VESTA FINANCE

SMART CONTRACT AUDIT

Certification

LIQUID STAKING

BHero.

SMART CONTRACT AUDITS

February 2nd, 2023 / v. 0.2

Audited source code version:

c58ca91ec766a36ae410d3af7e30a337ed30882b

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. Users withdrawals are not in order

Fixed / MEDIUM

Description: Let's imagine a scenario where only two users have EGLD staked with equal stake amount. One calls **'unstake'**, the admin calls **'adminUndelegate'**, then the other user calls **'unstake'** too. If nothing happens in the next 10 days, the second user will be able to withdraw first, leaving the first user with the only option to wait for the admin to call **'adminUndelegate'** again and then for another 10 days to get back his EGLD.

! Possible fix to research

Implement an auto-unstake function and trigger it every time a user calls **‘unstake’**.

! Response

Auto-Delegate was removed too from the stake user action so now both delegate and undelegate are under the responsibility of the admin.

! Status

Accepted & Closed. Auto-Delegate was removed too from the stake user action so now both delegate and undelegate are under the responsibility of the admin.

2. Reserve gas for the async call

Fixed / MEDIUM

Description: Let's imagine the following scenario: The admin calls any function in the **'admin.rs'** file, for example **'adminUndelegate'**. The function executes successfully, the async call executes also successfully, that is the **'undelegate'** in the **'delegation_contract'**, but there's no gas left for executing the callback function, that is the **'admin_undelegate_callback'** function. If this happens, the storage will not be updated and various problems will happen.

! Possible fix to research

Use the **‘with_gas_limit(needed_gas)’** when building the **‘async_call’** and add a require before launching it. You can test how much gas the remote call costs and how much the callback costs and estimate a value for the **‘needed_gas’**.

! Response

Fixed in second review.

! Status

Accepted & Closed.

3. Withdraw function can run out of gas

Fixed / MEDIUM

Description: Let's imagine a scenario where, for some reason, a user has a lot of unstaking packs. He would not be able to withdraw them by any means, because there are two for loop iterations, and gas would run out.

! Possible fix to research

```
Make the for loops iterate on a maximum of X elements. This way, a user that
has too many unstaking packs would be able to withdraw his tokens by making
multiple withdraw calls.
```

! Response

```
Fixed in second review.
```

! Status

```
Accepted & Closed.
```

4. Async calls tracking

Fixed / MEDIUM

Description: Let's imagine the scenario where two async calls are launched at the same time and one fails, or one callback fails for some reason. The owner will not easily know which of them failed.

! Possible fix to research

```
When launching an async call, make it such that the callback function clears
the right async call id by passing it as parameter, instead of reading the last
async call id from the storage. Also, in the async call mapper, you could store
the type of async call one id has, be it ‘delegate’ or ‘undelegate’, for better
visibility and convenience.
```

! Response

```
Fixed in second review.
```

! Status

```
Accepted & Closed.
```

Verification Conditions

1 Owner functions are marked using either 'only_owner' macro attribute or 'require_is_owner_or_admin'

```
        self.require_is_owner_or_admin();
        self.require_admin_action_allowed();
```

2 Admin functions are marked using 'require_is_owner_or_admin'

3 When vEGLD is not zero, pool EGLD should never be zero invariant

```
require!(
         pool_egld_amount != BigUint::zero(),
         “staked_egld_amount is zero while staked_vegld_amount is not zero.”
);
```

4 Unbond EGLD only after the unbonding period

```
if current_timestamp >= item.timestamp + unbonding_period {
     unbonded_amount += &item.amount;
     unbonded_count += 1;
}
```

5 Unstake only using vEGLD

```
require!(
         payment_token == self.vegld_identifier().get_token_id(),
         “You sent wrong token.”
);
```

6 Avoid division by zero when quoting

```
require!(
         self.pool_egld_amount().get() != BigUint::zero(),
         “pool_egld_amount is zero”
);
```

Suggestions (Optional)

