

01 JUN 2018



SMART CONTRACT AUDIT REPORT 1

HAECHILABS

01. INTRODUCTION

본 보고서는 Onther 팀이 제작한 ERC20 토큰과 토큰 세일 스마트 컨트랙트의 보안을 감사하기 위해 작성되었습니다. HAECHI Labs 팀에서는 Onther 팀이 제작한 스마트 컨트랙트의 구현 및 설계가 보안상 안전한지에 중점을 맞춰 감사를 진행했습니다.

Audit에 사용된 코드는 “Onther-Tech/tokyo” Github 저장소(<https://github.com/Onther-Tech/tokyo>)에서 찾아볼 수 있습니다. “Onther-Tech/tokyo” 저장소에서 “tokyo/packages/tokyo-reusable-crowdsale/audit/full-features” 경로에 있는 스마트 컨트랙트 솔리디티 코드가 Audit 대상입니다. Audit에 사용된 코드의 마지막 커밋은 “45fe2a0998c2dc415cbac961c5e59fbaed67c477”입니다. 해당 코드중 “zeppelin”, “minime” 폴더에 있는 오픈소스 라이브러리 코드는 audit의 대상이 아닙니다.

코드 저장소 링크: <https://github.com/Onther-Tech/tokyo/tree/45fe2a0998c2dc415cbac961c5e59fbaed67c477/packages/tokyo-reusable-crowdsale/audit/full-features>

02. AUDITED FILE

- base/common/HolderBase.sol
- base/crowdsale/BaseCrowdsale.sol
- base/crowdsale/BlockIntervalCrowdsale.sol
- base/crowdsale/BonusCrowdsale.sol
- base/crowdsale/FinishMintingCrowdsale.sol
- base/crowdsale/KYCCrowdsale.sol
- base/crowdsale/MiniMeBaseCrowdsale.sol
- base/crowdsale/MinimumPaymentCrowdsale.sol
- base/crowdsale/MintableBaseCrowdsale.sol
- base/crowdsale/PurchaseLimitedCrowdsale.sol
- base/crowdsale/StagedCrowdsale.sol
- base/crowdsale/ZeppelinBaseCrowdsale.sol
- base/kyc/KYC.sol
- base/locker/Locker.sol
- base/token/BurnableMiniMeToken.sol
- base/token/ClaimableToken.sol
- base/token/NoMintMiniMeToken.sol
- base/valut/MultiHolderVault.sol



- base/wallet/MultiSigWallet.sol
- AuditFullFeaturesCrowdsale.sol
- AuditFullFeaturesToken.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 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 Decipher 와 nonce research 에서 블록체인 연구를 한 사람들로 구성되어 있습니다.



04. ISSUES FOUND

HAECHI Labs는 Onther 팀이 발견된 모든 이슈에 대하여 개선하는 것을 권장합니다. 이어지는 이슈 설명에서는 코드를 세부적으로 지칭하기 위해서 {파일 이름}:{줄 번호} 포맷을 사용할 것입니다. 예를 들면, Token.sol:20은 Token.sol 파일의 23번째 줄을 지칭합니다.

1. HolderBase.sol 의 `initHolders` 함수 호출 시 Integer Overflow 가 발생할 수 있습니다.

```

35     function initHolders(address[] _addrs, uint96[] _ratios) public onlyOwner {
36         require(!_initialized);
37         require(holders.length == 0);
38         require(_addrs.length == _ratios.length);
39         uint256 accRatio;
40
41         for(uint8 i = 0; i < _addrs.length; i++) {
42             holders.push(Holder(_addrs[i], _ratios[i]));
43             accRatio = accRatio.add(uint96(_ratios[i]));
44         }
45
46         require(accRatio <= ratioCoeff);
47
48         initialized = true;
49     }
```

HolderBase.sol

HolderBase.sol:41 에서는 `initHolders` 함수의 파라미터인 `_addrs` 배열을 `for loop` 을 이용해서 `holders` 자료구조에 저장합니다. `for loop` 을 실행할 때 `i` 의 자료형이 `uint8` 이므로 2^8 인 256번까지만 `for loop` 이 올바르게 작동합니다. `_addrs` 배열의 크기가 256 보다 클 경우 `uint8` 에서 Integer Overflow 가 발생할 수 있습니다. 따라서 `require` 문을 이용해서 함수의 파라미터로 전달 받는 `_addrs` 배열 크기를 한정시키는 것을 권장합니다. 또는 `_addrs` 가 256 명 이상의 주소를 받을 수 있다면 `uint8` 보다 많은 bit를 이용하는 자료형을 사용하길 권장합니다.

