May 13th- May 27th , 2022

Krust Universe Pte. Ltd. Security Audit Report

version 1.1

Fanto Smart Contract Audit

Audited by SOOHO

SOOHO

목차

1. 개요

‘개요’ 파트에서는 SOOHO 보안 감사 프로세스와 보안 감사에 사용된 방법론을 서술합니다. 분석 내용을 이해하시는 데에 참고하시길 바랍니다.

1.1 시작하기 전에

• 본 문서는 블록체인 보안 전문업체 SOOHO에서 진행한 취약점 검사를 바탕으로 작성한 문서로, 스마트 컨트랙트의 보안 취약점의 발견에 초점을 두고 있습니다. 추가적으로 코드 퀄리티 및 코드 라이센스 위반 사항 등에 대해서도 논의합니다.

• 본 문서는 코드의 유용성, 코드의 안정성, 비즈니스 모델의 적합성, 비즈니스의 법적인 규제, 계약의 적합성, 버그 없는 상태에 대해 보장하거나 서술하지 않습니다. 감사 문서는 논의 목적으로만 사용됩니다.

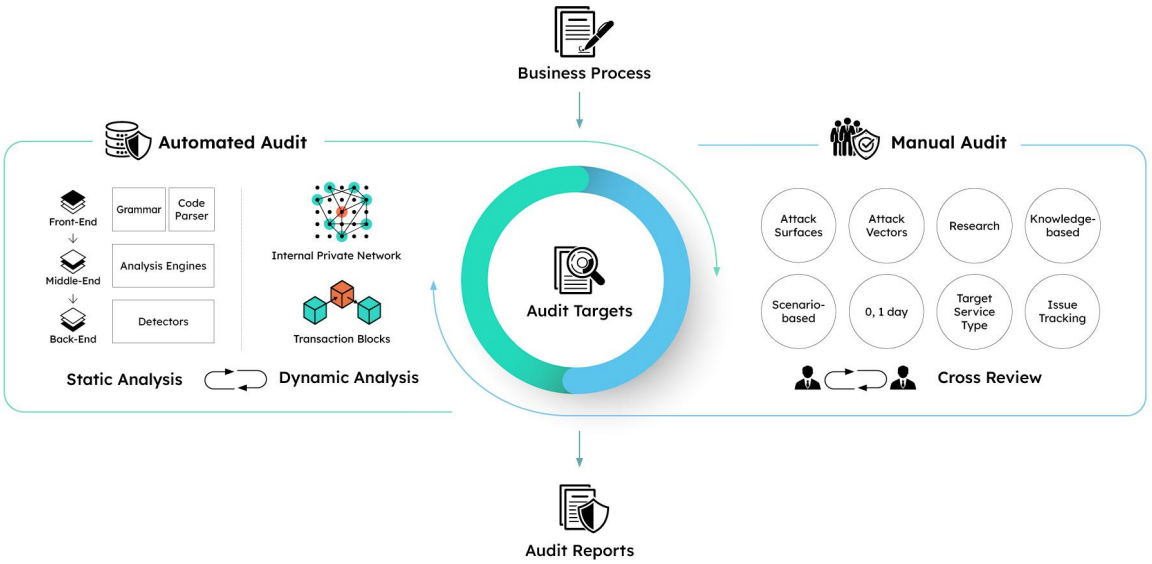
• SOOHO는 회사 정보가 대외비 이상의 성격을 가짐을 인지하고 사전 승인 없이 이를 공개하지 않습니다.

• SOOHO는 업무 수행 과정에서 취득한 일체의 회사 정보를 누설하거나 별도의 매체를 통해 소장하지 않습니다.

• SOOHO는 스마트 컨트랙트 분석에 최선을 다하였음을 밝히는 바입니다.

1.2 SOOHO 보안 감사 프로세스

SOOHO는 자동 분석(Automated Audit)과 수동 분석(Manual Audit)의 두 가지 감사 방법론을 적용하여 더욱 완벽한 블록체인 보안 감사를 진행합니다.



