SMART CONTRACT AUDIT

Certification

FARMS SMART CONTRACT

SMART CONTRACT AUDITS

February 21st, 2023 / v.0.2

Audited source code version:

06896e309b74f3823315d433ff4904c824e51445

Structure and Organization of the Document

Some sections are more important than others. The most critical areas are at the top, and the less critical sections are at the bottom. The issues in these sections have been fixed or addressed and will show by the "Resolved" or "Unresolved" tags. Each case is written so you can understand how serious it is, with an explanation of whether it is a risk of exploitation or unexpected behavior.

CRITICAL

These issues can have a dangerous effect on the ability of the contract to work correctly.

HIGH

These issues significantly affect the ability of the contract to work correctly.

MEDIUM

These issues affect the ability of the contract to operate correctly but do not hinder its behavior.

LOW

These issues have a minimal impact on the contract's ability to operate.

INFORMATIONAL

These issues do not impact the contract's ability to operate.

Issues

1. Mixing of funds

Unresolved / MEDIUM

Description: There is the general rule of thumb that each liquidity pool (either a swap pool or a staking pool or any other) must have its own smart contract instance. From a security perspective, it is dangerous to have all the funds from multiple pools mixed into a single instance because any problem that should arise might therefore affect all the funds and by extension all the users. Problems that might arise include: misconfigurations, bugs that include corner cases, security vulnerabilities.

! Possible fix to research

```
Refactor the current Farm Contract to be in fact a Farm Contract Deployer, and
for each staking pool, it deploy a new contract instance that handles just
that. This way, both the Farm Deployer Contract and the new Farm Contract code
will be much simplified, by splitting the functionalities between two contracts.
Also in the explorer, it would be much easier for users to see how many tokens
each pool has, and the activity within it. For this, see ‘deploy_from_source’
function inside MultiversX Wasm Rust Framework.
```

! Status

```
Not addressed.
```

2. Lack of testing

Unresolved / MEDIUM

Description: No tests were found.It is highly important for a smart contract that includes calculations (fees, rewards, shares) to be covered by at least a basic test suite. Those parts of the code that do the calculations are mandatory to be tested.

! Possible fix to research

```
Make a suite of tests and emphasise on ‘harvest’ and ‘calcHarvestableRewards’
and fee calculations in general.
```

! Status

```
Not addressed.
```

3. Unstaking does not claim rewards

Fixed / MEDIUM

Description: In MultiversX ecosystem, users usually expect that the unstaking everything action also claims the rewards that the staking position currently has. The problem should be addressed since not all the users read documentation or simply do not pay attention.

! Possible fix to research

```
Make the unstake function also claim the rewards, especially when fully
unstaking. This way, any action a user makes, will not result in him losing any
rewards he might have earned.
```

! Status

```
Claiming of rewards was introduced for full unstake scenario.
```

4. Sending zero value

Fixed / LOW

Description: There are cases in the contract where the **'send().direc()'** function is called but there's no check against the amount that will be transferred. For example, when the **'creator_charge'** is set to 1, the 'earning' for both **'Kostas'** and **'Eldar'** will be zero.

! Possible fix to research

```
Go through all the cases where a transfer is made and add checks against zero
before doing the call, or try adding checks that will enforce that such cases
never happen. For example setting the ‘creator_charge’ with at least value 1.
```

! Status

```
Accepted & Closed.
```

5. Hardcoded values

Unresolved / LOW

Description: The code contains a handful of hardcoded values that might not be intuitive and might be the source of errors in the future.

! Possible fix to research

```
Similar to how 'TOTAL_PERCENT' was made a constant and used throughout the
code, refactor the same for Pool Id 7 and the creator labels.
```

! Status

```
Not addressed.
```

6. Missing return values

Fixed / INFORMATIONAL

Description: It's important for endpoints to return values in some cases, for example in the case anyone wants to call the smart contract using another smart contract. If anyone does that and calls **'harvest'**, he wouldn't easily know how many rewards he actually got. Another example is for **'createFarm'**, one needs to know the **'farm_id'** associated with the farm in order to further interact with it.

! Possible fix to research

```
Add return values to endpoints.
```

! Status

```
Accepted & Closed.
```

7. Typos

Fixed / INFORMATIONAL

Description: Error messages contain typos. For example, search for **'Wrond'**.

