DLT Labs

Presearch Code Audit

By

DLT Labs

Table of Contents

1. INTRODUCTION

1.1. DOCUMENT PURPOSE

DLT Labs conducted the review of the smart contracts that makes up the **Presearch Token Sale** service. The findings of the review are presented in this document.

1.2. REVIEW GOALS & FOCUS

Sound Architecture

This review includes both objective findings from the contract code as well as subjective assessments of the overall architecture and design choices. Given the subjective nature of certain findings, comments from the Presearch development team have been included in the report as appropriate.

Smart Contract Best Practices

This review will evaluate whether the codebase follows the current established best practices for smart contract development.

Code Correctness

This review will evaluate whether the code works as expected.

Code Quality

This review will evaluate whether the code has been written in a way that ensures readability and maintainability.

Security

This review will look for exploitable security vulnerabilities.

1.3. TERMINOLOGY

This review uses the following severity terms which indicates the magnitude of an issue:

Minor

Minor issues are generally subjective in nature, or potentially deal with topics like "best practices" or "readability". Minor issues in general will not indicate an actual problem or bug in code.

The maintainers should use their own judgment as to whether addressing these issues improves the quality of codebase.

Major

Major issues will be things like bugs or security vulnerabilities. These issues may not be directly exploitable such as requiring a specific condition to arise to be exploited.

Left unaddressed these issues are highly likely to cause problems with the operation of the contract or lead to a situation which allows the system to be exploited in some way.

Critical

Critical issues are directly exploitable bugs or security vulnerabilities. Left unaddressed these issues are highly likely or guaranteed to cause major problems or potentially a full failure in the operations of the contract.

2. SOURCE CODE

2.1. SOURCE CODE

The source code under review was shared over the email on following dates:

- 1st Version - 2nd October 2017
- 2nd Version - 3rd October 2017
- 3rd Version – 8th October 2017
- 4th Version – 11th October 20177

2.2. CONTRACTS

The code audit carried out was limited to the smart contract(s) shared over the email. There was only one smart contract **preToken.sol** which was reviewed.

3. DISCLAIMER

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status.

4. FINDINGS

This section contains detailed issues and analysis on basis of white paper shared on 2nd October 2017.

4.1. CRITICAL ISSUES

Issue# 001
Code commit#: 2nd October 2017
Description:

The first version of the code was not getting compiled.

Action:

Team Presearch fixed the above issue by adding following lines in code which were missing in the previous version:

//crowdsaleAddress

address public crowdsaleAddress;

Issue# 002
Code commit#: 2nd October 2017
Description:

In function ***setCrowdsaleAddress,*** an incorrect variable was used to check address null condition. Variable ***crowdsaleAddress*** was used instead of ***newCrowdsaleAddress*** while checking the address null condition. The function ***setCrowdsaleAddress*** will always throw error as ***crowdsaleAddress*** was not initialized.

Action:

Team Presearch fixed the above issue by using the correct variable name. Also **crowdsaleAddress** was initialized to ***msg.sender*** in the contract constructor call, this sets the **crowdsaleAddress** to the address of the person who deploys the contract.

Issue# 003
Code commit#: 3rd October 2017
Description:

In function ***mint*** and ***burn*** logic to calculate ***totalSuppy*** was not written in the optimal way and it could be optimized to save gas cost.

It was written as **totalSupply = totalSupply += _amount** and **totalSupply = totalSupply -= _amount** respectively in ***mint*** and ***burn***. This could be optimized as **totalSupply += _amount** and **totalSupply -= _amount** respectively. This optimization will save around

5000 gas in both functions. Use of SafeMath library was also recommended for calculations.

Action:

Team Presearch fixed the issue by using the recommended optimized statement and also included the SafeMath library. The above-mentioned code was modified to:

Mint: ***totalSupply = totalSupply.add(_amount);***

Burn: ***totalSupply = totalSupply.sub(_amount);***

4.2. MAJOR ISSUES

Issue# 004
Code commit#:3rd October 2017
Description:

Following variables remained constant throughout the life cycle of contract:

- *maxSupply*
- *initialSupply*
- *unlockDate*
- *name*
- *symbol*
- *decimals*
- *version*

So, all the above variables should have been declared as constant since their values are supposed to remain constant throughout. Making the variables constant also reduces the gas cost.
Action:
Team Presearch fixed the above issue by defining all above variables as a public constant.

Issue# 005
Code commit#:3rd October 2017
Description:

In functions ***transfer*** and ***transferFrom,*** there was no **Zero-Address check** validation on input parameters ***_to*** and ***_from***.
Action:
Team Presearch fixed the above issue by adding the **Zero-Address *check*** on both parameters ***_to*** and ***_from***.

