coinspect

You build, we defend.

Folks FINANCE

Smart Contract Audit

Liquid Staking

August 2024

Liquid Staking
Smart Contract Audit

Version: v240821 Prepared for: Folks Finance August 2024

Security Assessment

1. Executive Summary

In **August 2024**, **Folks Finance** engaged Coinspect to perform a Smart Contract Audit of the Folks Finance Liquid Staking contract. The objective of the project was to evaluate the security of the application.

The Liquid Staking contracts describe a system to provide tokens representing user stake in the upcoming Algorand staking system.

✓ Solved	⚠ Caution Advised	✗ Resolution Pending
High 0	High 0	High 0
Medium 1	Medium 1	Medium 0
Low 0	Low 0	Low 0
No Risk 1	No Risk 0	No Risk 0
Total **2**	Total **1**	Total **0**

Coinspect found way to bypass the limits on what an adversarial admin account can do, showing how it can it steal capital from users even when there are protections in place that try to stop this behavior. The issue is described in `FFLS-002`. Similarly, Coinspect showed on `FFSL-003` how an adversarial admin could leverage its power to arbitrarily raise fees. On the other hand `FFLS-001` describes the risk of insecure procedures to change the admin address which lead to loss of the rewards accrued.

2. Summary of Findings

This section provides a concise overview of all the findings in the report grouped by remediation status and sorted by estimated total risk.

2.1 Findings where caution is advised

Issues with risk in this list have been addressed to some extent but not fully mitigated. Any future changes to the codebase should be carefully evaluated to avoid exacerbating these issues or increasing their probability.

Findings with a risk of None pose no threat, but document an implicit assumption which must be taken into account. Once acknowledged, these are considered solved.

Id	Title	Risk
FFLS-001	Admin can be locked out of system	Medium

2.2 Solved issues & recommendations

These issues have been fully fixed or represent recommendations that could improve the long-term security posture of the project.

Id	Title	Risk
FFLS-002	Attacker with admin privileges can steal user capital	Medium
FFSL-003	Attacker with admin privileges can update fees to a 100% with no delay	None

3. Scope

The scope was set to be the repository at https://github.com/blockchain-italia/ff-coinspect-contracts at commit `ce7c9893a87de8408532dbdd4cddebc63ce5f9ac` files:

- `contracts/xalgo/consensus.py`
- `contracts/xalgo/consensus_state.py`
- `contracts/common/clear_program.py`

For the fixes review, the in-scope commit was set to `bfd11e7e9f734880c98d075242776d137f43059e`.

4. Assessment

The application in scope contains the core logic to implement a liquid staking program that will leverage rewards given by the upcoming changes to Algorand consensus. These changes are described in the Algorand Foundation paper Algorand Consensus Incentivisation. The most relevant change for this reviews is that the `FeeSink` account will
transfer funds to a block proposer, which are described in the go-algorand repository.

With this change, Algorand now has place for a liquid staking protocol built on on top of its consensus layer. Folks Finance solution implements such a protocol, where users can transfer `ALGO` to get `xALGO`, a token representing their stake on the network. Conversely, users might burn `xALGO` to get `ALGO` equivalent to their deposited stake and a portions of the rewards accrued minus some fee charged by the protocol manager to run the participating nodes, called `proposers` in the codebase.

The main user facing actions in the application are `immediate_mint()`, `delayed_mint()`, `claim_delayed_mint()` and `burn()`.

- `burn()` allows users to get `ALGO` for their `xALGO`
- `immediate_mint()` allows users to get `xALGO` for their `ALGO`, immediately, paying an admin-set *premium* as their capital cannot be utilized for 320 rounds.
- `delayed_mint()` and `claim_delayed_mint()` work in tandem, allowing users to get `xALGO` for their `ALGO` with a delay of 320 rounds, but avoiding the *premium*.