! Possible fix to research

```
Revisit all error messages and fix the misspellings.
```

! Status

```
Accepted & Closed.
```

Verification Conditions

1 Owner functions are marked using either 'only_for_owners' at the beginning of the function body

```
self.only_for_owners();
```

2 The token type is verified against the expected one when staking into a farm

```
require!(farm.staked_token == token, “Wrond token staked for this farm.”);
```

3 The fee percent values are set within expected bounds

```
require!(early_unbonding_fee < TOTAL_PERCENT, “Wrong percentage. Fee must be
    below or equal to 10000 (=100%).”);
require!(rewards_fee < TOTAL_PERCENT, “Wrong percentage. Fee must be below or
    equal to 10000 (=100%).”);
```

Suggestions (Optional)

1. Write unit tests using Rust Testing Framework, so you can have the possibility of calling functions with dynamic values and test corner cases.

```
Status: Not addressed.
```

2. Use **'cargo fmt'** to format the code.

```
Status: Accepted & Closed.
```

3. Try splitting longer functions into many smaller functions.

```
Status: Not addressed.
```

4. Refactor hardcoded values.

```
Status: Not addressed.
```

5. Write interaction scripts. Use **'mxpy'** to automate transaction construction.

```
Status: Not addressed.
```

6. Improve the **'README.md'** by giving each public function a small description. A small description of the contract as a whole and a description of each fee would also be useful.

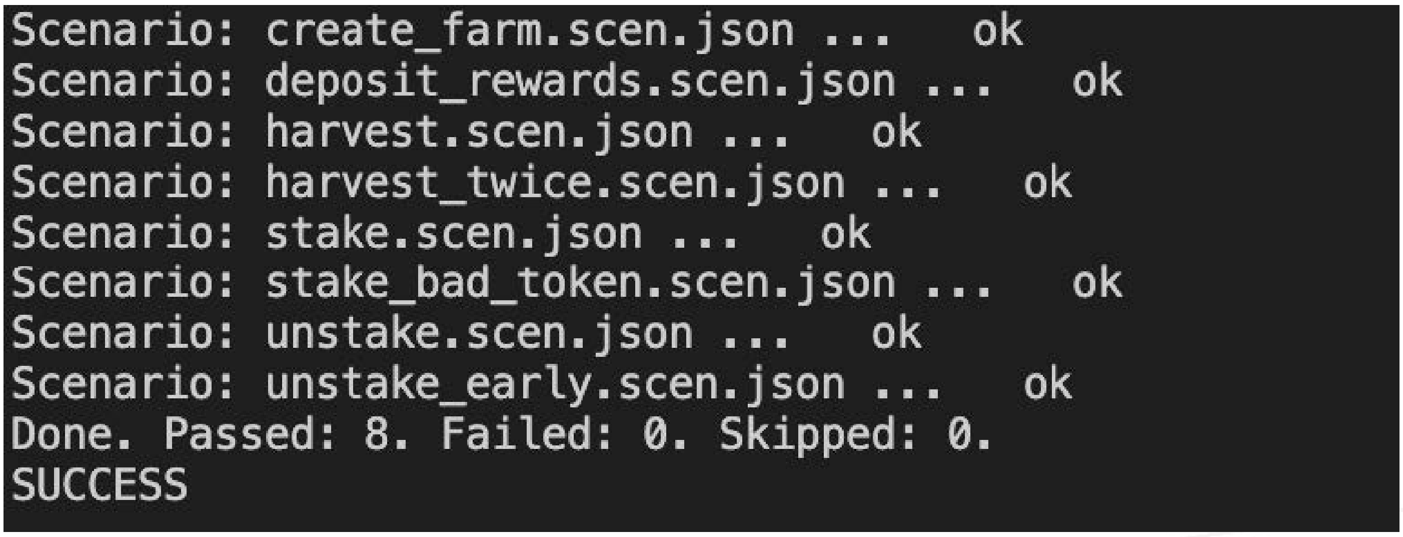
```
Status: Accepted & Closed.
```

7. Use **'init'** to make some of the initial configuration needed to **'becomeCreator'** and **'createFarm'**.

```
Status: Not addressed.
```

8. Pay attention to the destination shard when deploying the contract. It needs to be in the same shard with XOXNO contract.

Test results


```
Scenario: create_farm.scen.json ...   ok
Scenario: deposit_rewards.scen.json ...   ok
Scenario: harvest.scen.json ...   ok
Scenario: harvest_twice.scen.json ...   ok
Scenario: stake.scen.json ...   ok
Scenario: stake_bad_token.scen.json ...   ok
Scenario: unstake.scen.json ...   ok
Scenario: unstake_early.scen.json ...   ok
Done. Passed: 8. Failed: 0. Skipped: 0.
SUCCESS
```

Initial audited source code version
9cf75d2b0deafdb0bbd47cc172f5aad5fc0d20ac

* The audited source code version is the latest commit hash on the source code repo of SRB Farms SC.

Second version of audited source code
06896e309b74f3823315d433ff4904c824e51445

* This audited source code version is the latest commit hash of the shared repo