Issue# 006
Code commit#:3rd October 2017
Description:
In function ***approve,*** there was no **Zero-Address check** validation on input parameter ***_spender.***
Action:
Team Presearch fixed the above issue by adding the **Zero-Address *check*** on ***_spender*** parameter.

Issue# 007
Code commit#:3rd October 2017
Description:
According to white paper, value of token ***name*** and ***symbol*** should have been set to "***Pre-search***" and "***PRE***" respectively but it was found that in smart contract value of ***name*** and ***symbol*** was specified as "***pstTest***" and "***PST***" respectively.

Action:

Team Presearch fixed the above issue by placing correct values as mentioned in the white paper.

4.3. MINOR ISSUES

Issue# 008
Code commit#:3rd October 2017
Description:
During audit following functions were identified as publicly visible functions which were not explicitly specified as public:

- *preToken()*
- *balanceOf ()*
- *transfer()*
- *transferFrom()*
- *approve()*
- *allowance()*
- *transferOwnership()*
- *mint()*
- *burn()*
- *setCrowdsaleAddress()*

It was recommended to specify public visibility explicitly for functions that needs to be publicly visible.

Action:
Team Presearch fixed the issue by specifying the visibility of all above functions as ***public***.

Issue# 009
Code commit#:11th October 2017
Description:
According to latest standard, it is recommended to define functions which can read the state of blockchain but cannot write in it as ***view*** instead of constant. So we recommend to use ***view*** instead of constant in following functions:

- allowance()
- balanceOf()

Action:
Team Presearch fixed the above issue by changing all above functions from ***constant*** to ***view***.

Issue# 010
Code commit#:3rd October 2017
Description:
According to latest standard, it is recommended to define functions which can neither read the state of blockchain nor write in it as ***pure*** instead of constant. So we recommend to use ***pure*** instead of constant in following functions:

- ***mul()***
- ***div()***
- ***sub()***
- ***add()***

Action:
Team Presearch fixed the above issue by changing all above functions from ***constant*** to ***pure***.

Issue# 011
Code commit#:3rd October 2017
Description:
Data type of ***decimal*** was defined ***uint256*** but according to standard recommendation it should have been ***uint8***

Action:
Team Presearch fixed the above issue by changing the data type of ***decimal*** from uint256 to ***uint8***.

Issue# 012
Code commit#:11th October 2017
Description:
An old solidity version (0.4.11) was used so it was recommended to update solidity version to **0.4.17**

Action:

Team Presearch fixed the above issue by upgrading solidity version to **0.4.17**

Issue# 013
Code commit#:11th October 2017
Description:
Since solidity supports scientific notations, it was recommended to use them while initializing constants so Instead of writing "maxSupply = 1000000000000000000000000000" and "initialSupply = 250000000000000000000000000", it should have been written as "***maxSupply = 1000000000e18***" and **"initialSupply = 250000000e18"**

Action:

Team Presearch fixed the above issue by specifying the ***maxSuppy*** and **initialSupply** value in scientific notation.

Issue# 014
Code commit#:11th October 2017
Description:
In function ***updateUnlockDate***, there is no check for ***_newDate*** > the start time of token sale which allows someone to update unlock date any time prior to starting time of token sale.
Action:

Team Presearch decided to not add the check, as it is by design that there is no official start time/date.

Issue# 015
Code commit#:11th October 2017
Description:
In function ***updateUnlockDate***, we were checking ***require (_newDate <= 1512018000);*** which means it will always compare with "November 30th, 2017 00:00:00 AM EST". Consider a scenario when someone changed the unlock date to "November 15th, 2017" so he would also be able to change it back to a later date (between "November 15th, 2017" and "November 30th, 2017") which contradicts with the comment mentioned above the same function.

Action:

Team Presearch decided to not add the check, as it is by design.

5. General Comments

5.1. Standard ERC-20 Token

PRE token is implemented by following ERC-20 compliant standard token standards. Thus, following the latest approved coding standard on Ethereum token. Link of approved EIP - https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md

5.2. Use of Assert and Require

Proper error handling and validation of input parameters have been done extensively throughout the code by using assert and require. The code base does not use the old way of "If .. throw" pattern. It has made use of new guard functions **assert()** and **require()**.

5.3. Short-Address Attack

Checks have been kept in place to handle the short-address. It has been handled by creating a modifier **onlyPayloadSize** and modifier is used at required places.

```
//Prevent short address attack
 modifier onlyPayloadSize(uint size) {
   assert(msg.data.length == size + 4);
   _;
 }
```

5.4. Zero-Address Check

Zero-Address check is present to prevent inadvertent transfer of Ether or Token to a 0x0 address. Checks like below are added

require(newCrowdsaleAddress != address(0));

The above check using require and address cast is the clean and elegant way of doing Zero-Address check.