool = borrow_global<ManagedStakePool>(
        managed_pool_address
    );
    stake::set_operator_with_cap(
        &managed_stake_pool.stake_pool_owner_cap,
        new_operator
    );
}
```

In this example, `stake_pool_address` may be derived from the provided `managed_pool_address`; this is how `assert_is_stake_pool_owner` is implemented.

Remediation

Derive accounts whenever possible rather than using separate arguments and explicitly verifying conditions inside the function. In this case, omit the `stake_pool_address` argument and derive it from the `pool_owner`.

Patch

Resolved in ef89a88.

06 | TIP Findings

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-TOR-PRO-00	Medium	Resolved	The lack of commission rate sanitization allows public validators to set commission rates arbitrarily.
OS-TOR-PRO-01	Low	Resolved	Due to compatibility issues with the latest update, the incorrect resource capability is checked for public validators.

OS-TOR-PRO-00 [med] [resolved] | Missing Protocol Commission Check

Description

Public validators may become an associated validator by signing up through `validator_router::validator_signup`. While doing so, validators must provide the commission rates they charge both the Tortuga protocol and outside validators. However, there is no enforced commission limit, which means validators may charge more than the maximum, `max_commission`. `protocol_commission` not being sanitized during sign-up is the root issue.

Proof of Concept

The following example demonstrates how a public validator may sign up with a commission rate of 25%, even if `max_commission` is set to 20%.

```move
#[test(tortuga_governance_deployer = @tortuga_governance, pool_owner = @0x2,
    ↪  aptos_framework = @aptos_framework)]
#[expected_failure]
fun test_public_validator_cannot_signup_if_commission_too_high(
    pool_owner: signer,
    tortuga_governance_deployer: signer,
    aptos_framework: signer,
) acquires PublicValidators, Status, DelegationAccounts {
    account::create_account_for_test(signer::address_of(
    ↪  &tortuga_governance_deployer ));
    initialize_for_test(&tortuga_governance_deployer, 100);
    set_allow_self_signup(&tortuga_governance_deployer, true);
    set_max_max_commission(&tortuga_governance_deployer, 200000); // 20%

    // Set up stake pool for the owner
    // The validator commission percentage is 25
    test_helpers::test_configure_stake_pool_public_validator(&aptos_framework,
    ↪  &pool_owner, 0, 10000000, 0, true, true, 2500);
    let managed_pool_address = signer::address_of(&pool_owner);
    let stake_pool_address = delegation_pool::get_owned_pool_address(
        managed_pool_address
    );
    assert!(stake::get_validator_state(stake_pool_address) ==
    ↪  VALIDATOR_STATUS_ACTIVE, 0);

    // Validator signup should fail as current commission (25%) is greater than max
    ↪  commission (20%)
    validator_signup(&pool_owner, signer::address_of(&pool_owner), 500000, 250000);
}
```

Remediation

Ensure that the protocol commission is within the maximum limit.

```move
assert!(
    protocol_commission == 100 *
    ↪  delegation_pool::operator_commission_percentage(public_stake_pool_address),
    ECOMMISSION_MISMATCH
);

assert!(
    protocol_commission <= status.max_max_commission,
    error::invalid_argument(EMAX_COMMISSION_TOO_HIGH),
);
```

Patch

Fixed in 5369c8d.

OS-TOR-PRO-01 [low] [resolved] | Invalid Capability

Description

With the inclusion of the `delegation_pool` feature, several components require updates to facilitate its integration. However, `validator_router::set_withdraw_signature` and `validator_router::verify_withdraw_signature_for_validator` are missing the necessary modifications.

When public validators sign up, they must provide their `DelegationPoolOwnership` capability by invoking `delegation_pool::get_owned_pool_address`. However, the current version incorrectly assumes a `SharesData` capability as proof of owning a `stake_pool`, resulting in invalid validator sign-ups.

Proof of Concept

The following example demonstrates a scenario where a public validator's account is missing the resource:

```move
#[test(
    tortuga_governance_deployer = @tortuga_governance,
    pool_owner = @0x2,
    rando = @0x3,
    aptos_framework = @aptos_framework
)]
public entry fun test_set_withdraw_signature_public_validator(
    tortuga_governance_deployer: signer,
    pool_owner: signer,
    aptos_framework: signer,
) acquires Status, DelegationAccounts, WithdrawSignature, PublicValidators {
    init_environment_public_validator(&tortuga_governance_deployer, &pool_owner,
    ↪  &aptos_framework, true);
    validator_signup(
        &pool_owner,
        signer::address_of(&pool_owner),
        800000,
        0
    );
    let managed_pool_address = signer::address_of(&pool_owner);
    set_withdraw_signature(&tortuga_governance_deployer, managed_pool_address);

    let sig = borrow_global<WithdrawSignature>(
        @tortuga_governance
    );
    assert!(sig.managed_pool_address == managed_pool_address, 0);
    assert!(
        sig.unlocking_at ==
            stake::get_lockup_secs(
                delegation_pool::get_owned_pool_address(
                    managed_pool_address
                )
            ),
            0
        );
    }
```

Remediation

Introduce a helper function to replace `delegation_state::get_stake_pool_address` with support for the `delegation_pool` module.

```move
fun get_stake_pool_address(managed_pool_address: address): address acquires
    ↪  DelegationAccounts {
    if (is_an_associated_public_validator(managed_pool_address)) {
        delegation_pool::get_owned_pool_address(managed_pool_address)
    } else {
        delegation_state::get_stake_pool_address(managed_pool_address)
    }
}
```

Patch

Fixed in 5369c8d.

07 | 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-TOR-VER-00	Formalize data invariants.

OS-TOR-VER-00 | Data Invariant Specifications

Formal verification may be helpful for maintaining invariants about key structure properties.

stake_router.move RUST

```
    // Struct which holds the current state of the protocol, including the
    // storage for the protocol fee.
    struct StakingStatus has key {
        // Stores the protocol commission collected so far.
        protocol_fee: coin::Coin<StakedAptosCoin>,

        // The rate at which protocol charges commission.
        commission: u64,

        // When a user calls `unstake`,
        // This amount is taken from the user and
        // distributed among all the current users
        // of the protocol.
        community_rebate: u64,

        // Total balance of all the tickets issued since genesis.
        total_claims_balance: u128,

        // Total balance of all the redeemed tickets since genesis.
        total_claims_balance_cleared: u128,
```

For example, the `commission`, `cooldown_period`, and `community_rebate` fields in `StakingStatus` may be explicitly bounded through a data invariant.

stake_router.move RUST

```
spec StakingStatus {
    invariant cooldown_period <= MAX_COOLDOWN_PERIOD;
    invariant commission <= COMMISSION_NORMALIZER;
    invariant community_rebate <= MAX_COMMUNITY_REBATE;
}
```

Especially for fee parameters, formal verification allows for simple reasoning about invariants by examining the specification. For example, the fee will never be greater than 100%.

Apply a similar invariant to the `max_commission` field in `ManagedStakePool`.

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