자동 분석은 정적 분석(Static analysis)과 동적 분석(Dynamic analysis) 사이 상호 협력적 분석을 통해 다양한 공격을 정확하고 빠르게 찾아내며 높은 감사 퀄리티를 보장합니다. SOOHO의 자체 분석기는 정적 분석을 통해 고객사 컨트랙트 코드의 문법을 분석(parsing)하여 의미 추론, 변수 추적, 경로 탐색을 진행함으로써 조건을 검증합니다. 정적 분석에서는 SOOHO 자체 테스트 네트워크에서 실제 동작을 통한 분석과 퍼징(Fuzzing), 콘콜릭 실행(Concolic Execution)을 통해 더욱 정교한 분석을 진행합니다.

수동 분석에서는 SOOHO의 보안 감사 전문가 그룹이 다양한 보안 및 도메인 지식을 활용하여 고객사의 프로젝트를 직접 검증합니다. 보다큰 리스크를 내포하는 코드를 중점적으로 분석하고, 파트너 사가 의도한 대로 코드가 작성되었는지 살피며 접근 권한의 관리가 올바르게 기능하고 있는지 검사합니다. 보안 감사 전문가들은 복잡한 공격 시나리오나 최근 보안 이슈를 처리하며 자동 분석을 상호 보완하여 감사의 완성도를 높입니다. 또한, 전문가 사이의 교차 검증을 통해 더욱 정교한 보안 감사 결과물을 제공합니다.

위와 같이 다양한 방법론을 이용하여 리스크 검증을 진행함으로써 파트너 사의 컨트랙트를 알려진 취약점 (1-day)과 0-day 취약점의 위협으로부터 안전하게 만들어줍니다.

발견된 취약점은 심각도 척도를 기준으로 하여 등급이 매겨집니다. 심각성 척도는 OWASP의 Impact & Likelihood 기반 리스크 평가 모델을 기반으로 정해졌습니다.

2. 분석 내용

'분석 내용' 파트에서는 Fanto Smart Contract 보안 감사 결과에 대한 전반적인 요약과 발견된 취약점의 구체적인 내용을 서술합니다. 발견된 모든 취약점에 대해 개선하는 것을 권장드리고 앞으로도 꾸준한 코드 감사를 통하여 서비스의 안정을 도모하시길 바랍니다.

2.1 분석 대상

Project	fanto-contract
# of Files	30
# of Target	30
# of Lines	2547
File Tree	fanto-core └── contracts ├── creator │ └── CreatorFactory.sol ├── farm │ └── FantoFarm.sol ├── interfaces │ ├── ICTMiningTreasury.sol │ ├── IERC20.sol │ ├── IFantoCallee.sol │ ├── IFantoERC20.sol │ ├── IFantoFactory.sol │ ├── IFantoPair.sol │ ├── IFantoRouter.sol │ ├── IMiningTreasury.sol │ └── IWKLAY.sol ├── libraries │ ├── FantoLibrary.sol │ ├── FantoMath.sol │ ├── SafeMath.sol │ ├── TransferHelper.sol │ └── UQ112x112.sol ├── mocks │ ├── FantoVestingSampleV2.sol │ └── MockToken.sol

```
├── swap
│   ├── FantoBasePair.sol
│   ├── FantoFactory.sol
│   ├── FantoPair.sol
│   └── FantoRouter.sol
├── token
│   ├── CT
│   │   ├── CTMiningTreasury.sol
│   │   └── CreatorToken.sol
│   ├── FTN
│   │   ├── FantoToken.sol
│   │   └── MiningTreasury.sol
│   └── WKLAY.sol
└── vesting
    ├── FantoAirdrop.sol
    ├── FantoVestingWallet.sol
    └── FantoVestingWalletV2.sol
```

2.2 분석 결과 요약

심각도 등급	취약점 수
Critical	■■
High	■■
Medium	■
Low	■■
Note	■■■

#	설명	심각도 등급	상태
SH-001	누구나 router를 설정할 수 있습니다.	Critical	Resolved
SH-002	누구나 ftn을 설정할 수 있습니다.	Critical	Resolved
SH-003	재진입 공격을 통하여 beneficiary가 lockup 기간 완료 전에 토큰을 모두 가져갈 수 있습니다.	High	Resolved
SH-004	WKLAY와 Proxy Payable Contract의 상호작용이 실패할 수 있습니다.	High	Resolved
SH-005	setEndBlock에 의해 farming 잔고가 고갈 될 수 있습니다.	Medium	Resolved
SH-006	고정 소수점은 계산 오류를 일으킬 수 있습니다.	Low	Resolved
SH-007	Deflationary 토큰에 대해서 Router가 올바르게 작동하지 않을 수 있습니다.	Low	Acknowledged
SH-008	디버그 코드는 삭제 되어야 합니다.	Note	Resolved
SH-009	빈 조건문 블록은 삭제 되어야 합니다.	Note	Resolved

