JVault Staking Audit Report

Version V1

Audit for JVault Project - https://JVault.xyz/

October 3, 2024

JVault Staking Protocol Audit Report

Tonnel.Network

October 3, 2024

Prepared by: @tonnel

Lead Auditors:

- Mr.Freeman

Table of Contents

Protocol Summary

JVault Staking V2 is a service that allows any user on TON Ecosystem to create a staking pool for their token in a few clicks. The main distinguishing features of the platform are complete decentralization, a wide range of staking pool settings, and the presence of all the rational functions that a token founder may need.

Disclaimer

Mr.Freeman makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by me is not an endorsement of the underlying business or product. The audit was solely on the security aspects of the func implementation of the contracts not the business logic or user interface of the project.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

I use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

Scope

Commit Hash	Github Link	Date	Note
df09099d***b8f89280	Link	6, Sep 2024	Only the contracts folder

Executive Summary

Dear Ton Community,

Thank you for trusting *Mr.Freeman* to help **JVault** with this security audit. This executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of JVault Staking v2 according to Scope to support you in forming an opinion on their security risks level. JVault implements a novel customisable staking platform with sharded design which is fully compatible with TON architecture.

The most critical subjects covered in this audit are Functional Correctness, Access Control and Safety of Funds in pools. Based on our thorough checks, Security regarding all the aforementioned subjects is high. Please note that this audit only evaluate the security level of the mentioned scope, and any safety issue in the front-end or Mini-app of the product is out of scope of this Audit. Furthermore, the current implementation of the JVault Staking v2 contracts is partially upgradeable and the owner of the factory contract can upgrade the implementation of the pool factory which can lead to a potential risk for the pools that will be deployed after the upgrade(the pools that were deployed before the upgrade are not affected).

The general subjects covered are gas efficiency and error handling. Security regarding all the aforementioned subjects is high. In summary, we find that the codebase provides a high level of security and compatibility with TON sharded/async architecture. It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the issues uncovered and how they have been addressed or acknowledged. We are happy to receive questions and feedback to improve our service.

Sincerely yours, Mr.Freeman

Issues found

Overall, we found 8 issues in the JVault Staking V2 contracts. This section provides a brief overview of the issues found, their severity, and the affected areas.

ID	Description	Severity	Status
JVault-1	Wrong index size in the `pool_uninited_codes` dict	Medium	Fixed
JVault-2	Wrong send_mode in the `stake_wallet.fc`	Medium	Fixed
JVault-3	`set_data` called without proper checks in runtime	Low	Fixed
JVault-4	`set_code` called without proper checks in runtime	Low	Fixed
JVault-5	`storage::rewards_deposits_count` should be updated	Low	Fixed
JVault-6	`storage::pool_uninited_codes` could be empty	Info	Accepted by the JVault team
JVault-7	`rewards_deposits` dict were only updated locally	Gas	Fixed
JVault-8	`now()` function was called inside a nested for loop	Gas	Fixed

High

No high severity issues found!

Medium

JVault-1

JVault-1 Wrong index size in the `pool_uninited_codes` dict Description

In `pool_factory.fc`, when `op::update_uninited_code` is called, the transaction would cause issue in `pool_uninited_codes` dict. This function can be called only by the owner of the pool factory contract.

Impact

This issue can lead to a situation where the owner of the pool factory contract can not update `pool_uninited_codes` dict.

Proof of Concepts

Link to the code

```
if (op == op::update_uninited_code) {
    storage::pool_uninited_codes~udict_set_ref(128, storage::
        next_pool_id, in_msg_body~load_ref());
    save_data();
    return ();
}
```

Recommended mitigation The issue can be fixed by changing the index size in this function `pool_uninited_codes` dict to 32.

Response

The issue has been acknowledged by the JVault team and they fixed it in this commit.

JVault-2

JVault-2 wrong send_mode in the `stake_wallet.fc` (reported by the JVault team) Description

In `stake_wallet.fc`, when `op::receive_jettons` is called, in some cases the `send_mode` is set to `mode::carry_remaining_gas` which cause the contract to fail in action phase.

Impact

If the `storage::is_active` is false or the `storage::jetton_balance + received_jettons + storage::total_requested_jettons > storage::max_deposit` then the contract would fail in action phase and the user won't be able to receive their jettons back.

Proof of Concepts

Link to the code

```
if ((storage::jetton_balance + received_jettons + storage::
    total_requested_jettons > storage::max_deposit) | (~ storage::
    is_active)) {

    cell cancel_payload = begin_cell()
    .store_uint(op::cancel_stake, 32)
    .store_uint(query_id, 64)
    .store_slice(storage::owner_address)
    .store_uint(storage::lock_period, 32)
    .store_coins(received_jettons)
    .store_coins(deposit_commission)
    .end_cell();
    send_cell_message(sender_address, 0, cancel_payload, mode::
       carry_remaining_gas);

    return ();
}
```

Recommended Mitigation

change the `send_mode` to `mode::carry_remaining_balance` in this send_cell_message.

Response

The issue has been acknowledged by the JVault team and they fixed it in this commit.

