

28 MAY 2018



AIRBLOC
SMART CONTRACT
AUDIT REPORT 1

01. INTRODUCTION

본 보고서는 AIRBLOC 팀이 제작한 ABL ERC20 토큰과 프리 세일 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.

Audit에 사용된 코드는 “airbloc/token” Github 저장소(<https://github.com/airbloc/token>)에서 찾아볼 수 있습니다. Audit에 사용된 코드의 마지막 커밋은 “dd150e02317b29cfa07b04cf8f36766a34972076”입니다. 해당 코드 중 미리 이더리움 메인넷에 배포된 “ABLG.sol”, 사용되지 않는 “ABLGEchanger.sol”, “OwnedTokenTimelock.sol”, 오픈소스 라이브러리인 “openzeppelin-solidity” 관련 코드는 audit의 대상이 아닙니다.

AIRBLOC에 관한 백서는 [아래 링크](#)에서 열람 가능합니다.

02. AUDITED FILE

- ABL.sol
- PresaleFirst.sol
- OwnableToken.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 깊이 있는 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 사고 등을 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 Decipher와 nonce research에서 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. HAECHI Labs는 AIRBLOC 팀이 발견된 모든 이슈에 대하여 개선하는 것을 권장합니다.

Major Issues

1. ABL 토큰의 발행량이 백서에 명시된 발행량과 다릅니다.

```

12     // Token Distribution Rate
13     uint256 public constant SUM = 400000000;    // totalSupply
14     uint256 public constant DISTRIBUTION = 221450000; // distribution
15     uint256 public constant DEVELOPERS = 178550000; // developer
16
17     // Token Information
18     string public name = "Airblob Token";
19     string public symbol = "ABL";
20     uint256 public decimals = 18;
21     uint256 public totalSupply = SUM;

```

ABL.sol

ABL 토큰의 발행량은 400000000(SUM)이고 토큰 세일을 통해 분배되는 토큰량은 221450000(DISTRIBUTION), 개발팀이 소유하는 토큰량은 178550000(DEVELOPERS)입니다. 하지만 decimals 값이 18 이므로 totalSupply 는 SUM 에 해당하는 값은 400000000 가 아니라 $400000000 * (10 ^ \text{decimals})$ 이 되어야합니다. 이를 올바르게 수정하기 위해 SUM, DISTRIBUTION, DEVELOPERS 에 각각 $10 ^ \text{decimals}$ 을 곱하는 것이 바람직합니다.

2. PresaleFirst.sol에서 받은 이더를 ABL 토큰으로 교환하는 공식이 잘못됐습니다.

```

78     // buy
79     uint256 tokenAmount = purchase.div(1000000000000000000).mul(rate);
80     maxcap = maxcap.sub(purchase);
81

```

PresaleFirst.sol

투자자들이 이더리움을 프리세일 컨트랙트에 입금하고 미리 정해진 교환 비율에 맞게 ABL 토큰을 받게 됩니다. 하지만 백서에 명시된 프리세일 1차 교환 비율과 코드에 기술한 내용이 다릅니다. 백서에서는 프리세일 1차에 참여한 투자자는 1ETH 당 11500ABL 을 받을 수 있습니다. 그러나 PresaleFirst.sol 에서는 위의 코드처럼 `purchase.div(1000000000000000000).mul(rate)` 를 받게 되어있습니다. 1ETH = 10^{18} wei 이고 decimals 가 18 이므로, 올바른 ABL 토큰 교환 공식은 `purchase.mul(rate)` 가 되어야 합니다.

Minor Issues

1. ABL 토큰을 lock 하는 구현은 제거하는 것이 좋습니다.

```

44     bool isLocked = true;
45
46     modifier locked() {
47         require(!isLocked);
48         _;
49     }
50
51     function unlock() public onlyOwner {
52         isLocked = false;
53     }
54
55     function lock() public onlyOwner {
56         isLocked = true;
57     }
58
59     function transfer(address _to, uint256 _value) public locked returns (bool) {
60         return super.transfer(_to, _value);
61     }

```

ABL.sol

ABL 토큰의 `transfer` 함수에 `locked modifier`가 존재하여 `isLocked` 값에 따라서 토큰 전송을 막을 수 있는데 이러한 구현은 충분히 의심을 살 수 있습니다. ABL 토큰 판매기간이 끝나도 스마트 컨트랙트 주인이 토큰 `transfer`를 아무때나 `lock` 해서 악용할 수 있다고 판단됩니다. 예를 들면 거래소에 상장할 때 개발팀이 팔 물량을 미리 옮겨두고 `transfer`를 일시적으로 `lock` 해서 다른 투자자들은 `transfer` 못하게 하고 시세를 올려서 혼자만 파는 구조를 만들 수 있습니다. 따라서 이러한 `lock` 구현이 `transfer`에 있는 것은 바람직하지 못합니다. 토큰 세일 기간동안 토큰이 교환되고 판매되는게 걱정된다면 토큰 세일 기간이 다 끝난 뒤 토큰을 분배하거나 `lock` 을 풀면 다시 못걸게 하는 장치가 필요합니다.