2. HolderBase.sol 의 `initHolders` 함수의 파라미터인 `_addrs`, `_ratios` 배열 크기가 0이여도 초기화가 진행됩니다.

`initHolders` 함수의 파라미터인 `_addrs`, `_ratios` 배열에 빈 배열이 들어간다면 초기화가 올바르게 진행된다고 볼 수 없습니다. 따라서 `require` 문을 이용해서 배열의 크기가 0 보다 큰지 확인하는것을 권장합니다.



3. HolderBase.sol 의 *initHolders* 함수에서 *accRatio* 계산을 위한 자료형 변환이 올바르지 않습니다.

HolderBase.sol:43 에서 *accRatio* 를 계산할 때 *_ratios[i]* 의 자료형을 *uint96* 으로 변환합니다. 하지만 *accRatio* 는 *uint256* 이고 SafeMath 의 *add* 함수는 *uint256* 연산에 대해서 수행하게 설계 되었기 때문에 자료형 변환시에 *uint96* 이 아니라 *uint256* 으로 자료형을 변환하는 것을 권장합니다.

4. BaseCrowdsale.sol 에서 정의한 *buyTokensPostHook* 는 override 되지 않습니다.

BaseCrowdsale.sol:186 에 정의된 *buyTokensPostHook* 함수는 다른 Crowdsale 파일에서 단 한번도 override 되지 않습니다. 사용하지 않는 함수는 정의하지 않고 *buyTokens* 함수에서 호출하지 않는 것을 추천합니다.

5. BaseCrowdsale.sol 의 *buyTokens* 에서 가스 사용을 줄일 수 있습니다.

```

59     function buyTokens(address beneficiary) public payable whenNotPaused {
60         require(beneficiary != address(0));
61         require(validPurchase());
62
63         uint256 weiAmount = msg.value;
64
65         uint256 toFund = calculateToFund(beneficiary, weiAmount);
66
67         uint256 toReturn = weiAmount.sub(toFund);
68         require(toFund > 0);
69
70         buyTokensPreHook(beneficiary, toFund);
71
72         // calculate token amount to be created
73         uint256 tokens = getTokenAmount(toFund);
74
75         // update state
76         weiRaised = weiRaised.add(toFund);
77
78         if (toReturn > 0) {
79             msg.sender.transfer(toReturn);
80         }
81
82         buyTokensPostHook(beneficiary, tokens, toFund);
83
84         generateTokens(beneficiary, tokens);
85         emit TokenPurchase(msg.sender, beneficiary, toFund, tokens);
86         forwardFunds(toFund);
87     }
```

BaseCrowdsale.sol

BaseCrowdsale.sol:68 에 있는 *require* 문을 67번째 줄과 바꾸면 예외가 발생할 때 가스를 좀 더 적게 사용할 수 있습니다.

6. StagedCrowdSale.sol에서 Stage struct의 cap은 uint256으로 선언하는 것이 올바릅니다.

```

17   struct Stage {
18     uint128 cap;
19     uint128 maxPurchaseLimit;
20     uint128 minPurchaseLimit;
21     uint128 weiRaised; // stage's weiAmount raised
22     uint32 startTime;
23     uint32 endTime;
24     bool kyc;
25   }

```

StageCrowdsale.sol

Stage struct에서 Stage.cap 을 구할 때는 StageCrowdsale.sol:59 처럼 토큰 세일의 전체 캡인 cap(uint256)에서 _capRatios[i][j]를 곱하고 coeff를 나눕니다. 따라서 uint256 인 cap 을 uint128 인 Stage.cap 에 넣는 것은 올바르지 못합니다. 토큰 세일의 캡이 매우 커서 2^128 보다 큰 숫자일 경우에 잘못된 값이 Stage.cap 에 들어갈 수 있습니다. Stage.cap 의 자료형을 uint256 으로 하는 것을 권장 합니다.