Low

JVault-3

JVault-3 `set_data` called without proper checks in runtime Description

In `pool_factory.fc`, when `op::set_code` is called, the `set_data` could be called and in case of malformed input, it could lead to a situation where the `pool_factory` would be stuck.

Impact

This issue can lead to a situation where the `pool_factory` would be stuck which then the users won't be able to make a new pool.

Proof of Concepts

Link to the code

```
if (op == op::set_code) {
        set_code(in_msg_body~load_ref());
        if (in_msg_body.slice_refs()) {
            set_data(in_msg_body~load_ref());
        }
        return ();
    }
```

Recommended Mitigation

The risk can be mitigated by adding proper checks after the `set_data` function is called so that we ensure the new data layout is correct and `load_data` won't raise any errors.

Response

The issue has been acknowledged by the JVault team and they fixed it in this commit.

JVault-4

JVault-4 `set_code` called without proper checks in runtime Description

In `pool_factory.fc`, when `op::set_code` is called, the `set_code` would change the code of the contract and in case of a mistake in the new code, it could lead to a situation where the `pool_factory` would be stuck.

Impact

This issue can lead to a situation where the `pool_factory` would be stuck which then the users won't be able to make a new pool.

Proof of Concepts

Link to the code

```
if (op == op::set_code) {
        set_code(in_msg_body~load_ref());
        if (in_msg_body.slice_refs()) {
            set_data(in_msg_body~load_ref());
        }
        return ();
    }
```

Recommended Mitigation

TVM has `set_c3` function which can be used to set the code of the contract in runtime. Unlike `set_code` that is only take effect after the transaction, `set_c3` will take effect immediately(note: after execution of this primitive, the current code (and the stack of recursive function calls) won't change, but any other function call will use a function from the new code.), so we can use this op_code and then call `load_data` or other check functions to ensure the new code won't stuck in the next transaction call.

Response

The issue has been acknowledged by the JVault team and they fixed it in this commit.

JVault-5

JVault-5 `storage::rewards_deposits_count` should be updated after the `rewards_deposits` dict modification Description

In `staking_pool.fc`, when `op::request_update_rewards` is called by the stake wallet of user, the `rewards_deposits` dict would be modified and then the `storage::rewards_deposits_count` is not updated.

Impact

`storage::rewards_deposits_count` is only used to check the overflow of hashmap of `rewards_deposits` dict. So in case of reaching the limit of the hashmap, the pool owner won't be able to add new rewards to the pool.

Proof of Concepts

Link to the code

```
if (end_time <= time_now) {
    rewards_deposits~udict_delete?(32, index);
    time_now = end_time;
```

```
4 }
```

Recommended Mitigation

The issue can be fixed by decreasing the `storage::rewards_deposits_count` after an element is removed from `rewards_deposits`.

Response

The issue has been acknowledged by the JVault team and they fixed it in this commit.

Informational

JVault-6

JVault-6 - `storage::pool_uninited_codes` could be empty when the pool factory contract is deployed. Description

In `pool_factory.fc`, `storage::pool_uninited_codes` is not checked when the pool factory contract is deployed. This could lead to a situation where the `pool_uninited_codes` dict is empty and the users won't be able to create a new pool.

Impact

Although the risk and impact of this issue are low, it would be better to prevent this situation by checking the `pool_uninited_codes` dict when the pool factory contract is being deployed so that at least one code is available in the index 0.

Proof of Concepts

Link to the code

```
(_, cell pool_code, _) = storage::pool_uninited_codes.udict_get_max_ref
    ?(32);
```

Recommended Mitigation

The issue can be fixed by checking the `pool_uninited_codes` dict when the pool factory contract is being deployed.

Response

The risk was accepted by the JVault team.

Gas

JVault-7

JVault-7 - `rewards_deposits` dict were updated locally without storing it in the storage Description

In `staking_pool.fc`, `rewards_deposits` was updated locally without storing it in the storage. This would spend unnecessary gas and the changes would be lost after the transaction.

Impact

Gas usage and this update without storing it in the storage would cause confusion in the future.

Proof of Concepts

Link to the code

```
1 else {
2     rewards_deposits~udict_set_builder(32, index, begin_cell().
          store_coins(distribution_speed).store_uint(time_now, 32).
          store_uint(end_time, 32));
3 }
```

Recommended Mitigation

Clarify whether the update should be stored in the storage or not. If it should be stored, then the issue can be fixed by adding `save_data()` after the update otherwise the local update should be removed.

Response

The issue has been acknowledged by the JVault team and they fixed it in this commit.

JVault-8

JVault-8 - `now()` function was called inside a nested for loop Description

In `staking_pool.fc`, `now()` was called inside a nested for loop which would cause unnecessary gas usage.

Impact

Gas usage.

Proof of Concepts

Link to the code, Link to the code

```
while (success) {
    ***
    while (success2) {
        ***
        int time_now           = now();
        ***
    }
}
```

Recommended Mitigation

The issue can be fixed by calling `now()` once and storing it in a variable and then using that variable in the nested for loop.

Response

The issue has been acknowledged by the JVault team and they fixed it in this commit.