SMART CONTRACT AUDIT

Certification

STAKING CONTRACT

xaudits
SMART CONTRACT AUDITS
October 14th, 2022 / v. 0.2
Source code version: be97236bcd0346014eec5ea51b2d09141f066b3a

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.

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

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

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

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

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

Issues

1. Loss of funds

Not addressed / CRITICAL

Description: Let's imagine the following scenario: Two users stake 100 tokens. The APR is 100%. After the locking duration the owner does not want to (or he might not be able to) fund the contract with the necessary rewards. In this case, the first that will unstake the rewards will get his tokens back and also the reward. **The reward in this case will be the second user's initial stake**. The second user will lose his initial tokens and the promised reward.

! Possible fix to research

```
Keep users' funds separate from the rewards (in the same contract but different
counters). Avoid relying on balance, because it has a mixed group of funds
(initial user stake and the rewards).
```

! Response

```
Not addressed. Since the whole contract relies that the owner knows what he is
doing and he will send all the tokens at the right time.
```

! Status

```
Accepted & Closed. Pay great attention when adding new packages and handling
reward tokens.
```

2. Update total_tokens_staked when unstake

Not addressed / HIGH

Description: When someone unstakes, the **total_tokens_staked** global counter is not decreased.

! Possible fix to research

```
At the end of the unstake function, add an update() call on the storage and
decrease total_tokens_staked by staker_info.tokens_staked
```

! Response

```
Not addressed. It is intended that total_tokens_staked keeps track of all
tokens that were ever staked, regardless if they were unstaked or not.
```

! Status

```
Accepted & Closed. Keep in mind that if the contract is ever made public,
integrators may misunderstand the meaning of this storage, better document it well.
```

3. Update total_staked_amunt when unstake

Not addressed / HIGH

Description: When someone unstakes, the **total_staked_amunt** counter behind each package is not decreased.

! Possible fix to research

At the end of the **unstake()** function, add an **update()** call on the storage and decrease the **total_staked_amunt** by **staker_info.tokens_staked**

! Response

Not addressed. It is intended that total_staked_amunt keeps track of all tokens that were ever staked on this package, regardless if they were unstaked or not.

! Status

Accepted & Closed. Keep in mind that if the contract is ever made public, integrators may misunderstand the meaning of this storage, better document it well.

4. Check reward_frequency to be less than 365

Fixed / LOW

Description: If the value is set to higher than 365, the contract misbehaves, and the problem will result in a division by zero in **compute_rewards_per_cycle()** function.

! Possible fix to research

Add a check for this when a package is created.

! Response

Fixed.

! Status

Accepted & Closed.

5. Missing RemovePackage function

Not addressed / LOW

Description: If a package is misconfigured, it cannot be deleted or overwritten, and will not be able to reuse the package_name.

! Possible fix to research

```
Add an only_owner function that can remove packages that have zero total_
staked_amunt
```

! Response

```
Not addressed. Won't be needed since there are just a handful of 3-4 packages.
```

! Status

```
Accepted & Closed.
```

6. Package.txt misconfiguration

Unresolved / INFORMATIONAL

Description: The token amounts that are present in package.txt do not include the decimals.

! Possible fix to research

```
Update the numbers or, when using the numbers in a TX, don't forget to also add
the decimals.
```

! Response

```
Not addressed. The JS script that uses this file takes into account the
decimals to and adds them.
```

! Status

```
Accepted & Closed.
```

7. Update total_staked_amunt when reinvest

Not addressed / HIGH

Description: When someone reinvest, the **total_staked_amunt** counter behind every package should be increased.

! Possible fix to research

Add an **update()** call on the storage and increase the **total_staked_amunt** by **claimable_rewards**

! Response

Not addressed. It is intended that the total_staked_amunt keeps track of the tokens that the user originally invested, and not the ones that comes from the rewards.

! Status

Accepted & Closed.

8. Can reinvest without the SC having the tokens

Not resolved / HIGH

Description: The contract allows users to reinvest the claimable rewards without actually having the tokens.

! Possible fix to research

When calling **reinvestRewardsToExistingStake()**, check if the contract also has enough tokens before allowing that.

! Response

Not addressed. It is intended. When reinvesting, the reinvested amount comes from rewards and rewards generally do not have to be present in the contract until the end of the staking period.

! Status

Accepted & Closed.

9. Check lock_period value

Fixed / LOW

Description: The contract misbehaves when configured with lock_period value as 0.

! Possible fix to research

Add a zero value check to the **lock_period** parameter on **add_package()** function.

! Response

Fixed.

! Status

Accepted & Closed.