7. StageCrowdSale.sol에서 weiAmount 구하는 공식이 잘못되었습니다.

```

147   // pre-calculate `toFund` with the period's cap
148   if (p.cap > 0) {
149     uint256 postWeiRaised = uint256(p.weiRaised).add(weiAmount);
150
151     if (postWeiRaised > p.cap) {
152       weiAmount = uint256(p.cap).sub(weiRaised);
153     }
154   }

```

StageCrowdsale.sol

StageCrowdSale.sol:152에서 weiAmount(투자할 이더리움의 양) 때문에 스테이지의 캡을 넘기게 되었을 때 p.cap에서 weiRaised 를 빼서 weiAmount 를 새롭게 구합니다. 여기서 p.cap 은 스테이지의 cap 이고 weiRaised 는 스테이지와 무관하게 세일 전체기간동안 모인 이더리움의 양입니다. 따라서 스테이지 캡때문에 weiAmount 를 조정하는거면 weiRaised 가 아닌 p.weiRasied (스테이지에서 모인 이더리움의 양)을 빼서 weiAmount 를 구해야 합니다.

8. BonusCrowdsale.sol 의 `setBonusesForTimes`, `setBonusesForAmounts`에서 Integer Overflow 가 발생할 수 있습니다.

```

56     function setBonusesForAmounts(uint128[] amounts, uint32[] values) public onlyOwner {
57         require(amounts.length == values.length);
58         for (uint i = 0; i + 1 < amounts.length; i++) {
59             require(amounts[i] > amounts[i+1]);
60         }
61
62         BONUS_AMOUNTS = amounts;
63         BONUS_AMOUNTS_VALUES = values;
64     }

```

BonusCrowdsale.sol

```

35     function setBonusesForTimes(uint32[] times, uint32[] values) public onlyOwner {
36         require(times.length == values.length);
37         for (uint i = 0; i + 1 < times.length; i++) {
38             require(times[i] < times[i+1]);
39         }
40
41         BONUS_TIMES = times;
42         BONUS_TIMES_VALUES = values;
43     }

```

BonusCrowdsale.sol

`setBonusesForTimes`, `setBonusesForAmounts` 함수의 파라미터로 배열을 받는데 배열의 크기가 커서 `uint`에서 표현할 수 있는 수를 넘어버리면 Integer Overflow 가 발생할 수 있습니다. 이런 상황이 발생하기는 어렵지만 함수의 파라미터로 받는 배열의 크기의 최대값을 정하는게 안전합니다.

9. MultiHolderVault.sol 의 `close` 함수에 불필요한 `require` 문이 존재합니다.

```

20     function close() public onlyOwner {
21         require(state == State.Active);
22         require(wallet != 0x0 || initialized);
23
24         super.distribute(); // distribute ether to holders
25         super.close(); // transfer remaining ether to wallet
26     }

```

MultiHolderVault.sol

`MultiHolderVault` 는 `RefundVault` 를 상속받고 있습니다. `RefundVault` 의 생성자에서 `wallet` 의 주소가 `0x0` 이 아닌지 확인하는 `require` 문이 존재합니다. 그렇기 때문에 `wallet` 의 주소는 항상 `0x0` 이 아닙니다. 따라서 `MultiHolderVault.sol:22` 의 `require(wallet != 0x0 || initialized);` 는 항상 참입니다. 따라서 불필요한 `require` 문을 제거하는 것을 권장합니다.

