28 MAY 2018

AIRBLOC

SMART CONTRACT AUDIT REPORT 2

HAECHILABS

01. INTRODUCTION

본 보고서는 AIRBLOC 팀이 제작한 프리 세일 2차 스마트 컨트랙트의 보안을 감사하기 위해 작성되었습니다 HAECHI Labs 팀에서는 AIRBLOC 팀이 제작한 스마트 컨트랙트의 구현 및 설계가 백서 및 공개된 자료에 명시한 것처럼 잘 구현이 되어있고, 보안상 안전한지에 중점을 맞춰 감사를 진행했습니다.

Audit 에 사용된 코드는 “airbloc/token” Github 저장소(https://github.com/airbloc/token) 에서 찾아볼 수 있습니다. 보안 감사에 사용된 코드의 마지막 커밋은 “d47404005f6596af955350a9e38ce01133c452be” 입니다. ERC20인 ABL 토큰과 프리 세일 1차에 대한 보안 감사는 포함되지 않습니다. ABL 토큰과 프리 세일 1차에 대한 보안 감사 리포트는 이전 레포트를 참조하시기 바랍니다. 오픈 소스로 사용하는 “openzepplin-solidity” 라이브러리 역시 보안 감사에 포함되지 않습니다.

AIRBLOC 에 관한 백서는 아래 링크에서 열람 가능합니다.

02. AUDITED FILE

- PresaleSecond.sol

03. ABOUT “HAECHI LABS”

“HAECHI Labs” 는 기술을 통해 건강한 블록체인 생태계에 기여하자는 비전을 가지고 있습니다. HAECHI Labs는 스마트 컨트랙트의 보안 뿐만 아니라 블록체인 기술에 깊이있는 연구를 진행하고 있습니다. The DAO, Parity Multisig Wallet, SmartMesh(ERC20) 해킹 사건과 같이 스마트 컨트랙트의 보안 취약점을 이용한 사건들이 지속적으로 발생하고 있습니다. “HAECHI Labs”는 이러한 보안 사고들을 예방하기 위해 안전한 스마트 컨트랙트 설계와 구현 및 보안 감사에 최선을 다합니다. 고객사가 목적에 맞는 안전한 스마트 컨트랙트를 구현하고 운영시 발생하는 가스비를 최적화할 수 있도록 스마트 컨트랙트 관련 서비스를 제공합니다. “HAECHI Labs” 는 스타트업에서 다년간 Software Engineer로 개발을 하고 서울대학교 블록체인 학회 Decipher 와 nonce research 에서 블록체인 연구를 한 사람들로 구성되어 있습니다.

04. ISSUES FOUND

발견된 이슈는 중요도 차이에 따라 Major, Minor로 나누어집니다. Major 이슈는 보안상에 문제가 있거나 의도와 다른 구현으로 수정이 반드시 필요한 사항입니다. Minor 이슈는 잠재적으로 문제를 가져올 수 있으므로 수정이 요구되는 사항입니다. AIRBLOC 팀에서는 발견된 모든 이슈에 대하여 개선을 하는 것을 권장합니다.

Major Issues

1. 토큰 세일기간이 명확하지 않습니다.

```
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

프리 세일 2차 스마트 컨트랙트에는 토큰 세일 기간이 명시되어 있지 않습니다. 대신 *ignited* 변수를 이용해서 토큰 세일 중인지 아닌지를 판단합니다. 하지만 *ignite, extinguish* 함수는 아무때나 호출할 수 있습니다. 즉, AIRBLOC 팀이 아무 때나 토큰 세일을 중단하고 실행할 수 있다고 의심받을 수 있습니다. 따라서 *ignite, extinguish* 함수에서 timestamp 나 blocknumber 를 확인하고 실행할 수 있게 하는 것을 권장합니다. 그리고 *ignited* 변수의 값이 true 에서 false 로 변경된 뒤에 다시 true 될 수 없게 만드는 것이 바람직합니다.

2. *getPurchaseAmount* 에서 *refund* 계산 과정이 올바르지 않습니다.

```
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

refund 변수는 2차 프리 세일이 max cap 을 초과했거나 개인 투자자가 자신이 보낼 수 있는 exceed 보다 많은 이더리움을 보냈을 때, 초과된 이더리움을 환불하기 위한 값을 계산합니다. 예를들어, *maxcap* 이 3800ETH 이고 *weiRaised* 가 0ETH, *exceed* 300ETH, *buyers[_buyer]* 값이 0ETH 인 경우를 가정해봅시다. 이 때 *_buyer* 주소에서 토큰 세일에 참여하기 위해 1ETH 를 보낸 경우 *msg.value* 는 1ETH 가 됩니다. 그리고 앞에 가정한 값들에 의해서 *d1* 은 3800ETH, *d2* 는 300ETH 가 됩니다. *min(d1,d2)* 는 *d1*, *d2* 의 값 중 작은 값을 반환하므로 *min(d1,d2)* 는 300ETH 가 됩니다. 여기서 *msg.value.sub(300ETH)* 를 하게 되면 1ETH 에서 300ETH 를 빼게 되는것입니다.

