SMART CONTRACT AUDIT

Certification

XSTAKE CONTRACT

SMART CONTRACT AUDITS

Feb 23rd, 2024 / v.0.4

Audited source code version:

`29d4e1ea31c226b75b4b61c5268880c39e91c8d8`

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. Loss of funds

Not Applicable / CRITICAL

Description: The contract generates a new checkpoint (for a specific **'stake_id'**) every time a user calls **'stake'** or **'unstake'**. This could lead to the generation of a lot of checkpoints. This is a problem because in order to calculate the **'reward'** for a specific position, for example in **'claimRewards'** endpoint, all the checkpoints for that specific **'stake_id'** must be iterated. The iteration could easily fail due to too much gas being utilised, or too many reads from the trie being done. **'consolidateCheckpoints'** is not enough to fix the problem because: 1, the iterations inside **'consolidateCheckpoints'** can fail because of the same reasons, or 2, if a single user simply does not claim the rewards or does not unstake, it will prevent the past checkpoints from being cleared. There is no programmed limit to how much the checkpoint vector can grow and if it grows too much, users will not be able to claim the rewards or to unstake their position, hence the tokens will be locked inside the contract forever.

! Possible fix to research

The design of the reward calculation must be reconsidered, it is a dealbreaker, the contract will not work this way. A common way (for example in MvX and Evm ecosystems) to calculate the rewards in staking protocols is by holding an **'accumulator'** value, RPS (**'rewardPerShare'**).

! Response

Not Applicable.

! Status

Accepted & Closed (not applicable anymore for code v.0.2, starting with this version there is a new SC design using RPS accordingly with our suggestion)

2. Unchecked staked ratio

Not Applicable / MEDIUM

Description: There are no boundaries checks for the values of staked ratios.

! Possible fix to research

```
Add basic boundaries checks, like greater than 0 and maybe smaller than a
global maximum value.
```

! Response

```
Not Applicable.
```

! Status

```
Accepted & Closed (not applicable anymore for code v.0.2,  starting with this
version there is a new SC design using RPS accordingly with our suggestion)
```

3. Framework Version

Not Applicable / LOW

Description: The framework version that is used is **'0.42.3'** while the latest is **'0.47.0'**.

! Possible fix to research

```
Updating the framework can bring bug fixes and optimizations. It is generally
a good practice to use the latest version when deploying a new contract. The
testing framework version needs to be updated too, since a lot of functions that
are used in tests are marked as deprecated.
```

! Response

```
Not Applicable.
```

! Status

```
Accepted & Closed (not applicable anymore for code v.0.2,  starting with this
version there is a new SC design using RPS accordingly with our suggestion)
```

4. Empty error message

Not Applicable / LOW

Description: When staking, there's a check that verifies the EGLD amount received against an expected value, in case the check fails, the returned error message is empty.

! Possible fix to research

```
Add a useful message so the user gets a good idea of the reason should the TX fail.
```

! Response

```
Not Applicable.
```

! Status

```
Accepted & Closed (not applicable anymore for code v.0.2,  starting with this
version there is a new SC design using RPS accordingly with our suggestion)
```

5. Redundant checks

Not Applicable / LOW

Description: When staking and searching for the staking tokens against the payment tokens, the comparison **'i == j'** is redundant since **'j'** starts having the value of **'i+1'** and then incrementing.

! Possible fix to research

```
Remove that part of the condition in the 'require' statements.
```

! Response

```
Not Applicable.
```

! Status

```
Accepted & Closed (not applicable anymore for code v.0.2,  starting with this
version there is a new SC design using RPS accordingly with our suggestion)
```

6. Total unstake should claim rewards

Not Applicable / LOW

Description: When total unstaking, meaning unstaking the whole staked position, users usually expect the rewards also, in the same TX. Some users might forget that the rewards are still unclaimed after the total unstake.