The protocol `admin` and `register_admin` have access to critical methods that *need to be called timely* for the protocol to work correctly:

1. `initialise()` needs to be called by the `admin` for the protocol to work
2. `schedule_update_sc()` can be called by the `admin` so as to announce and update.
3. `update_sc()` can be called by the `admin` after the `time_delay` has passed to apply the update commit to in `schedule_update_sc()`.
4. `rebalance_proposers()` needs to be called by the `admin` when the balance of any of the proposers are beyond the consensus maximum or below its minimum
5. `update_proposer_rebalance_range()` can be called by the `admin` to change the limits used in `rebalance_proposers()`.
6. `update_fee()` and `update_premium()` can be called by the `admin` to change the fee and premium charged to users, at any time
7. `claim_fee()` can be called at any time by the `admin` to claim the fees accumulated by a proposer

8. `pause_minting()` can be called at any time by the `admin` to pause and unpause the protocol

The second privileged account, `register_admin`, has fewer capabilities, as it only can call:

1. `add_proposer()`, to add a new proposer address to the system
2. `register_online()`, to send a `KeyRegistration` inner transaction for the proposer and mark it as `online` to the protocol
3. `register_offline()`, equivalent to `register_online()` but marking the proposer as `offline`.

One important aspect to emphasize is that the application does its best to be trustless. For example, `proposer` addresses need to rekey to the application account. This attempts to make it impossible, even for the admin, to use a fake `proposer` and the `rebalance_proposers()` method to steal capital from the users. Nevertheless, Coinspect found two ways in which an adversarial admin might steal money from users, described in `FFSL-002`.

4.1 Security assumptions

While performing this review, Coinspect assumed some properties and behavior of out-of-scope systems and processes. In particular, Coinspect considered that for the application to work correctly:

1. Proposers need to be online
2. `register_admin` needs to be able to detect proposers have been `unlucky` and call the Algorand's heartbeat
3. Algorand's consensus changes need to be implemented as described in their whitepaper
4. PyTEAL compiler and their `Router` need to work correctly
5. Protocol managers need to make necessary calls to keep the system healthy, such as `rebalance_proposer()` calls
6. The application needs to be correctly `initialized()` and exactly `10e15 xALGO` has to be minted to the application address.

Point `6.` is worth emphasizing: the correct procedure to deploy the application seems to be deploying the `governance.py` application and updating it to the `consensus.py` program, as the `consensus.py` program needs state that is set in `governance.py`. Because `governance.py` was out of scope for this review, Coinspect assumed that this process was carried out safely and correctly.

Coinspect considered attacks that need a privileged account to be less likely, but still part of the threat model.

4.2 Testing

The `consensus` application is generally well tested, with a ~70% of lines being covered by the testing suite.

Some tests are marked as `TODO` and should be implemented to improve the coverage, but critical functionality such as minting and burning is tested.

```
test("fails when proposer goes below min balance", async () => {
  // TODO
});
```

A possible point of improvement for the testing suite is to make the tests function as unit tests and not depend on the state of others tests in the suite. Currently, test modify the state of a common blockchain and depend on the order in which they are ran.

Coinspect was able to use the tests to probe the program for failures and unexpected behavior.

5. Detailed Findings

FFLS-001

Admin can be locked out of system

Status

Caution Advised

Risk

Medium

Impact

High

Likelihood

Low

Resolution

Acknowledged

Location

`ff-coinspect-contracts/contracts/xalgo/consensus.py`

Description

The `admin` and `proposer_admin` accounts can be locked out because the `update_admin` method does not require confirmation that the new `new_admin` is a controlled account.

This makes it possible for the current admin to make a mistake and send a non-controlled address (such as the zero-address) as the `new_admin` parameter.

While this requires a mistake in the update process, the consequences are dire, specially for the `admin` account: the proposer rewards would be lost and proposers would, in time, become more and more capital-inefficient due to the impossibility of rebalancing them.

Recommendation

Make the admin update a two step proposal, where a new address is proposed. Then, the proposed address must send a transaction confirming it accepts the new role.

Status

Acknowledged. Folks Finance stated that they already have an out-of-band two-step process as the `admin` is a multisig:

```
We already have a two step process. The admin account is a multisig
account that already requires multiple signers to review the
transaction and approve.
The long term plan is also to move away from using admin accounts and
replace them with a DAO.
```

FFLS-002

Attacker with admin privileges can steal user capital

Status
Solved

Resolution
Fixed

Risk
Medium

Impact
High

Likelihood
Low

Location

`ff-coinspect-contracts/contracts/xalgo/consensus.py`

Description

An attacker with `admin` privileges can steal user capital by adding the same proposer multiple times, artificially inflating the proposers balance and thus incrementing the rewards they are able to extract from the protocol.