SH-010	무의미한 SafeMath의 도입으로 가스비가 낭비되고 있습니다.	Note	

주요 감사 포인트

Fanto는 Fanto Team에서 개발한 ERC20 Token과 AMM DEX, Vesting Contract 그리고 Farming Contract로 이루어져 있습니다. 따라서 SOOHO Audit은 해당 컨트랙트 코드가 Reenterancy와 같은 일반적인 스마트 컨트랙트 보안 취약점은 없는지, 제공된 문서에 명세된 내용을 수행할 수 있도록 구현되었는지, Gas의 낭비가 존재하지 않는지 등을 중점적으로 감사하였습니다.

단, 관리자의 내부 해킹을 비롯해 컨트랙트 외부 서버 환경의 안정성은 고려하지 않았습니다. 본 보고서에서는 언급하지 않았지만 노드 자체에 대한 보안 검증과 외부 연동되는 서비스에 대해서도 검토하기를 제안합니다. 분석은 대상 프로젝트에 포함된 컨트랙트의 기능 안정성에 관한 것입니다.

2.3 분석 결과

Critical [Resolved] 누구나 router를 설정할 수 있습니다.

File Name	CTMiningTreasury.sol
File Location	/contracts/token/CT/CTMiningTreasury.sol
SHA1	5ebe213abd895b9098dea8361584044fc20f7fa6

```
32    function setRouter(IFantoRouter router_) public {
33          require(address(router) == address(0), "router is already set");
34          router = router_;
35    }
```

Details

CTMiningTreasury의 setRouter에는 제한자가 없어 누구나 router를 설정할 수 있습니다.

Mitigation

setRouter에 onlyOwner 조건을 주거나, Initializer에서 router를 설정하여 임의의 접근을 차단하십시오.

```
32    function setRouter(IFantoRouter router_) public onlyOwner {
33          require(address(router) == address(0), "router is already set");
34          router = router_;
35    }
```

Update

Fanto 팀은 본 문서에 제안된 대로 onlyOwner 제한자를 추가하여 문제를 해결하였습니다.

Critical [Resolved] 누구나 ftn을 설정할 수 있습니다.

File Name	MiningTreasury.sol
File Location	/contracts/token/FTN/MiningTreasury.sol
SHA1	7a6bff530068f5136f8c1db19809e71c5c2b2e39

```
28    function setFtn(ERC20Upgradeable ftn_) public {
29          require(address(ftn) == address(0), "ftn is already set");
30          ftn = ftn_;
31    }
```

Details

MiningTreasury setFtn에는 제한자가 없어 누구나 ftn을 설정할 수 있습니다.

Mitigation

setFtn에 onlyOwner 조건을 주거나, Initializer에서 ftn를 설정하여 임의의 접근을 차단하십시오.

```
28    function setFtn(ERC20Upgradeable ftn_) public onlyOwner {
29          require(address(ftn) == address(0), "ftn is already set");
30          ftn = ftn_;
31    }
```

Update

Fanto 팀은 본 문서에 제안된 대로 onlyOwner 제한자를 추가하여 문제를 해결하였습니다.

High [Resolved] 재진입 공격을 통하여 beneficiary가 lockup 기간 완료 전에 토큰을 모두 가져갈 수 있습니다.

File Name	FantoVestingWalletV2.sol
File Location	contracts/vesting/FantoVestingWalletV2.sol
SHA1	4ad5cf118d77d1c1a352908a9936b14aee811bea

```
97    function claim(uint256 _amount) checkStatus(status == VestingStatus.ACTIVE)
      virtual onlyBeneficiary public {
98        require(_amount > 0, "FantoVesting: `_amount` is 0");
99
100       uint256 _claimable = getClaimableAmount();
101       require(_amount <= _claimable, "FantoVesting: insufficient funds");
102
103       token.transfer(beneficiary, _amount * 10 ** 18);
104       claimedAmount = claimedAmount + _amount;
105
106       emit Claimed(_amount);
107   }
```

Details

FantoVestingV2의 claim 함수가 올바른 Checks-Effects-Interactions Pattern을 따르지 않아, 악의적인 token의 경우, beneficiary가 lockup 기간 완료 전에 토큰을 모두 가져갈 수 있습니다.