10. MintableBaseCrowdsale.sol 와 ZeppelinBaseCrowdsale.sol 은 사용되지 않습니다.

AuditFullFeaturesCrowdsale.sol 에서 MintableBaseCrowdsale.sol 와 ZeppelinBaseCrowdsale.sol 을 사용하지 않습니다. 따라서 두 파일을 지우는 것을 권장합니다. 그리고 ZeppelinBaseCrowdsale.sol 에서 구현 내용은 *finishMinting* 함수가 없는 MintableBaseCrowdsale 컨트랙트와 똑같습니다. 어떤 역할을 하는 solidity 파일인지 명확하지 않습니다. 따라서 해당 파일을 지우는 것을 권장합니다.

11. ClaimableToken.sol 은 사용되지 않습니다.

ClaimableToken.sol 을 사용하는 곳이 없으므로 지우는 것을 추천합니다.

12. Locker.sol 의 생성자에서 *_beneficiaries*, *_ratios* 가 빈 배열이면 올바르게 초기화가 되지 않습니다.

```

118 function Locker(address _token, uint _coeff, address[] _beneficiaries, uint[] _ratios) public {
119     require(_token != address(0));
120     require(_beneficiaries.length == _ratios.length);
121
122     token = ERC20Basic(_token);
123     coeff = _coeff;
124     numBeneficiaries = _beneficiaries.length;
125
126     uint accRatio;
127
128     for(uint i = 0; i < numBeneficiaries; i++) {
129         require(_ratios[i] > 0);
130         beneficiaries[_beneficiaries[i]].ratio = _ratios[i];
131
132         accRatio = accRatio.add(_ratios[i]);
133     }
134
135     require(coeff == accRatio);
136 }
```

Locker.sol

Locker.sol 의 생성자에서 *_beneficiaries* 와 *_ratios* 의 배열 크기가 같은지는 120번째 줄에서 *require* 문을 이용해서 확인하고 있습니다. 하지만 두 배열이 빈 배열일 경우에도 배열 크기가 같으므로 *require* 문이 통과됩니다. 따라서 생성자가 예외없이 실행되지만 올바른 초기화가 아니므로 *require(_beneficiaries.length > 0)* 와 같은 조건을 119 번째 줄 다음에 추가하는 것을 추천합니다.



13. Locker.sol 의 *lock* 함수에서 *mapping releases*에 값을 저장할 때 가스 사용을 줄일 수 있습니다.

```

211     // create Release for the beneficiary
212     releases[_beneficiary].isStraight = _isStraight;
213
214     // copy array of uint
215     releases[_beneficiary].releaseTimes = new uint[](len);
216     releases[_beneficiary].releaseRatios = new uint[](len);
217
218     for (i = 0; i < len; i++) {
219         releases[_beneficiary].releaseTimes[i] = _releaseTimes[i];
220         releases[_beneficiary].releaseRatios[i] = _releaseRatios[i];
221     }

```

Locker.sol

Locker.sol의 215~220줄처럼 *release mapping*에 *releaseTimes*와 *releaseRatios*를 저장할 때 *new uint[](len)*을 이용해서 배열의 크기를 미리 정의하고 for loop 을 이용해 값을 대입할 수 있습니다. 하지만 이렇게 storage 크기를 미리 잡고 값을 변경하는 것보다 한번에 배열을 대입하는 것이 가스를 더 적게 사용합니다.

Line 215: *releases[_beneficiary].releaseTimes = _releaseTimes;*

Line 216: *releases[_beneficiary].releaseRatios = _releaseRatios;*

이렇게 초기화 시키는 것이 더 효율적이므로 위처럼 변경드리는 것을 추천합니다.

14. Locker.sol 의 *lock* 함수의 파라미터인 *_releaseTimes*, *_releaseRatios* 가 빈 배열 인지 확인해야합니다.

```

183     function lock(address _beneficiary, bool _isStraight, uint[] _releaseTimes, uint[] _releaseRatios)
184     external
185     onlyOwner
186     onlyState(State.Init)
187     onlyBeneficiary(_beneficiary)
188     {
189         require(!locked[_beneficiary]);
190         require(_releaseRatios.length == _releaseTimes.length);
191
192         uint i;
193         uint len = _releaseRatios.length;
194
195         // finally should release all tokens
196         require(_releaseRatios[len - 1] == coeff);
197
198         // check two array are ascending sorted
199         for(i = 0; i < len - 1; i++) {
200             require(_releaseTimes[i] < _releaseTimes[i + 1]);
201             require(_releaseRatios[i] < _releaseRatios[i + 1]);
202         }
203
204         // 2 release times for straight locking type
205         if (_isStraight) {
206             require(len == 2);
207         }