! Possible fix to research

```
Add the claiming of rewards when total unstaking.
```

! Response

```
Not Applicable.
```

! Status

```
Accepted & Closed (not applicable anymore for code v.0.2,  starting with this
version there is a new SC design using RPS accordingly with our suggestion)
```

7. Gas optimization

Fixed / LOW

Description: The **'update_rps'** function tries to update the RPS even if the **'elapsed_nonces'** is zero. This calculus happens when, for example, multiple users claim their rewards at the same time (in the same block). The actual update of RPS will happen in the first TX and the following calls will try to increment the RPS with 0.

! Possible fix to research

```
Add a check for ‘elapsed_epochs > 0’ in the ‘if staked > 0’ statement.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

8. Redundant code

Fixed / LOW

Description: In **'claimRewards'** endpoint, the stake of the user is checked and if the value is zero, the rewards are sent and the storage is cleared. With the current design, the stake storage of the user cannot contain values where the stake amount is zero, it is either zero or when it reaches zero (upon unstake) it is cleared in place.

! Possible fix to research

```
Remove the handling of ‘else’ branch in the last ‘if’ statement of the
function body.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

Verification Conditions

1 User actions are guarded by 'active state checks on the contract'.

```
self.assert_active();
```

2 User stake action is guarded by 'active state checks of the stake'.

```
self.assert_stake_active(stake_id);
```

3 Valid payments are checked on input.

```
require!(found_tokens == payments.len(), ERROR_UNKNOWN_TOKEN);
```

4 Only the stake's owner can do configurations on the stake.

```
self.assert_stake_owner(stake_id);
```

5 Only the service address can call 'consolidateCheckpoints'.

```
require!(caller == self.service_address().get(), ERROR_NOT_AUTHORIZED);
```

Suggestions (Optional)

1. Token ordering. The contract has several places where payment tokens are searched against staked tokens. An idea would be to implement ordering. For example, when creating a new stake and configuring the staked tokens and the reward tokens, order them first alphabetically and store them in the storage this way. When receiving a payment or when unstaking, order the payments or parameters by token Id and this way, you can diminish the gas consumption by just knowing for example (when staking) that **'payment[i].tokenId == stakedTokens[i].tokenId'**. Code would be more clean, more readable, gas cost would be lower and the complexity diminished.

2. Refactor. Use clippy to find warnings, for example, replace **'refunds.len() > 0'** with **'!refunds.empty()'**.

```
Response: Fixed.

Status: Accepted & Closed.
```

3. Rename **'view_user_stake'** to **'user_stake'** since it is confusing when trying to write to the storage (eg. view_user_stake.set()). The name suggests it is read-only.

4. The tests have a lot of warnings, most of them due to the use of deprecated functions of the test frameworks. Update the testing files and fix the warnings.

Test results

```
     Running tests/test.rs (target/debug/deps/test-a8e6565f07d8ad1f)

running 8 tests
test init_test ... ok
test create_stake_test ... ok
test user_stake_test ... ok
test two_users_stake_test ... ok
test user_stakes_owner_changes_stake_duration_smaller_test ... ok
test two_users_stake_owner_adds_more_rewards_test ... ok
test user_stakes_owner_withdraw_some_rewards_user_claims ... ok
test user_stakes_stake_expires_owner_adds_more_rewards_and_prolonges_test ... ok

test result: ok. 8 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.01s
```

Audited source code version:
`9c8d4c1760846dae5540b9798ae308d5efa75121`

```
running 8 tests
test init_test ... ok
test create_stake_test ... ok
test user_stake_test ... ok
test user_stakes_owner_changes_stake_duration_smaller_test ... ok
test two_users_stake_test ... ok
test user_stakes_owner_withdraw_some_rewards_user_claims ... ok
test two_users_stake_owner_adds_more_rewards_test ... ok
test user_stakes_stake_expires_owner_adds_more_rewards_and_prolonges_test ... ok

test result: ok. 8 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.02s
```

Second review code version:
`29d4e1ea31c226b75b4b61c5268880c39e91c8d8`