2. PresaleFirst.sol 의 Fallback function 은 public 보다 external 이 적합합니다.

```

60     function () public payable {
61         collect(msg.sender);
62     }

```

PresaleFirst.sol

`PreSaleFirst` 의 fallback function 인 `function () public payable` 은 `public` 보다 `external` 이 적합합니다. `public` 은 `internal`, `external` 모두 호출이 가능하고 gas 소모량이 `external`에 비해 높습니다. 해당 function 은 투자자들이 토큰을 구매하기 위해 호출할 때에 사용되기 때문에 내부적으로 호출될 필요가 없으므로 `public` 보다 `external` 을 사용하는게 좋습니다.

3. PresaleFirst.sol에서 `maxcap`을 줄이는 것은 초기 cap을 추적하기 어렵게 합니다.

`maxcap`은 프리세일에서 받을 수 있는 이더리움의 양입니다. PresaleFirst에서는 토큰 세일이 진행될 때마다 입금된 이더리움의 양을 `maxcap`에서 빼서 전체 `maxcap`을 줄이고 있습니다. 하지만 이럴 경우 `maxcap`이 계속 바뀌게 되어서 초기에 올바른 `maxcap`이 설정 됐는지 추적하기 어렵습니다. 따라서 `maxcap`은 constant 변수로 놓고 `weiRaised`와 같은 새로운 변수를 두어서 `maxcap`을 `weiRaised`가 넘기는지 판단하는 방식으로 구현하는 게 옳습니다.

4. 프리 세일의 토큰 세일 기간이 명시적이지 않습니다.

```

11 module.exports = async (deployer, network, accounts) => {
12   const [_ , owner, wallet, buyer, fraud] = accounts;
13
14   let token;
15   let presale;
16
17   let startTime = Date.now() + duration.days(1);
18   let endTime = startTime + duration.weeks(1);
19
20   const maxEth = 1500 * ETH;
21   const exdEth = 300 * ETH;
22   const minEth = 0.5 * ETH;
23
24   const rate = 11500;
25
26   // deploy ABL token
27   await deployer.deploy(ABL, owner, owner);
28   await deployer.deploy(PresaleFirst, startTime, endTime, maxEth, exdEth, minEth, wallet, ABL.address, rate);
29 }

```

2_deploy_contracts.js

PresaleFirst의 `startTime`, `endTime` constructor를 통해 정해지는데 “`2_deploy_contracts.js`” 스마트 컨트랙트 배포 스크립트에서 주입하는 `startTime`, `endTime` 이 토큰 세일 기간과 다를 수 있어보입니다. 위의 코드 이미지의 17-18번째 줄을 보면 `Date.now()`를 통해서 세일 시작 시간을 정하고 있습니다. 이럴 경우 정확히 프리 세일 24시간 전에 배포 스크립트를 실행시켜야만 홈페이지에서 명시한 프리 세일 기간이 설정됩니다. 토큰 세일이 시작하고 끝나는 기간에 맞는 unix epoch timestamp를 constant로 넣는 것이 바람직합니다.

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

```

64     function collect(address _buyer) public payable onlyWhitelisted whenNotPaused {
65         require(_buyer != address(0));
66
67         require(preValidation());
68
69         uint256 refund;
70         uint256 purchase;
71
72         (refund, purchase) = getTokenAmount(_buyer);
73         emit TokenAmount(_buyer, refund, purchase);
74
75         // refund
76         _buyer.transfer(refund);
77
78         // buy
79         uint256 tokenAmount = purchase.div(1000000000000000).mul(rate);
80         maxcap = maxcap.sub(purchase);
81
82         // wallet
83         wallet.transfer(purchase);
84         buyers[_buyer].FundAmount = buyers[_buyer].FundAmount.add(purchase);
85         buyers[_buyer].TokenAmount = buyers[_buyer].TokenAmount.add(tokenAmount);
86         emit BuyToken(_buyer, purchase, tokenAmount);
87     }

