

# Certification

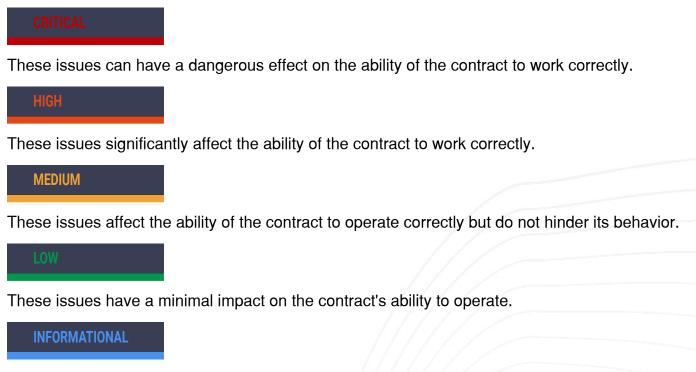
# **STAKING CONTRACT**



Feb 9th, 2024 / v.0.2 Audited source code version: 2d1364e1fedf777130f89ca40de67668d202eb37

## 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 do not impact the contract's ability to operate.



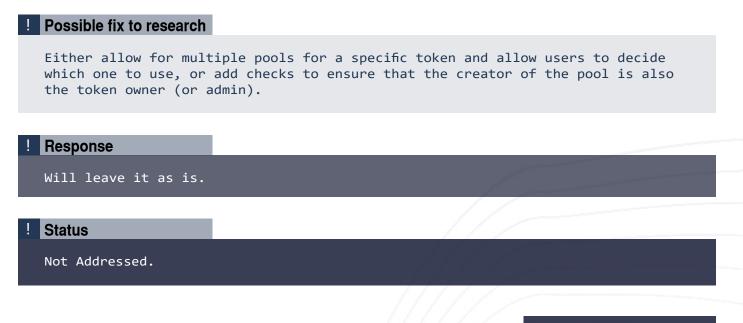
OneDEX Contract Audit

#### Issues

#### 1. Frontrun Pool

Not Addressed / HIGH

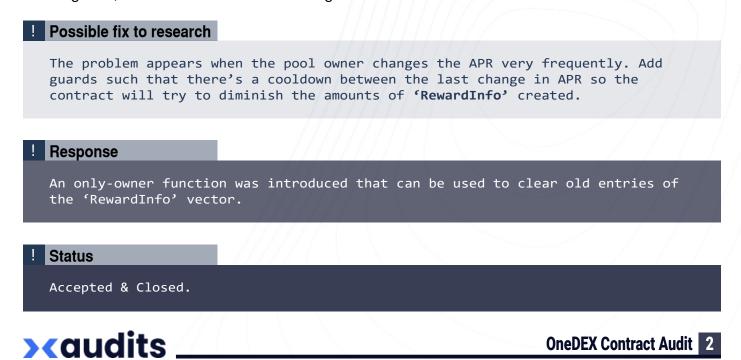
Description: There can only be one pool for a specific token. Since there are no ownership checks on the specific user that creates the pool for a token, one can frontrun the creation of the pool to the disadvantage of the well intended creator.



#### 2. Out of gas / read operation

Fixed / HIGH

Description: When calculating the reward for a user, the smart contract does an iteration over all the **'RewardInfo'** structs. A new **'RewardInfo'** struct is created each time the pool owner changes the APR. Assuming that there are few changes in the APR, this should not be a problem, but if the vector grows, user funds are at risk of being locked in the contract forever.



#### 3. Out of gas / read operation

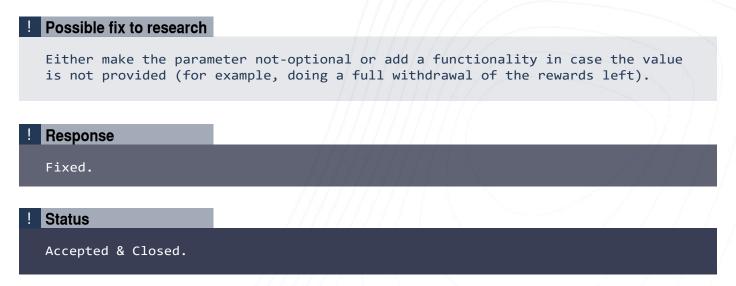
Description: In the view function **'getDaoMembers'**, the smart contract does an iteration over all the users that are currently staking for a specific token. This can easily run out of gas / read operations when the number of users reaches a reasonable value.

