28 MAY 2018

AIRBLOC

SMART CONTRACT AUDIT REPORT 2

HAECHILABS

01. INTRODUCTION

This report audits the security of the second pre-sale smart contract created by the AIRBLOC team. The HAECHI Labs audited the smart contract code produced by the AIRBLOC team to ensure that it was designed for the purposes outlined in the white paper and published materials, and that the code was secure.

The code used for this audit can be found in the "airbloc / token" Github repository (https://github.com/airbloc/token). The final commit of the code used for the audit is "d47404005f6596af955350a9e38ce01133c452be". Security audits for ABL tokens and pre-sale primary with ERC20 are not included. Please refer to the previous report for the security audit report on ABL tokens and pre-sale primary. The "openzepplin-solidity" library, which is an open source library, is not also subject to this audit.

A white paper on AIRBLOC can be found at the link.

02. AUDITED FILE

- PresaleSecond.sol

03. ABOUT “HAECHI LABS”

"HAECHI Labs" has a vision to contribute to a healthy blockchain ecosystem through technology. HAECHI Labs conducts in-depth research on the security of smart contracts. There has been many incidents caused by smart contract vulnerabilities including The DAO, Parity multisig wallet and SmartMesh (ERC20) hacking. "HAECHI Labs" is committed to designing and implementing the security auditing of smart contracts. We provide smart contract related services to enable customers to optimize the gas cost during operation and to implement a secure smart contract that meets their purpose. "HAECHI Labs" is composed of people who have experiences as software engineers in start-ups and lead blockchain researchers at Decipher (the blockchain research group at Seoul National University) and nonce research.

04. ISSUES FOUND

The issues found are divided into ‘major’ and ‘minor’ depending on their importance. Major issues should be modified since the code is neither secure nor implemented for its original intention. Minor issues would be potentially problematic and require modification. HAECHI Labs recommends the AIRBLOC team to improve on every issue found.

Major Issues

1. The token sale period is not explicit.

```
106     function ignite() external onlyOwner {
107         ignited = true;
108         emit Ignite();
109     }
110
111     function extinguish() external onlyOwner {
112         ignited = false;
113         emit Extinguish();
114     }
```

PresaleSecond.sol

The second pre-sale smart contract does not specify a token-sale period. Instead, the AIRBLOC team used the *ignited* variable to determine whether the token is being sold or not. However, the *ignite* and *extinguish* functions can be called at any time. In other words, it might seem suspicious that the AIRBLOC team can stop and run the token sale at any time. Therefore, we recommend the AIRBLOC team to see and implement timestamp or blocknumber in the *ignite* and *extinguish* functions. Once the value of the *ignited* changes from true to false, it is desirable that the value of the variable can not be re-true.

2. The calculation of refund in *getPurchaseAmount* is incorrect.

```
159     function getPurchaseAmount(address _buyer)
160         private
161         view
162         returns (uint256, uint256)
163     {
164         uint256 d1 = maxcap.sub(weiRaised);
165         uint256 d2 = exceed.sub(buyers[_buyer]);
166
167         uint256 refund = msg.value.sub(min(d1, d2));
168
169         if(refund > 0)
170             return (msg.value.sub(refund) ,refund);
171         else
172             return (msg.value, 0);
173     }
```

PresaleSecond.sol

The *refund* variable calculates the amount of excess ETH for refund when investors send more than the individual cap or the second pre-sale exceed its maxcap. For instance, let us assume that *maxcap* is 3800ETH, *weiRaised* is 0ETH, *exceed* 300ETH, and *buyers[_buyer]* value is 0ETH. When an investor sends 1ETH to participate in token sale at address *_buyer, msg.value* becomes 1ETH. Then, *d1* is 3800ETH and *d2* is 300ETH, which makes *min(d1, d2)* 300ETH because *min(d1, d2)* returns the smaller variable among *d1* and *d2*. Therefore, *msg.value.sub(300ETH)* is to subtract 300ETH from 1ETH.