예를 들어, 악의적인 token이 transfer 함수 내에서 FantoVestingV2의 잔액이 getClaimableAmount()보다 크면 claim을 호출하도록 되어 있고, getClaimableAmount()가 100이며, FantoVestingV2의 잔액이 1000이라고 가정합시다.

beneficiary가 claim을 호출한다면, transfer에서 토큰이 이동하고, claimedAmount가 업데이트 되기 전에 다시 transfer내에서 claim이 호출되어 getClaimableAmount()에 비해서 훨씬 많은 토큰을 편취할 수 있습니다.

Mitigation

Checks-Effects-Interactions Pattern에 맞추어, 먼저 claimedAmount를 업데이트 한 후 transfer를 호출하십시오.

Update

Fanto 팀은 본 문서에 제안된 대로 Checks-Effects-Interactions Pattern에 맞추어, 먼저 claimedAmount를 업데이트 한 후 transfer를 호출하도록 수정하여 문제를 해결하였습니다.

High [Resolved] WKLAY와 Proxy Payable Contract의 상호작용이 실패할 수 있습니다.

File Name	WKLAY.sol
File Location	/contracts/token/WKLAY.sol
SHA1	2976992c1b68a9e6c6a705e11750fe16a22b462d

```
26    function withdraw(uint256 wad) public {
27        require(balanceOf[msg.sender] >= wad);
28        balanceOf[msg.sender] -= wad;
29        payable(msg.sender).transfer(wad);
30        emit Withdrawal(msg.sender, wad);
31    }
```

Details

WKLAY의 withdraw에서 transfer를 사용하면, 콜의 가스비가 2300으로 고정되어 Proxy Payable Contract와 상호작용 시에 실패할 수 있습니다.

transfer는 ether를 지정된 address로 보내는 콜을 생성하는데, 이때, 해당 Internal Call의 가스비를 2300으로 제한합니다. 이것은 포크 이전까지는 문제가 없었습니다. 하지만 Constantinople 포크로 인하여, SLOAD 명령어의 가스비가 오름에 따라서, Proxy Contract 호출의 가스비가 2300 이상으로 상승하게 되어 Proxy Contract는 WKLAY transfer를 호출할 수 없게 되었으며, 이는 서비스 운영에 큰 제약으로 작용합니다.

Mitigation

transfer 대신 call을 사용하십시오.

Update

Fanto 팀은 본 문서에 제안된 대로 transfer 대신 call을 사용하여 문제를 해결하였습니다.

Medium [Resolved] setEndBlock에 의해 farming 잔고가 고갈 될 수 있습니다.

File Name	FantoFarm.sol
File Location	/contracts/farm/FantoFarm.sol
SHA1	983892045e90f9d386762e44fbc4a94d84a5c771

```
114   // Set reward end block. You can set the end of the reward.
115   function setEndBlock(uint256 _endBlock) public {
116         require(msg.sender == address(miningTreasury), "not mining treasury");
117         require(block.number < _endBlock, "fund: too late, the farm is closed");
118
119         endBlock = _endBlock;
120
121         emit UpdateEndBlock(_endBlock);
122   }
```

Details

FantoFarm에 setEndBlock은 Farming 보상이 끝나는 시점을 지정하는 함수입니다. 초기에 fund 함수를 통해서 예치된 ERC20 잔액을 블록당 보상액으로 나누어 endBlock을 구합니다. 그런데, setEndBlock을 이용해서 endBlock을 임의로 조정하면, 보상기간이 남았음에도 잔고가 고갈되어 보상이 지급되지 않을 수 있습니다.

Mitigation

endBlock 조정 시에 잔액에 맞추어 rewardPerBlock을 조정하거나, rewardPerBlock과 남은 블록 수의 곱이 잔고를 넘지 못하도록 제한하십시오.