```

PresaleFirst.sol

프리 세일 진행 중에 모인 이더리움을 출금할 수 있다면 프리 세일 도중에 이더리움을 출금하고 프리 세일 스마트 컨트랙트에 재입금할 수 있습니다. 이러한 기능은 개발팀이 maxcap 을 채우기 위해 혹은 개발팀이 소유하는 토큰을 늘리기 위해 악용될 수 있습니다. 예를 들어 출금한 10개의 이더리움을 다시 프리 세일 스마트 컨트랙트에 전송한다면 스마트 컨트랙트는 20개의 이더리움을 입금 받았다고 기록합니다. 이를 반복할 경우, 적은 양의 이더리움으로 maxcap 을 충분히 채울 수 있게 되고 적은 양의 이더리움으로 많은 양의 ABL 토큰을 받을 수 있게 됩니다. 모금된 이더리움을 바로 지갑으로 옮기지 않고, 나중에 프리 세일이 끝나고 ABL 토큰을 release 할 때 이더리움이 스마트 컨트랙트에서 지갑으로 이동하게 코드를 짜는 것을 추천합니다.

6. PresaleFirst.sol 의 Buyer struct 에 불필요한 데이터 저장으로 gas 가 낭비됩니다.

```

51     struct Buyer {
52         uint256 FundAmount;
53         uint256 TokenAmount;
54     }

```

PresaleFirst.sol

`Buyer` struct에 `FundAmount` 와 `TokenAmount` 가 존재하는데 `TokenAmount` 는 $FundAmount * rate$ 을 이용해서 구할 수 있는 값입니다. 따라서 `TokenAmount` 가 저장되지 않아도 `TokenAmount` 를 구해낼 수 있습니다. 이러한 불필요한 데이터 저장은 gas 의 낭비로 이어집니다.

7. PresaleFirst.sol 의 `getTokenAmount` 함수의 기능이 분리될 필요가 있습니다.

```

104     function getTokenAmount(address _buyer) private returns (uint256, uint256) {
105         uint256 cAmount = msg.value;
106         uint256 bAmount = 0;
107         uint256 pAmount = 0;
108
109         // get exist amount
110         if (buyers[_buyer].FundAmount != 0) {
111             bAmount = buyers[_buyer].FundAmount;
112
113             if (bAmount >= exceed) {
114                 emit LogString("Buyer cannot purchase over exceed");
115                 revert();
116             }
117         } else {
118             keys.push(_buyer);
119         }
120
121         if (cAmount >= exceed) {
122             pAmount = exceed;
123         }
124
125         // 1. check individual hardcap
126         if (cAmount.add(bAmount) > exceed) {
127             pAmount = exceed.sub(bAmount);
128         } else {
129             pAmount = cAmount;
130         }
131
132         // 2. check sale hardcap
133         if (pAmount >= maxcap) {
134             pAmount = maxcap;
135         }
136
137         return (cAmount.sub(pAmount), pAmount);
138     }

```

PresaleFirst.sol

`getTokenAmount` 함수는 두가지를 반환합니다. 첫번째, ABL 토큰 구매에 사용되는 이더리움의 양을 반환합니다. 두번째, 개인 cap과 프리 세일의 cap을 확인하고 토큰 구매에 사용되지 않는 이더리움의 양을 반환합니다. 복잡한 함수는 안전한 스마트 컨트랙트를 작성하기 위한 원칙에서 벗어납니다. 함수의 로직은 간단해야합니다. 그리고 모듈화를 시켜서 함수를 간단하게 유지해야합니다. 그래야 테스트 케이스 작성에도 용이하며 오류를 최소화할 수 있습니다. 따라서 `getTokenAmount` 함수는 ABL 토큰 구매에 사용되는 이더리움의 양과 반환되는 이더리움의 양을 한번에 계산하는 것이 아니라 기능을 분리할 필요가 있습니다. 마찬가지로 cap을 검사하는 함수들도 별도로 작성하는 것이 이더리움의 안전한 스마트 컨트랙트 작성 원칙에 적합합니다.

또한 해당 함수의 연산 로직은 복잡하고 연산과정 도중에 revert 함수를 호출하는등 비효율적인 부분들이 존재합니다. 위

의 코드 111~115번째 줄에서 `buyers[_buyer].FundAmount` 와 `exceed` 값을 비교해서 `revert` 를 하는 것을 연산 도중에 하는 것이 아니라 `getTokenAmount` 함수의 최상단에 `require` 문으로 확인하고 실패에 대해 빠르게 알리는 것을 권장합니다.

