SOMISH

Blockchain Labs

Smart Contract Audit Report

BoidToken

September 20th, 2019

Index

Summary

Audit Report prepared by Somish Blockchain Labs for BoidToken smart contract.

Process and Delivery

Two (2) independent experts performed an unbiased and isolated audit of the code below. The debrief took place on September 20th, 2019, and the final results are presented here.

Audited Files

The following contract(s) were covered during the audit:

- boidtoken.cpp
- boidtoken.hpp

Client Documents

The following document(s) were provided by the client for the purpose of the audit:

- https://docs.google.com/document/d/1_RARbyAd_t4wRMr4C8bXGPi9Ot9evK9O8GgTqKKirMs - highlighting BOID terminology and token functionality
- https://docs.google.com/document/d/1cPYx2Qj5PU5jhiyg8tuDtjW4NIUOiaIlawjPCa7O3ro/edit#heading=h.w55e8c33vfra - highlight BOID mainnet plan
- https://docs.google.com/document/d/13GhMc3cj7jWBBu6r3wew7yGo-eLZLdtkT8neACQWm_Y - further documentation on token functions

Notes

Audit was performed on commit **f6d131af5557b33aa41eae918a1394c86a99c4ee**

Intended Behavior

Terminology

- Stake - Frozen tokens that receive a return on investment. The stake pool is useful for predicting token inflation. In the future it will be useful for analyzing voting patterns as well
- Season - Period of time that staked tokens remain frozen
- Delegator - Stake account sending tokens. Token owner
- Delegate - Stake account receiving tokens.
- Power - Computational contributions to the BOID network

- Powered stake - Amount of stake that is eligible for bonus returns
- Power/stake bonus - BOID tokens are given as bonuses in two portions: for accounts that have boidpower and for accounts that have staked tokens
- Difficulty - Amount that boidpower or stake contributions to the BOID network have a positive effect on the network. Set a higher difficulty to give out higher rewards and thus a higher inflation

Usages

Staking

- Delegates tokens to be frozen in the stake pool
- Can be done from one account to itself or to another.
- Stakes must be set with a time limit in seconds, but a time limit of 0 denotes an indefinite time limit (not yet implemented). After the set time limit, the tokens will expire. They are no longer eligible to gain bonuses and they are available to return to the token owner.
- All staking assumes the form of a delegated stake, with special rules for self-delegations.
- Staking sends a message to the delegator.
- Staking can be done from an account's liquid balance or from it's staked balance, assuming that there are sufficient tokens that are self-delegated.
- Staking can only be done out of season by all accounts except for the token issuer, which can stake in-season as well
- Stake delegation an account that currently holds delegated tokens will reset the delegation time limit as well as add to the delegated token amount.

Claiming

- Claims available token bonuses. Bonuses are accrued since the previous claim.
- Claiming has the additional effects to return expired tokens. Expired tokens from both the delegator and the delegate will be returned on a call to claim
- Claim bonuses for staked tokens are only available until the token expiration date. A side effect of this is that claim bonuses for delegated tokens from alphabetically higher named accounts are given precedent when claiming. This is important because an account can only collect bonuses on staked tokens up to that account's allowed powered_stake quantity
- In the future, an account will be able to claim a percentage of tokens to their staked account and the rest to their liquid account.

Unstaking

- Removes delegated tokens from the delegate
- Tokens can be returned to the liquid balance or the staked balance
- Tokens can only be returned to the liquid balance out of season and only after the time limit has expired
- Tokens can be returned to the staked balance only after the time limit has expired

- Unstaking can be done by the staked account as well as the token issuer. The token issuer can unstake for any account in-season or out-of-season

Token recycling
- Deflate tokens by returning them to the available supply from the token issuer account

Account reclaim
- Request to reclaim tokens thru app
- This will open the BOID account that was previously erased by the contract owner
- The contract owner will resend previously airdropped BOID tokens to your newly opened account

Issues Found

Critical

1. Contract account can transfer token holder's funds to any other account.

The **reclaim()** function, contains an authentication check, **require_auth(get_self())** which implies that **contract account** can transfer **BOID** tokens from token holder's account to any other account without his/her approval. This can result in serious trust issues as non-contract accounts can lose their tokens.

Recommendation
Replace the authentication check with **require_auth(account)**. Only token holder should be able to transfer funds.