!	Possible fix to research						
	Implement the user set using an iterable method such that one can iterate all the users in a controlled manner (for example using <b>'offset'</b> and <b>'limit'</b> ).						
!	Response						
	The function was remo	oved.					
!	Status						
	Accepted & Closed.						

#### 4. Full Withdraw

Fixed / MEDIUM

Description: The owner of a pool can withdraw the pool's rewards using 'withdrawRewards' endpoint. The endpoint has an optional argument which represents the amount of the reward that he wants to withdraw. If this value is not given (being optional), one might expect to withdraw the full reward amount left but instead, the TX just fails (it tries to withdraw zero).



#### 5. Fees Leftover

Description: The contract splits the fee retained when creating a new pool into two. One half to the **'treasury\_address'** and the other to **'burner\_address'**. However if the fee amount is an odd number, a rounding error of **'1'** will remain in the contract.

Possible fix to research			
Send <b>'amount / 2'</b> to order to avoid encour	'amount - amount ,	/ 2' to the ot	her in
! Response			
Fixed.			
! Status			
Accepted & Closed.			

#### 6. Unused decimals

Description: When a pool is created, 'stake\_token\_decimals' is required, however it is not used in the contract.

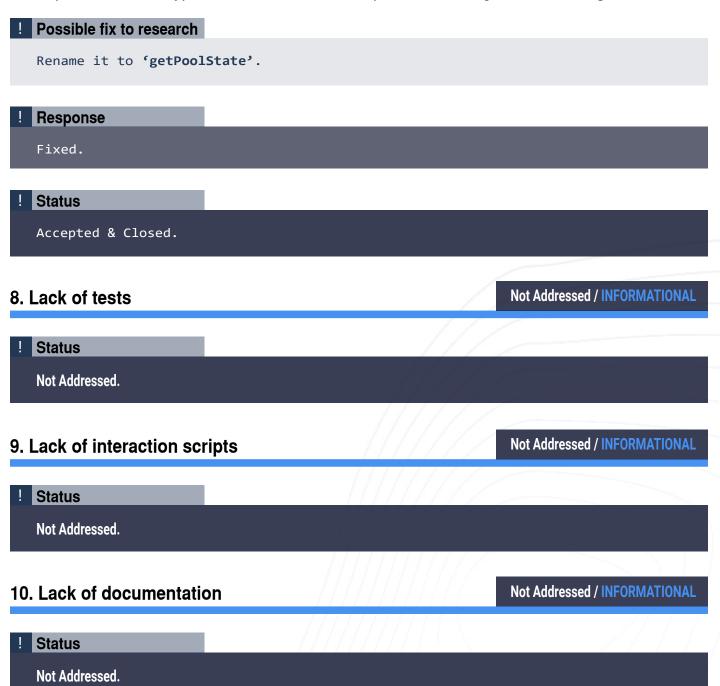
<b>!</b> Possible fix to research				
Either use the value	where it was	intended doing	; the development	or remove it.
! Response				
Fixed.				
_				
! Status	/			
Accepted & Closed.				



Fixed / LOW

#### 7. Storage name

Description: There's a typo in the view name of the pool state storage. It is named 'getPoolte'.



audits

OneDEX Contract Audit 5

### **Verification Conditions**

#### 1 Only the owner of the streaming swap can call 'cancelStreamingSwap'.

```
#[only_owner]
#[endpoint(setCreationCost)]
fn set_pool_creation_cost(
```

#### 2 Pool admin functions are guarded correctly.

```
self.assert_pool_owner(pool_id);
```

#### **3** Valid payments are checked on input.

```
require!(
   stake_token_id == self.pool_stake_token_id(pool_id).get(),
        "Invalid Stake token id"
);
```

4 Actions are taken when the contract and the pool are not paused (and only on valid pools).

```
self.assert_valid_pool_id(pool_id);
self.assert_unpaused();
self.assert_pool_unpaused(pool_id);
```



# Suggestions (Optional)

1. Update to latest framework version (best practice when deploying a new contract).

2. Write tests (Rust Testing Framework is recommended, documentation (at least a readme) and interaction scripts.

3. Format the code using 'cargo fmt' and solve the warnings (check them using 'cargo clippy').

4. Construct the **'Pool'** structure and store it inside a single storage instead of splitting pieces of information between multiple storages.

# **Test results**

There are no tests.

Audited source code version ef45ea78ccaa59ab2a0932f92aedcfdbea3cd791

Second Audited source code version 2d1364e1fedf777130f89ca40de67668d202eb37