8. `LogString` Event 는 항상 발생하지 않습니다.

“Minor Issue 7” 에서 설명한 `getTokenAmount` 함수에서 114번째 줄을 보면 `LogString` event 를 발생시킵니다. 하지만 115번째 줄에 있는 `revert` 함수 때문에 `LogString` 은 항상 발생하지 않습니다. 따라서 불필요한 코드를 제거하는 것을 권장합니다.

9. `OwnableToken` 대신 `Whitelist` 를 사용하기를 권장합니다.

`OwnableToken` 은 owner 로 지정받은 사람이 새롭게 owner 를 지정할 수 있는 구조로 설계되어있습니다. 이럴 경우 무분별하게 owner 가 확산될 수 있으므로 위험할 수 있습니다. 그리고 한번 owner 가 되면 자기 자신이 `transferOwnership` 를 통해 owner 를 양도하기 전까지 권한이 사라지지 않습니다. 따라서 owner 로 지정한 사람이 악의를 품을 경우 막을 방법이 없습니다. 특정 기능을 선택받은 사람만 호출하게하길 원한다면 “openzeppelin-solidity” 라이브러리의 `Whitelist` 를 사용하는게 더 안전하다고 판단됩니다.

10. `PresaleFirst.sol` 의 `release` 함수를 하기 전에 transfer lock 상태를 확인하기를 권장합니다.

```

155     function release() public onlyOwner onlyOnce {
156         require(block.timestamp > endTime);
157         once = true;
158
159         for(uint256 i = 0; i < keys.length; i++) {
160             token.safeTransfer(keys[i], buyers[keys[i]].TokenAmount);
161             emit Release(keys[i], buyers[keys[i]].TokenAmount);
162         }
163     }

```

PresaleFirst.sol

이더리움에서 스마트 컨트랙트의 로직을 실행시키는 것은 비싸기 때문에 함수의 로직을 실행시키기 전에 조건을 미리 체크하는 것은 중요합니다. 현재의 `release` 함수는 160번째줄의 ABL 토큰 `transfer` 함수를 실행할 때 ABL 토큰에 lock 이 걸려있으면 실패하게 됩니다. `release` 함수의 상단부에 `require` 한 줄을 추가하여 `transfer` 가 lock 이 걸려있는 상태인지 아닌지 확인하는 것을 추천합니다. ABL 토큰의 `transfer` 함수에 lock 을 제거하는 것이 더 이상적입니다.

11. PresaleFirst.sol 의 `release` 함수는 `public` 보다 `external` 이 적합합니다.

“Minor Issue 10” 에서 살펴본 `release` 함수를 `public` 으로 선언하면 `internal`, `external` 모두 호출이 가능합니다. 하지만 해당 함수는 프리 세일이 끝난 이후 개발팀에 의해서 직접 호출될 것이므로 `public` 보다는 `external` 로 한정하는 것이 적합합니다. 또한 `external` 은 `public` 보다 gas 소모량이 적습니다.

12. PresaleFirst.sol 의 `release` 함수에서 `token` 을 전송하고 `token` 을 차감할 것을 권장합니다.

만약 external call 이 `release` 함수를 다시 호출한다면 “Reentrancy” 문제가 발생할 수 있습니다. 여기서 “Reentrancy” 이슈는 `buyer`에게 전송할 토큰의 양을 0으로 설정하지 않아서 external call 이 다시 `release` 함수를 호출하면 같은 주소에게 여러번 토큰을 전송하는 것입니다. 이러한 문제를 해결하기 위해 `tokenAmount` 를 0으로 바꾼뒤 `token` 을 전송하는 것이 올바른 구현 방법입니다. 물론 해당 코드에서는 `token` 이 Airbloc 팀이 작성한 ABL 토큰이므로 신뢰할 수 있는 스마트 컨트랙트이고 Reentrancy 를 방지하기 위한 `onlyOnce` 구현도 되어있어서 당장 문제가 발생하지는 않습니다. 하지만 안전한 코딩 패턴을 지향하는 것이 좋습니다. 따라서 “Minor Issue 10” 에서 살펴본 `release` 함수에서 ABL 토큰을 전송하기전에 `mapping buyer`에서 `amount` 를 0 으로 변경하기를 권장합니다.

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.