To understand the issue, consider the `add_proposer` method, which takes only the address as a parameter and never checks whether the proposer already exists in the proposers box.

Once the same proposer appears at least twice in the list, the `get_proposers_algo_balance` will start to count the same balance multiple times, leading to an artificial increment in what the system believes the balance of the proposers is.

Because the system assumes all balance in excess of the capital deposited are rewards from consensus, a percentage of this artificially incremented rewards will be considered as fees. Now consider that `FFSL-003` allows an attacker to set the fee to 100%. This means that the attacker is able to steal capital from the users, and is only capped in the amount by the `MAX_NUM_PROPOSERS` value.

The exact steps are as follows:

1. Set themselves as the `proposer_admin` to be able to add proposers.
2. Increase the fee to 100% by abusing `FFSL-003`
3. Add a repeated proposer to the list
4. Send ALGO into the repeated proposed, so as to maximize the amount they are able as fees
5. Claim fees, which will be a 100% of the artificially inflated rewards

Note that sending `ALGO` to the proposer in this scenario has no cost for the attacker: all of the new ALGO will be considered rewards, and the fees are set at a 100%.

Consider that the issue may be triggered by a compromised admin account or even scripting issues. For example, if the script that adds a proposer sends the transaction twice due buggy retry.

Recommendation

Do not allow to add the same address as a proposer twice.

Status

Fixed in commit `bfd11e7e9f734880c98d075242776d137f43059e`. There is a new check in the `add_proposer` method which attempts to create a new `Box` with the proposer address. If the box already exists, an assert fails to be met and the whole transaction will revert.

A test for this specific scenario has also bee added.

Proof of concept

A proof of concept was shared to Folks Finance team.

FFSL-003

Attacker with admin privileges can update fees to a 100% with no delay

Status
Solved

✓

Resolution
Partially Fixed

Risk
None

▼
☐☐☐

Impact
Recommendation

Likelihood
-

Location

`ff-coinspect-contracts/contracts/xalgo/consensus.py`

Description

An attacker with `admin` privileges can update the critical `fee` value to 100% with no time delay and allowing all subsequent rewards to go to the admin via the `claim_fees` method.

All rewards accrued from that point on will go to the admin. Users would need to burn their `xALGO` as soon as possible to prevent the admin from stealing their rewards. This is also less likely to be noted by users as `update_fee` doest not emit any logs.

A similar problem can be found in the `update_premium` method. The impact in this case is lower because the premium is capped at 1% and users are able to set a `min_received` value which, if used correctly, can prevent the malicious admin from using the new, higher `premium`.

This issue is considered only informational because no capital can be directly stolen from users via this method and because it does not apply retroactively to already accrued rewards.

Recommendation

Set a time delay for `update_fee()` and `update_premium()`.

For further protection, consider adding adding logs to the `update_fee()` method and capping the maximum fee to less than `1e4`.

Status

Partially fixed in commit `bfd11e7e9f734880c98d075242776d137f43059e`. There are now logs that users can track to more easily detect when the fees change.

Folks Finance also stated that this issue can be prevented by users leveraging the `min_received` parameter in the `immediate_mint` scenario:

```
When you call "immediate_mint", one of the parameters is "min_received"
which allows you to protect against the scenario where the premium
changes between the time you submit the transaction and it
being confirmed on the blockchain. In regards to adding a delay, we
don't believe this is necessary because users can burn their xALGO at
any time if they aren't happy with the fee set. We have added
additional logs like you recommended to make it easier for users to
track these changes.
```

6. Disclaimer

The contents of this report are provided "as is" without warranty of any kind. Coinspect is not responsible for any consequences of using the information contained herein.

This report represents a point-in-time and time-boxed evaluation conducted within a specific timeframe and scope agreed upon with the client. The assessment's findings and recommendations are based on the information, source code, and systems access provided by the client during the review period.

The assessment's findings should not be considered an exhaustive list of all potential security issues. This report does not cover out-of-scope components that may interact with the analyzed system, nor does it assess the operational security of the organization that developed and deployed the system.

This report does not imply ongoing security monitoring or guaranteeing the current security status of the assessed system. Due to the dynamic nature of information security threats, new vulnerabilities may emerge after the assessment period.

This report should not be considered an endorsement or disapproval of any project or team. It does not provide investment advice and should not be used to make investment decisions.