*uint256*연산은 "openzeppelin-solidity" 의 "SafeMath" 를 사용하기 때문에 167번 째줄에서 *sub* 함수가 실행될 때 1ETH 에서 300 ETH 를 빼면서 Integer Underflow 방지를 위한 예외가 발생하게 됩니다. 따라서 투자자는 토큰을 구매하기 위해 트랜잭션을 발생시켰지만 예외가 발생하게 되므로 토큰을 구매할 수 없게됩니다.

즉, 투자에 성공하려면 *exceed* 의 값인 300ETH 를 입금해야만 합니다. 하지만 대부분의 투자자가 *exceed* 의 값만큼 이더리움을 입금하지 않을 것이기에 대다수의 투자자는 토큰 구매에 실패하게 됩니다. 따라서 투자자가 보낸 ETH 값인 *msg.value*가 지금까지 모금된 금액에 합해졌을 때 max cap을 넘는지 그리고 한 개인이 그동안 모금한 금액이 개인 cap 을 넘는지 확인하는 로직을 포함시킬 것을 권장합니다.

Minor Issues

1. 라이브러리로 *import* 된 *MintableToken* 이 사용되지 않습니다.

```
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

"openzeppelin-solidity" 의 "MintableToken.sol" 이 *import* 되어있지만 사용하지 않습니다. 사용하지 않는 라이브러리의 *import* 문을 삭제하는 것이 좋습니다.

2. *keys* 자료구조는 불필요합니다.

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

PresaleSecond.sol

토큰 세일에 참여한 *keys* 자료구조가 단순히 토큰 세일에 참여한 사람의 수를 구하기 위한 *getKeyLegnth* 함수에서만 사용됩니다. storage 에 상태를 저장하거나 변경하는 함수는 스마트 컨트랙트에서 굉장히 많은 가스를 사용합니다. 따라서 반드시 필요한 데이터가 아니면 저장하지 않는 것이 좋습니다. 토큰 세일에 참여한 사람의 수를 얻기 위해서는 *Purchase* event 로그를 사용하는 방법을 권장합니다.

3. 프리 세일 도중에 모은 이더리움을 출금하면 의심을 받을 수 있습니다.

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

PresaleSecond.sol

withDrawEther 함수는 토큰 세일 기간과 관계없이 실행될 수 있습니다. 따라서 토큰 세일 도중에 이더리움을 출금하고 출금한 이더리움을 토큰 세일의 Cap 을 채우기 위해 개발팀이 다시 투자하기 위해 사용할 수 있습니다. 이러한 구현은 투자자들에게 의심을 받을 수 있습니다. 따라서 출금을 하기 전에 토큰 세일 기간이 끝났는지 확인하는 로직이 필요합니다.

4. 토큰 세일 도중에 ABL 토큰을 출금할 수 있습니다.

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

PresaleSecond.sol

withDrawToken 함수는 토큰 세일 기간과 관계없이 실행될 수 있습니다. 해당 함수는 프리 세일 2차가 종료되고 판매되지 않고 남아있는 ABL 토큰을 회수하기 위해 사용됩니다. 하지만 목적과 무관하게 토큰 세일 도중에도 ABL 토큰을 출금할 수 있게 되어있다면 투자자들에게 의심을 받을 수 있습니다. 따라서 출금을 하기 전에 토큰 세일 기간이 끝났는지 확인하는 로직이 필요합니다.

5. *refund* 함수가 언제 호출 될 수 있는지 명확하지 않습니다.

```
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

일반적으로 refund 는 토큰 세일별로 최소 모집 이더리움의 양이 있어서 그 금액을 넘기지 못하면 투자자들에게 다시 이더리움을 돌려주는 기능입니다. 에어블록 백서에 따르면 프리 세일 2차에서 다팔지 못한 토큰은 퍼블릭 세일때 이어서 팔게됩니다. 따라서 *refund* 함수의 기능이 필요하지 않습니다. 혹시나 다른 조건이 존재해서 refund 를 해야된다면 이유를 코드나 백서에 명시하는게 좋다고 생각합니다.

05. DISCLAIMER

해당 리포트는 투자에 대한 조언, 비즈니스 모델의 적합성, 버그 없이 안전한 코드를 보증하지 않습니다. 해당 리포트는 알려진 기술 문제들에 대한 논의의 목적으로만 사용됩니다. 리포트에 기술된 문제 외에도 이더리움, 솔리디티 상의 결함, 발견되지 않은 문제들이 있을 수 있습니다. 안전한 스마트 컨트랙트를 작성하기 위해서는 발견된 문제들에 대한 수정과 충분한 테스트가 필요합니다.