KEYis Smart Contracts Final Audit Report

The Code is located in the KEYis github Repository and the version used for this commit is.
`d30a969dbc9df61b295e33a8201235d64611a2a1`

5th January 2018

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:-

1) Token amount is not multiplied by 1 ether or 10^18 to get sufficient amount of tokens, that’s why after calling buyToken function by whitelisted user having value 1 ether, investor is not be able to get 2000 tokens, they are only getting 0.000000000000002000

Status : fixed by developer

2) Test case fail of approving negative tokens, it should not approve negative tokens.

Status : fixed by developer

Medium Severity Issues:-

1) Avoid state change after transfer function or external calls token sale contract line no 225 must be before transfer function.

Status : fixed by developer

Low Severity Issues:-

1.) Solidity version must be fixed(Always use latest Version).
It should not pragma solidity ^0.4.23;
It should be pragma solidity 0.4.23;

Status : fixed by developer

2.) you can remove event emit at line no 229 as already transfer function contain event emit, at etherscan event is listen and it's showing ERC20 token is transferred to two address, it means it's listening two transfer event even though both are same still you can remove from line no 229.

Status : fixed by developer

3.) your specification document contain compiler version 0.4.21, and you are using ^0.4.23 make sure you are using same resources, compilation version and libraries that are listed in your paper.

Status : fixed by developer

Review Audit Contains Following Issues :

High severity issues:-

1) Logic for `buyToken()` is not correct if i send ether sufficient to buy all the tokens in one go, then token sold in tier 1 is 15000000, token sold in tier 2 is 135000000 and token sold in tier 3 is zero .

Also if you will not end Sale immediately when all the sale token is sold then by participating in a sale, investor will drain all the tokens in contract and you will not be able to end Sale as team and other will not be able to get sufficient tokens

Status : Not fixed yet

Solution : You are updating all variables locally like tier = 0 always, you can globally update variable and every time buy function is called tier will be start from current tier in automatic tier change.

```
{ tokensSold[tier + 1] = tokensSold[tier + 1].add(leftoverQuantity); }
```

Will add tokens to next tier but the line shown below is out of if() scope and will check token sold in previous tier not the current tier.

```
    require(tierToLimits[tier].sub(tokensSold[tier]) >= quantity);
```

You should check token sold in current tier using require condition immediately after balances is updated in that tier and in same if() scope.

Also avoid transfer of tokens in fractions at line 199, you are transferring tokens that are remaining in that tier and then before checking next tier left tokens, balance is update you can hold the remaining tokens in a local variable, and check further before transferring tokens, if next tier have sufficient amount of tokens to transfer or not.

```
balances[msg.sender] = balances[msg.sender].add(leftoverQuantity);
balances[address(this)] = balances[address(this)].sub(leftoverQuantity);
```

Check tokens left in that tier before updating balance.

```
require(tierToLimits[tier].sub(tokensSold[tier])>=quantity or leftoverquantity);
```

```
tokensSold[tier].add(quantity or  leftoverquantity);
```

Use require after updating balance

```
require(tierToLimits[tier] >= (tokensSold[tier]);
```

2.) As you are not following above require conditions in correct manner and not updating variables in their scope, this raised a big issue, if you will fail to end sale using endSale() function immediately after tokens for investors is sold completely, token that alloted for team and cost will be sold to investors as tokens are in contract and tier 3 will not reach and tokens will be directly drawn by investors, and neither you will be able to end sale nor tokens for team and cost will be send to them.

Status : Not fixed yet

Use solution given in high severity issue 1 to solve this issue also you can use below given require statement.

```
require( getTokensSold() <= investorAlloc );
```

Unit Testing

Test Suite Result

✓ Should correctly initialize constructor values of Contract (1612ms)

✓ Should change Owner account from accoutns[0] to accounts[9] (456ms)

✓ Should change Owner account from accoutns[9] to accounts[0] (226ms)

✓ Should change cost allocation token account from accounts[0] to accounts[1] (363ms)

✓ Should change Team allocation token account from accoutns[0] to accounts[2] (231ms)

✓ Should change withdraw wallet account from accoutns[0] to accounts[3] (302ms)

✓ Should whitelist accounts[4] (286ms)

✓ Should whitelist accounts[6] (385ms)

✓ Should whitelist accounts[5] (290ms)

```
      ✓ Should whitelist accounts[8]  (320ms)
      ✓ Should buy correct Tokens when manual tier is off accounts[4] (870ms)
      ✓ Should buy correct Tokens when manual tier is off accounts[5] (604ms)
      ✓ Should buy correct Tokens when manual tier is off accounts[6] (1094ms)
      ✓ pause and get sale status (191ms)
      ✓ unpause and get sale status (200ms)
      ✓ Should Burn tokens (330ms)
      ✓ Should not Burn Negative tokens (187ms)
      ✓ Should remove whitelisted accounts[1]  (295ms)
      ✓ should Approve address to spend specific token (178ms)
      ✓ should not Approve address to spend negative tokens  (125ms)
      ✓ should not be able to transfer negative approved tokens  (71ms)
```

Implementation Recommendations

Now Contract doesn't contain issues that are listed in initial report, KEYis development team shows great work to solve issues and make contract more secure but some of the functionalities that were not tested in initial audit because of some high severity issues need to be fix before testing further. As now KEYis development team have fixed all issues and we are able to test further, Our Audit team found some high severity issues that need to be solve before deploying on main net, all the issues with their possible solution is listed with recommendations.

Comments:-

Overall, the code is clearly written, and demonstrates effective use of abstraction, separation of concerns, and modularity. KEYis development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.
We found some critical issue and several additional issues that require the attention of the KEYis team. Given the subjective nature of some assessments, it will be up to the KEYis team to decide whether any changes should be made.

Previous hash commit used to Audit

`e323342db359e18f532392b0d4d4d733ff4b0566`

Transaction Hash of all the test performed on test network (Rinkeby).

https://docs.google.com/spreadsheets/d/1NJY_mLfpPCzsbpD6l1DfgaB4whxcXofxdk4dTfN_c5Q/edit?usp=sharing