```

Locker.sol

Locker.sol 의 `lock` 함수에서 `_releaseTimes` 와 `_releaseRatios` 배열의 길이가 같은지는 검사하지만 empty array 인지는 검사하지 않습니다. 만약 빈배열일 경우에 Locker.sol:196 의 `_releaseRatios[len - 1]` 을 수행할 때 `len - 1` 에서 Integer Underflow 가 발생하게 됩니다. 따라서 189번 째줄 다음에 `require(_releaseRatios.length > 0)` 조건을 추가하는 것을 추천합니다.

15. AuditFullFeaturesToken.sol 의 `init` 함수 파라미터인 `args` 의 길이를 확인해야합니다.

```

54     function init(bytes32[] args) public {
55         uint _startTime = uint(args[0]);
56         uint _endTime = uint(args[1]);
57         uint _rate = uint(args[2]);
58         uint _coeff = uint(args[3]);
59         uint _cap = uint(args[4]);
60         uint _goal = uint(args[5]);
61         uint _lockerRatio = uint(args[6]);
62         uint _crowdsaleRatio = uint(args[7]);
63         address _vault = address(args[8]);
64         address _locker = address(args[9]);
65         address _nextTokenOwner = address(args[10]);

```

AuditFullFeaturesToken.sol

`init` 함수가 실행될 때 파라미터로 받은 `bytes32[] args` 를 55~65줄에 걸쳐서 이용합니다. 이때 `args` 배열의 길이가 11이라고 가정하고 `args[0] ~ args[10]` 에 `index` 를 하드코딩해서 접근합니다. 예외처리를 올바르게 하기 위해서 `init` 함수의 최상단에 `require(args.length == 11)` 을 넣는 것을 추천합니다.

16. AuditFullFeaturesToken.sol 은 `MiniMeToken` 을 상속할 필요가 없습니다.

```

7  contract AuditFullFeaturesToken is MiniMeToken, BurnableMiniMeToken, NoMintMiniMeToken {
8      function AuditFullFeaturesToken(address _tokenFactory)
9          MiniMeToken(
10             _tokenFactory,
11             0x0, // no parent token
12             0, // no snapshot block number from parent
13             "For Audit", // Token name
14             18, // Decimals
15             "FA", // Symbol
16             true // Enable transfers
17         ) {}
18     }

```

AuditFullFeaturesToken.sol

`AuditFullFeaturesToken` 은 `MiniMeToken`, `BurnableMiniMeToken`, `NoMintMiniMeToken` 을 상속받고 있습니다.
`BurnableMiniMeToken`, `NoMintMiniMeToken` 은 각각 `MiniMeToken` 을 상속받고 있습니다. 그렇기 때문에
`AuditFullFeaturesToken` 이 다시 `MiniMeToken` 을 상속받을 필요가 없습니다.

17. **`AuditFullFeaturesCrowdsale` 는 `BaseCrowdsale`, `KYCCrowdsale` 을 상속받을 필요가 없습니다.**

`AuditFullFeaturesCrowdsale` 는 contracts/base/crowdsale 경로에 있는 모든 `Crowdsale` 컨트랙트를 상속받고 있습니다. 하지만 다른 모든 컨트랙트들은 `BaseCrowdsale` 를 상속하고 있습니다. 따라서 `AuditFullFeaturesCrowdsale` 는 `BaseCrowdsale` 을 명시적으로 상속할 필요가 없습니다. 마찬가지로 `KYCCrowdsale` 은 `StageCrowdsale` 이 상속받고 있기 때문에 AuditFullFeaturesCrowdsale.sol:13 처럼 명시적으로 `KYCCrowdsale` 을 상속받을 필요가 없습니다.

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.