2. The contract owner has authority to delete any record from several critical tables.

By using erase functions such as **erasetoken(), erasestats(), eraseacct()**, etc, the smart contract account can remove any record. This can lead to trust issues and instability of the system. As all tables are interlinked with one another, there is a chance of creating instability in the system by deleting records in any table. For example, If a user has **BOID** token and **BOID** token from stats table is being deleted, It will show **BOID** token in the accounts table but **BOID** token will not be available in **stats** table. This will result in freezing of all tokens as functions like **transfer(), stake(), unstake()** will start throwing.

Similarly, if a row from **staked** table is removed, the token holder loses tokens and there is an incorrect token supply.

Recommendation
Consider removing functions that forcefully change the state of the system's data structure. In case data needs to be changed, make these internal functions callable only as per code flow.

Major

1. No check for valid accounts in transtake().

No checks have been applied to ensure the validity of **to** account in **transtake**(). This can result in transferring stake to invalid accounts, resulting in perpetual loss of tokens.

Recommendation
Ensure the validity of accounts using **is_account**() check.

2. Issuer can unstake indefinitely staked tokens.

The **unstake()** function allows the smart contract account to unstake tokens on the token holder's behalf as long as the variable **issuer_unstake** is true. We understand that this functionality has been added as a service and that it makes sense for tokens whose stake has expired. However, the smart contract account should be restricted from unstaking tokens that were indefinitely staked by the token holder.

3. Incorrect authentication checks for issuer unstake.

The In **unstake()** function, for issuer unstake, the check should be **require_auth(issuer)** but **require_auth(get_self())** found.According to available check, issuer unstake can be callable by token owner but it should be callable by issuer.

4. Implementation does not match documentation for reclaim() function.

The implementation of reclaim function differs from documentation provided as under:

Documentation	Implementation
- Account holder can request to reclaim tokens thru app	Function not callable by account holder. Check **require_auth(get_self())** ensures it is callable by smart contract account

	only.
- This will open the BOID account that was previously erased by the contract owner	There is no check if the **token_holder** was previously erased or not.
- The contract owner will resend previously airdropped BOID tokens to your newly opened account	The function is transferring tokens from 1 account to another. No new tokens being issued. Also,an account can be erased by the issuer via **eraseacct**(), only if it has 0 balance. Hence, we could not understand the purpose of reclaiming tokens
- {no mention in documentation}	Additionally, the function focuses on balance and does not cover transferring stakes.

5. The system is dependent on the contract owner account for switching season break on/off.

Since token holders can only unstake to liquid balance when stakebreak is **on** and stakebreak switching authority is with the owner account, there is a high dependency on the smart contract account. If the smart contract account never switches the stakebreak on, the user shall never be able to move his funds to a liquid account.

Recommendation
We suggest using fixed time seasons post which there is a minimum stake break period within which token holders can unstake if they wish to.

6. The contract owner has the authority to modify any record from any table

By using set functions such as **setstakeinfo**(), the smart contract account can modify any record. This can lead to trust issues and instability of the system. As all tables are interlinked with one another, there is a chance of creating instability in the system by modifying records in any table. For example, If **active_accounts/total_staked** being manually being set to some random value by using **setstakeinfo**(), the value of **active_accounts/total_staked** will be differ from real value.

Recommendation
Consider removing functions that forcefully change the state of the system's data structure. In case data needs to be modified, make these internal functions callable only as per code flow.

7. No check for valid value in set configuration parameter functions.

There are missing checks for valid input in few setter functions such as **setstakeinfo(), setstakediff(), setpowerdiff(), setpowerrate(), setpwrstkmul(), setminstake(), setmaxpwrstk(), setmaxwpfpay(), etc.** It may lead to unwanted values for invalid inputs. For example, On calling **setstakeinfo()** with **active_accounts** = -23, will set **active_accounts** in stakeconfigs table 4294967273. This is because the function accepts active_accounts as uint. Similarly, minstake can be set as negative.

8. recyclewpf() only callable if issuer=contract.

The function **recyclewpf()**, contains two authentication checks, ie, **require_auth(get_self())** & **require_auth(issuer)** which implies that **issuer** must be equal to **contract account**. This function will become uncallable if **issuer** and **contract account** are not the same.

Recommendation
Remove one of the two authentication checks.

9. Non-contract accounts can not claim bonuses.

The **claim()** function, contains two authentication checks.ie, **require_auth(get_self())** & **require_auth(stake_account)** which implies that **stake_account** must be equal to **contract account**. This results in non-contract accounts being unable to claim their bonuses.