Verification Conditions

1 Integrity of User's Payment on Stake

```
let (payment_amount, payment_token) = self.call_value().payment_token_pair();
require!(
        payment_token == self.token_identifier().get(),
        “invalid staked token”
);
require!(
        payment_amount >= package_info.min_stake_amount,
        “stake amount too small”
);
```

2 Ownership of Staked Tokens

```
let staker_ids = self.staker_ids(&caller).get();
require!(staker_ids.contains(&id), “id is not defined for the staker”);
```

3 Unstake period bound

```
let locked_until = staker_info.stake_timestamp + package_info.lock_period * 86400;
require!(
        self.blockchain().get_block_timestamp() > locked_until,
        “tokens are under locking period”
);
```

Suggestions (Optional)

1. Allow APR to be in the range of (0, 100_000] instead of (0, 100] for more flexibility

```
Response: Done.

Status: Accepted & Closed.
```

2. Do not iterate to get an index (**.iter().position()**). Try using **SetMapper** instead

```
Response: Not addressed because of time trouble.

Status: Accepted & Closed.
```

3. Rather than calculating it everytime, put locked_until in the **StakerInfo** struct

```
Response: Done.

Status: Accepted & Closed.
```

4. Remove hardcoded values and have constants instead, eg. SECONDS_IN_DAY

```
Response: Done.

Status: Accepted & Closed.
```

5. Solve clippy warnings

```
Response: Done.

Status: Accepted & Closed.
```

6. Remove **set_if_empty** with default (0 like). It's already the default

```
Response: Not addressed. For better visibility, the set_if_empty calls
shall stay.

Status: Accepted & Closed.
```

7. Use **MapMapper** for **packageInfo** because it's iterable

```
Response: Not addressed. Not needed since there will be only a handful of
packages.

Status: Accepted & Closed.
```

8. Keep the rewards inside a separate storage

```
Response: Not addressed because of time trouble.

Status: Accepted & Closed.
```

9. Make the contract **non-payable**

```
Response: Not addressed because of time trouble.

Status: Accepted & Closed.
```

10. Make **enum** for PausedRewards eg. **NotPaused, Paused{Timestamp}**

```
Response: Changed such that paused_stake is just a bool.

Status: Accepted & Closed.
```

11. Make a **forceUnstake** function that does not also give rewards. It is good for emergency cases when the contract is not filled with tokens but an user urgently needs his staked tokens

```
Response: Not addressed. Defeats the purpose of the locking.

Status: Accepted & Closed.
```

12. Add events to endpoints

```
Response: Done.

Status: Accepted & Closed.
```

13. Convert existing test from Mandos to RustTestingFramework and add new automatic tests to improve coverage

```
Response: Not addressed due to time trouble.

Status: Accepted & Closed.
```

14. Remove **ManagedVec::new()**. When reading a **ManagedVec** from an empty storage, it'll give you a new and empty one.

```
Response: Done.

Status: Accepted & Closed.
```

15. The best practice is to always to the storage updates and then to do the transfers

```
Response: Done.

Status: Accepted & Closed.
```

16. Update framework version. Some of the function signatures have changed since the used version

```
Response: Not addressed due to time trouble.

Status: Accepted & Closed.
```

17. Fix typo **total_staked_amunt** -> **total_staked_amount**

```
Response: Done.

Status: Accepted & Closed.
```

18. Do not use the word **invest** in smart contracts, just to be sure that there's no possible legal problems. Instead, use compound

```
Response: Done.

Status: Accepted & Closed.
```

19. It'll be good to have also a check for **lock_period % rewards_frequency == 0** when adding a new package

```
Response: Done.

Status: Accepted & Closed.
```

20. Reinvest and Claim do not check for **package.enabled**. If that is intended just for new stakes, nothing should be done, otherwise, checks should be added on both functions.

```
Response: Not addressed. Intended behaviour.

Status: Accepted & Closed.
```

Test results

```
running 9 tests
test deploy ... ok
test create_new_stake ... ok
test add_package ... ok
test reinvest_rewards ... ok
test claim_rewards ... ok
test total_tokens_staked_increases_after_reinvest ... FAILED
test total_staked_amount_is_0_after_unstake ... FAILED
test unstake_too_early ... ok
test total_tokens_staked_is_0_after_unstake ... FAILED
```

After Fix & Feedback round:

```
running 9 tests
test deploy ... ok
test total_tokens_staked_is_same_after_reinvest ... ok
test claim_rewards ... ok
test add_package ... ok
test create_new_stake ... ok
test total_tokens_staked_is_same_after_unstake ... ok
test reinvest_rewards ... ok
test unstake_too_early ... ok
test total_staked_amount_is_same_after_unstake ... ok
```

Initially audited source code version: **be97236bcd0346014eec5ea51b2d09141f066b3a**

Afterwards, reviewed & audited **PR #16** that contains the fixes.