1. Write unit tests (either mandos or ideally in rust framework). Setting the roles for VEGLD ESDT token can be done with either **'setState'** step or the specialized **'set_esdt_local_roles'** rust framework function.

```
Response: No automated tests were written but TS scripts were implemented to
facilitate manual testing and deploying.

Status: Accepted & Closed.
```

2. Write delegation mock contract to facilitate integration testing.

```
Response: Not implemented since automated tests were not implemented.

Status: Accepted & Closed.
```

3. When calling owner or admin functions that require **'delegationAddress'** one might do it wrong so it would be better to remove that argument and place the address into the storage to avoid the need to specify it everytime.

```
Response: Fixed in second review.

Status: Accepted & Closed.
```

4. Remove the issue function from the code in order to make the resulting contract smaller, thus decreasing the cost of each transaction, and issue it by hand and transfer the ownership and the roles to the deployed contract.

```
Status: Not addressed.
```

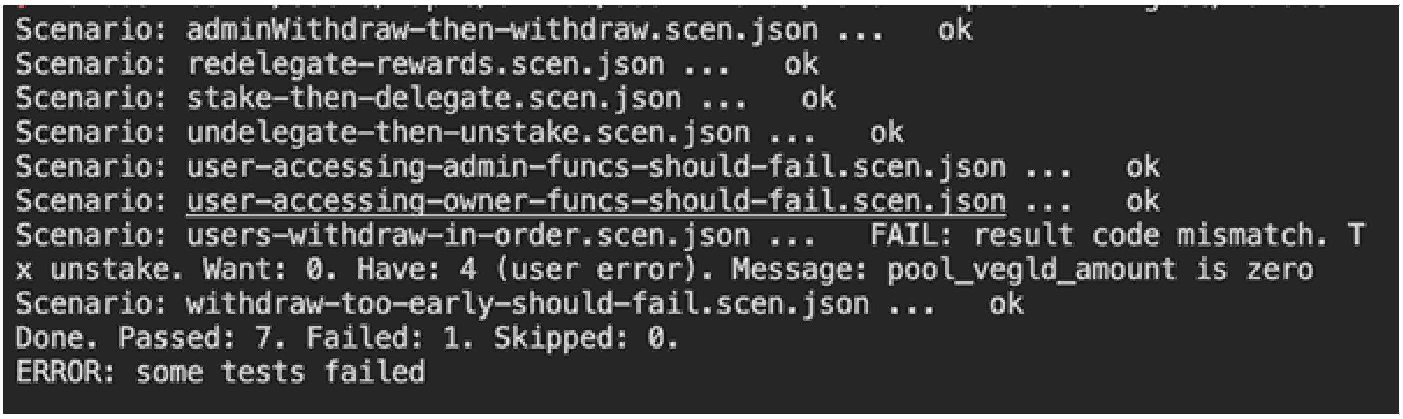
5. Avoid multiple storage reads for the same value, for example in **'quote_vegld'**, **'pool_egld_amount'** is read two times.

```
Status: Not addressed.
```

6. Write **'donate'** function for VEGLD also.

```
Status: Not addressed.
```

Test results



```
Scenario: adminWithdraw-then-withdraw.scen.json ...   ok
Scenario: redelegate-rewards.scen.json ...   ok
Scenario: stake-then-delegate.scen.json ...   ok
Scenario: undelegate-then-unstake.scen.json ...   ok
Scenario: user-accessing-admin-funcs-should-fail.scen.json ...   ok
Scenario: user-accessing-owner-funcs-should-fail.scen.json ...   ok
Scenario: users-withdraw-in-order.scen.json ...   FAIL: result code mismatch. T
x unstake. Want: 0. Have: 4 (user error). Message: pool_vegld_amount is zero
Scenario: withdraw-too-early-should-fail.scen.json ...   ok
Done. Passed: 7. Failed: 1. Skipped: 0.
ERROR: some tests failed
```

Audited source code version
97b2e81d45ae99d0ab5b878d8d231485bf8a5a854ba66c38e2246a11dcd3c747

* The audited source code version is the sha256sum of the zip archive containing the source code that was sent.

Second review source code version
c58ca91ec766a36ae410d3af7e30a337ed30882b

* The audited source code version is the hash of the latest commit.