Recommendation
There should only be one check **require_auth(stake_account)** which shall allow account holders to claim their bonus.

10. Problems with function claim()

The logic to distribute stake and power bonus has been commented. Hence, currently, the function does nothing, except returning the expired stakes to the stakeaccount.

Also, while the function works for a self stake, it does not work for a stake to another account. Eg, if alice stakes x BOID to Bob and post expiration period, alice calls the claim function, it does not work.

Minor

1. No check whether BOID is a valid token.

Few functions such as **updatebp()**, **resetbonus()**, **reclaim()**miss the check whether BOID token exists in **stats** table or not. A check for valid existence of BOID token should be applied wherever the token is being used.

2. No check for the minimum staking period.

There is no check for minimum staking period in **transtake()** and **stake()** but comments above **stake()** function mention that "**Stake period must be valid staking period offered by this contract**".

3. No logic is available for calculation of power bonus.

In function **get_power_bonus(),** there is no logic that updates **power_payout.** The function is only reading some tables but neither updating anything nor returning anything.

4. Condition expiration time > current time will always be true.

In function **claim_for_stake()**, **check(claim_time <= curr_time)** shall always return true as in the above **if** condition, the value of **claim_time** is being set in such a way that it can never be more than **curr_time**.

5. Overflow/Underflow checks

Consider adding checks for value overflow and underflow and div/0 wherever int, uint or float is used. For example:-
expiration_time in **transtake()** function
powered_stake_amount in **claim()** function
wpf_amt, stake_coef, payout_amount and **wpf_payout_amount** in **get_stake_bonus()** function

Notes

1. Code commenting

Consider adding comments to functions and variable declarations in the contracts as it gives clarity to the reader and reduces confusion for the developers as well. Comments are missing in a few functions.

2. Tokens are being issued to the issuer and then transferred to the user

In function **issue**(), if to is not the same as **issuer**, the tokens are being issued to issuer 1st and then it is transferring to to.

Recommendation
Instead of issuing to the **issuer** and then transferring to the user, tokens can directly be issued to the user. It will reduce code complexity and save RAM.

3. Should throw in case supply falls below 0

In function **recycle**(), in case supply is falling below 0, the code just raises a warning and sets supply as 0. In an ideal scenario, this should never be the case and hence the transaction should revert instead.

4. Few functions only handle BOID token

Token **issue()/create()** can issue/create tokens of any **SYM** but there are certain functions that only handle **BOID** token. such as **get_power_bonus(), reclaim(), updatebp(), setminstake(), recyclewpf()**.

Recommendation
Consider, making function that handles all possible tokens. If the intention is to only create **BOID** token then consider restricting the creation of any other kind of tokens.

5. RAM Optimizations.

- **Consider removing unused arguments/variables**
  Function argument **percentage_to_stake** is not used anywhere in **claim()** function.

Config table variables like **Boidpower_decay_rate, total_season_bonus,** etc are not used anywhere.

- **No point of else block**
  In **open()** function, nothing happens inside **else if** block, it is just reading the table and not updating anything. So, consider removing that **else if** block.
- **Add check for balance>0**
  In **reclaim()** function, consider adding check **if(balance>0)** before transferring**.** because there is no point in transferring 0 amount and it will cost extra RAM and CPU.

6. Maintain single table for stake_row and delegations

Consider removing table which store similar data. We understand that the intention may be to find records based on different keys. However, we recommend using the concept of multiple keys instead of maintaining the same data in two tables. Removing duplicate tables shall lead to decrease in RAM cost and will avoid any unforeseen errors caused by duplicate data.

7. Notify ‘to‘ address when someone is transferring stake

Consider adding a line that notifies **to** address when someone transfers stake to him/her.

8. Remove unnecessary comments from the contract file

Consider removing unnecessary comments from the contract file. It will make the contract much clear to read.

Closing Summary

Upon audit of the BOID token smart contract, it was observed that the contract contains critical, major and minor issues, along with several areas of notes.

We recommend that the critical, major and minor issues should be resolved. Resolving for the areas of notes are up to BOID's discretion. The notes refer to improving the operations of the smart contract.

Disclaimer

Somish Blockchain Labs's audit is not a security warranty, investment advice, or an endorsement of the BOID's platform or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Somish from legal and financial liability.

Somish Solutions Limited