BlockWRK SMART CONTRACT AUDIT REPORT

by Quill Audits, October 2018

Introduction

This Audit Report highlights the overall security of BlockWRK Smart Contracts. With this report, we have tried to ensure the reliability of their smart contract by complete assessment of their smart contract codebase.

Auditing Approach and Methodologies applied -

Quillhash team has performed thorough testing of the project starting with analysing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract in order to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase we coded/conducted Custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration of our multiple team members and this included -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line
3. Deploying the code on testnet using multiple clients to run live tests

4. Analysing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities
5. Checking whether all the libraries used in the code are on the latest version
6. Analyzing the security of the on-chain data

Summary of BlockWRK Smart Contracts:-

BlockWRK contracts are customised ERC20 with following added functionality:-

- **Taxed Token** variation on the standard ERC20 transfer function (from OpenZeppelin) to impose a variable transaction fee on each token transfer that occurs outside of the BlockWRK application environment.

- **Transaction Handler** function that allows users to send tokens within and also outside of the BlockWRK application without the need to hold Ether for paying gas costs.

- **Internal Token Purchase** function that allows for a variable fee to charged when users load their wallets with tokens from the Distribution Pool.

- **Internal Token Distribution** function that allows the Application to distribute tokens for Proof-of-WRK and other administrative token transfers.

- **In-App Purchases** function that has variable rates for Users and business accounts to purchase WRK without needing to buy from an external cryptocurrency exchange.

It also contain an contract to create sub-admins or sub-owners which can be authorised and removed only by owner of the contract.

Contracts also contains an ico contract to distribute tokens in different tiers according to the rate of on-going tier.

Security Level references

Every issue in this report was assigned a severity level from the following:

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

High severity issues:-

No high severity issues are found.

Medium Severity Issues:-

1. Approval racing condition:- The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval, and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender is able to spend their entire approval amount twice.
Line no 155:- **approve() has a race condition problem.**

approve() doesn't check if the value of allowance is equal to 0 before performing operation.
We recommend disabling users from calling this function if the value of allowance is not equal to 0.
We also recommend adding this code before performing operation:

```
require((_value == 0) || (allowed[msg.sender][_spender] == 0));
```

Status : Fixed

2. Negative tokens approval :- Contract should not be able to approve negative value tokens but this test case is failing.
Add:

```
require((_value == 0) || (allowed[msg.sender][_spender] == 0));
if(allowed[msg.sender][_spender] == 0){
        require(_value > 0);
        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, spender, value);
        return true;
}
else {
```

```
        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, spender, value);
        return true;
}
```

Status : Fixed

3. Wrong numAuthorized value:- It should be checked before removing an address from authorized list that the address is already authorised.If address is not already authorised and owner will try to remove the address then it will decrement the numAuthorized value but actual number of authorised addresses are greater than numAuthorized.

Status : Fixed

(It is demonstrated in unit test cases of authorizable contract.)

Low Severity Issues:-

1. **Solidity version must be fixed(Always use latest Version).**

It should not `pragma solidity ^0.4.24;`
It should be `pragma solidity 0.4.24;`

Status : Fixed

2. **keccak256 encoding behavior**

Before ABI encoding functions were introduced, keccak function accepts multiple arguments like

=> keccak256("AAAA", "BBBB", 42);

It has been implicitly doing encodePacked. But now if you try calling keccak with those, you are likely to get a compiler warning.

Warning:-This function only accepts a single "bytes" argument

To remove these warning, replace following lines:-

Line 82 :- keccak256(_to, _value, _fee, _nonce);

With:- keccak256(abi.encodePacked(_to, _value, _fee,_nonce));

Line 94:- keccak256("\x19Ethereum Signed Message:\n32", _hash);

With:- keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", _hash));

Status : Fixed

Unit Testing

Test Suite

1. Contract: Authorizable

Passed Test cases:

Contract: authorizable contract

✓ should have an owner which is msg.sender (146ms)

✓ should allow owner to add authorised address (126ms)

✓ should allow increment numAuthorized (56ms)

✓ should allow owner to remove authorised address and decrement numAuthorized (59ms)

✓ should not increment numAuthorised ,if owner is adding an address which is already authorised (89ms)

✓ should prevent non-owners from adding (102ms)

✓ should prevent non-owners from removing (97ms)

✓should not decrement numAuthorised ,if owner is removing an address which is not authorised

2. Contract: **Taxed contract**

✓ should be able transfer token with taxed fee

✓ should be able approve tokens and transfer tokens

✓ should revert if balance of sender is less than transfer amount

✓ should revert if balance of sender is less than transferFrom amount

✓ should revert if spender does not have enough approved allowance

✓ should not transfer negative token amount

✓ should not be to able approve negative tokens

✓ should not allow an owner to approve tokens to spender before setting allowance to zero

3. Contract: BlockWRK tokens contract

✓ should initialize constructor (156ms)

✓ should be able to distribute inAppTokens by authorised address (103ms)

✓ should be able to distribute inAppTokens to inAppPurchaseWallet address (82ms)

✓ should not able to distribute inAppTokens by unAuthorised address

✓ should not able to distribute inAppTokens more than distributionPool

✓ should not able to distribute inAppTokens more than distributionPool

✓ should be able to set new tax rate

✓ non owner should not be able to change tax rate

✓ authorised address should be able to do inAppTokenPurchase (119ms)

✓ emits the transfer event on inAppPurchase

✓ reverts when receipient address is zero inApp token Purchase

✓ emits the transfer event on inAppDistribution (41ms)

✓ should not able to use inAppPurchase by unAuthorised address

✓ should not able to do inAppPurchase more than purchase wallet balance

✓ should not able to do inAppPurchase with negative tokens passed

✓ should be able to handle transactionHandler (322ms)

✓ should be able to handle transactionHandler if sender is unAuthorised

✓ should be able to handle transactionHandler if balance is less

✓ should be able to handle transactionHandler if signature is wrong

---

4. Contract: BlockWRKICO

✓ Crowdsale should be ended only after end

✓ should reach cap if cap sent

✓ when the beneficiary is not the zero address when the wei amount is not zero when the total wei raised is less than the hardcap and when the crowdsale is open

✓ should show correct tokens remaining in current tier

✓ transfers funds to the salesWallet

✓ emits the purchase events

✓ reverts when the amount sent plus the wei raised is more than the hardcap.

✓ reverts when the wei amount is zero

✓ reverts when the beneficiary is the zero address
transfer remaining tokens after the sale

✓ transfers the remaining tokens to the distribution pool wallet,when available tokens is greater than zero,when sender is the owner and when the crowdsale has ended

✓ emits the closeout sale event (42ms), when available tokens is not greater than zero

✓ Should reverts when the crowdsale has not ended

✓ Should reverts when the sender is not the owner

Final Result of Test:

✓ **49 passing**

❌ **0 failing**

Implementation Recommendations

=> Contract does not have much functionality to track nonce of preSigned transactions.According to ERC865, nonce is the number of presigned transactions sent from contract. So there should be a function to get the nonce for build new transaction on frontend.

=> There should be a function to pause the sale and transfer function in case of any bug arises in future.Pause function will be helpful to prevent the loss.

=> Similar to _transferPreSigned(), a function to approve tokens can be added so that users can also use transferFrom() function without paying fee in gas.

Comments:

Overall, the code is clearly written, and demonstrates effective use of abstraction, separation of concerns, and modularity. BlockWRK development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.
We would like to recommend that BlockWRK team continue with the process of securing their code by posting public bug bounties and soliciting community feedback. It would be better to conduct public bounties if possible because the ERC-865 is still not a finalized standard .