The AIRBLOC team uses the "SafeMath" of "openzeppelin-solidity" for the *uint256* operation. Therefore, when the *sub* function is executed on line 167, 300 ETH is subtracted from 1ETH, which causes an exception to prevent Integer Underflow. As a result, investors cannot purchase tokens because an exception occurs even if a transaction for purchasing token is executed. In other words, investors only can invest if they transfer 300ETH, the exact value of *exceed*. However, most investors will fail to buy the tokens because most investors will not invest as much as exceed value. We recommend to use the logic to check whether the sum will reach over the maxcap of the second token sale when *msg.value*, what investor transfer, is added to the amount of ETH collected. And we also recommend to check if an investor's total investment reach over the individual cap when the msg.value is added to the amount of individual investment.

Minor Issues

1. The *MintableToken* imported into the library is not used.

```
3  import "openzeppelin-solidity/contracts/token/ERC20/ERC20.sol";
4  import "openzeppelin-solidity/contracts/token/ERC20/SafeERC20.sol";
5  import "openzeppelin-solidity/contracts/token/ERC20/MintableToken.sol";
6  import "openzeppelin-solidity/contracts/math/SafeMath.sol";
7  import "openzeppelin-solidity/contracts/ownership/Whitelist.sol";
8  import "openzeppelin-solidity/contracts/ownership/Ownable.sol";
```

PresaleSecond.sol

"MintableToken.sol" of "openzeppelin-solidity" is imported but not used. We recommend to delete *import* statements for unused libraries.

2. The keys data structure is unnecessary.

```
120      address[] public keys;
121
122      function getKeyLength()
123          external
124          view
125          returns (uint256)
126      {
127          return keys.length;
128      }
```

PresaleSecond.sol

The *keys* data structure in the token sale is used only in the *getKeyLegnth* function to determine the number of people who participated in the token sale. Functions that store or change state in storage use a lot of gas in smart contracts. Therefore, it is not recommended to store data unless it is absolutely necessary. We recommend to use the *Purchase* event log to get the number of people participating in the token sale.

3. If you can withdraw ETH during the pre-sale, it might seem suspicious.

```
256     function withdrawEther() public onlyOwner {
257         wallet.transfer(address(this).balance);
258         emit WithdrawEther(wallet, address(this).balance);
259     }
```

PresaleSecond.sol

The *withDrawEther* function can be executed regardless of the token sale period. Therefore, you can withdraw ETH during the token sale and use the withDrawEther for the development team to fill the token sale cap. These implementations might seem suspicious to investors. Therefore, we recommend to include a separate logic to confirm that the token sale period has ended before you can withdraw ETH.

4. If you can withdraw ABL tokens during the pre-sale, it might seem suspicious.

```
248     function withdrawToken() public onlyOwner {
249         Token.safeTransfer(wallet, Token.balanceOf(address(this)));
250         emit WithdrawToken(wallet, Token.balanceOf(address(this)));
251     }
```

PresaleSecond.sol

The *withDrawToken* function can be executed regardless of the token sale period. This function is used to withdraw the remaining ABL tokens that have not been sold since the second pre-sale ended. However, if you are able to withdraw ABL tokens during token sales, investors may find this suspicious. Therefore, we recommend the AIRBLOC team to use logic to confirm that the token sale period has ended before AIRBLOCK team can withdraw.

5. It is unclear when refund function can be called.

```
221     function refund(address _addr)
222         external
223         returns (bool)
224     {
225         require(!ignited && !finalized);
226         require(msg.sender == distributor); // only for distributor
227         require(_addr != address(0));
228
229         if(buyers[_addr] == 0) return false;
230
231         _addr.transfer(buyers[_addr]);
232         emit Refund(_addr, buyers[_addr]);
233
234         // TODO 동작하는지 확인
235         delete buyers[_addr];
236         return true;
237     }
```

PresaleSecond.sol

Generally, *refund* returns ETH to investors if the AIRBLOC team do not reach the minimum amount of ETH during the token sale. According to the AIRBLOC white paper, tokens that have not been sold out of second pre-sale will continue to be sold during the public sale. Therefore, the *refund* function is not required. If there are other reasons the AIRBLOC team wants to use *refund* function, we recommend the AIRBLOC team to write the reasons for writing the code in the white paper.

05. DISCLAIMER

The findings of this report is not exhaustive and is limited to the current understanding of known security patterns. This report only discusses known technical issues and does not include any investment advice, the profitability of the business model nor any other issues outside the technical realm. Regardless of the problems described in this report, there may be unknown problems and defects in Ethereum or the solidity language. In order to create a secure smart contract, we recommend the AIRBLOC team to fix the problems proposed in this report and conduct enough testing.