SMART CONTRACT AUDIT

Certification

SFT BOOST STAKING CONTRACT

SMART CONTRACT AUDITS

Jan 30th, 2024 / v.0.2

Audited source code version:

ef8ad84cf1fa76ae1f94c39d723920333d548da1

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

Fixed / CRITICAL

Description: The endpoint made for configuring the contract parameters is public and callable by anyone.

! **Possible fix to research**

```
Add '#[only_owner]' on top of the 'setConfig' endpoint or do the configuration in
the 'init' method and remove the 'setConfig'.
```

! **Response**

```
Fixed.
```

! **Status**

```
Accepted & Closed
```

2. Loss of funds

Fixed / CRITICAL

Description: The **'stake'** endpoint accepts multiple payments but does not check for the length of the payments vector, except when staking for the first time, when it does check. It just takes into account the first payment and 'eats' all the following ones.

! **Possible fix to research**

```
In order to protect the user, make sure that the payment vector length is 1 for
the case where it's not the first 'stake' the user makes.
```

! **Response**

```
Fixed.
```

! **Status**

```
Accepted & Closed
```

3. Loss of funds

Fixed / CRITICAL

Description: When someone unstakes, the **'reward_deposit_amount'** is left unchanged. In case it is zero, a user that unstakes will receive rewards not from the **'reward_deposit'**, but from the contract balance aka other user's funds.

! Possible fix to research

```
Check ‘reward_deposit_amount’ against the reward amount in order to make sure
the action can be completed and update it accordingly (decrease it) when doing
a full unstake because the reward is sent too.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed
```

4. Lack of tests

Not applicable / HIGH

Description: There are no tests for this contract.

! Possible fix to research

```
Write tests in order to test the functionality of the contract.
```

! Response

```
Not applicable. Tests were done manually.
```

! Status

```
Accepted & Closed
```

5. Unstake zero amount

Fixed / LOW

Description: When someone is trying to unstake **'Some(0)'**, the contract will try to send **'0'** tokens to the caller. The call will fail but it might be harder to understand why, because the **'send'** function will fail.

! Possible fix to research

```
When receiving ‘Some(a)’ as a parameter of ‘unstake’, make sure the value of ‘a’
is greater than zero.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed
```

6. Anyone can deposit

Fixed / LOW

Description: The **'deposit'** endpoint is callable by anyone.

! Possible fix to research

```
There’s no fix because it’s not necessarily an issue but it might be a good idea
to make it ‘only_owner’ or at least to add a check against a whitelisted set of
addresses that are allowed to deposit.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed
```

Verification Conditions

1 Integrity of User's Payment on Stake

```
let caller = self.blockchain().get_caller();
let payments = self.call_value().all_esdt_transfers();
let one_payment = payments.get(0);
require!(
    one_payment.token_identifier == self.one_token_id().get(),
    "Invalid stake token id"
);
```

2 When fully unstaking, the user also gets the rewards

```
self.user_reward_amount(&caller).set(BigUint::zero());
self.send().direct_esdt(
    &caller,
    &self.one_token_id().get(),
    0,
    &(unstake_amount + &reward_amount),
);
```

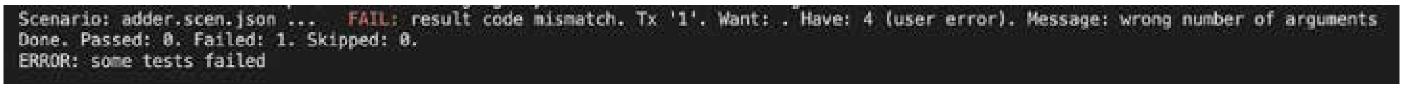
3 Claiming the rewards is possible only if there's enough tokens in the reward deposit

```
require!(
    self.reward_deposit_amount().get() >= reward_amount,
    "No enough deposit for reward"
);
```

Suggestions (Optional)

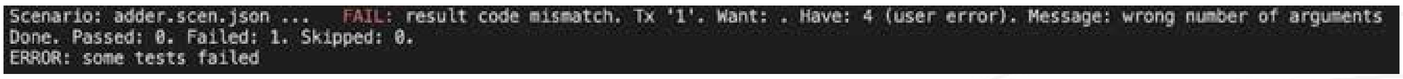
1. Recommendation is to write them in Rust Framework instead of Mandos.

Test results



```
Scenario: adder.scen.json ...   FAIL: result code mismatch. Tx '1'. Want: . Have: 4 (user error). Message: wrong number of arguments
Done. Passed: 0. Failed: 1. Skipped: 0.
ERROR: some tests failed
```

Initially audited source code version: c839067a0ffefa1f010b0a87631f6c4837a75e91



```
Scenario: adder.scen.json ...   FAIL: result code mismatch. Tx '1'. Want: . Have: 4 (user error). Message: wrong number of arguments
Done. Passed: 0. Failed: 1. Skipped: 0.
ERROR: some tests failed
```

Second review source code version: ef8ad84cf1fa76ae1f94c39d723920333d548da1