Update

Fanto 팀은 setEndBlock 함수를 삭제하여 오직 fund 함수 만을 통해서 endBlock이 설정되게 하는 방식으로 문제를 해결하였습니다.

Low [Resolved] 고정 소수점은 계산 오류를 일으킬 수 있습니다.

File Name	FantoVestingWalletV2.sol
File Location	contracts/vesting/FantoVestingWalletV2.sol
SHA1	4ad5cf118d77d1c1a352908a9936b14aee811bea

```
44  function prepare(address _distributor, address _beneficiary, uint256 _amount,
                     uint256 _duration, address _token)
46          checkStatus(status == VestingStatus.CREATED) virtual public onlyOwner {
47      require(_beneficiary != address(0),
                "FantoVesting: `_beneficiary` is zero address(0)");
48      require(_duration > 0, "FantoVesting:  `_duration` is 0");
49      // otherwise, claimer can claim all amount after first unlock period
50      require(_amount > 0, "FantoVesting: `_amount` is 0");
51      require(_amount >= _duration,
                "FantoVesting: _amount must be greater than _duration");
52
53      token = IERC20(_token);
54      duration = _duration;
55      beneficiary = _beneficiary;
56      initialVestingAmount = _amount;
57      unlockUnit = initialVestingAmount / duration;
58      remainder = initialVestingAmount - (unlockUnit * duration);
59      token.transferFrom(_distributor, address(this),
                           initialVestingAmount * 10 ** 18);
60      status = VestingStatus.PREPARED;
61
62      emit Prepared(_distributor, beneficiary, initialVestingAmount, duration,
                      _token);
63  }
```

Details

59번째 줄의 token을 이체하는 구문에 토큰 액수를 계산하는 과정을 보면, initialVestingAmount * 10 ** 18로 계산하는 데, 이 때 10**18은 token의 decimal 값을 의미하는데, 이것을 고정하는 경우, token의 decimal이 18이 아닌 경우 계산 오류를 일으킬 수 있습니다.

Mitigation

10**18로 고정된 값을 10 ** token.decimal() 의 값으로 수정해 주십시오.

Update

Fanto 팀은 본 문서에 제안된 대로 10**18로 고정된 값을 10 ** token.decimal() 의 값으로 수정하는 방식으로 문제를 해결하였습니다.

Low [Acknowledged] Deflationary 토큰에 대해서 Router가 올바르게 작동하지 않을 수 있습니다.

File Name	FantoRouter.sol
File Location	contracts/swap/FantoRouter.sol
SHA1	7a47e0d90a49d9949222f045ae82932972ad427d

Details

현재 FantoRouter 구현체는 transfer의 파라미터로 들어간 amount 값 만큼 잔액이 전송된다는 전제 하에 작성되었습니다. 하지만, Deflationary 토큰의 경우, transfer가 발생할 때, 전송 요청액의 일부를 수수료로 소각하므로, 전송된 금액과 요청한 금액이 다릅니다. 따라서, 전제가 맞지 않으므로 Router가 올바르게 작동하지 않을 수 있습니다.

Mitigation

PancakeSwap 구현체 등을 참조하여 Deflationary Token에 대해서도 올바르게 작동하도록 구현하십시오.

Update

Fanto 팀은 Deflationary Token을 포함된 유동성 풀에 추가하지 않을 것이기 때문에, 본 이슈를 수정하지 않기로 결정하였습니다.

Note **[Resolved] 디버그 코드는 삭제 되어야 합니다.**

File Name	MiningTreasury.sol
File Location	contracts/token/FTN/MiningTreasury.sol
SHA1	7a6bff530068f5136f8c1db19809e71c5c2b2e39

```
10    import "hardhat/console.sol";
```

Details

hardhat/console.sol 파일은 디버그용 파일입니다. 프로덕션 환경에서는 삭제되어야 합니다.

Mitigation

위의 Import 문을 삭제하십시오.

Update

Fanto 팀은 디버그 코드를 삭제하는 것을 통해 문제를 해결하였습니다.

Note **[Resolved] 빈 조건문 블록은 삭제 되어야 합니다.**

File Name	FantoVestingWalletV2.sol
File Location	/contracts/vesting/FantoVestingWalletV2.sol
SHA1	4ad5cf118d77d1c1a352908a9936b14aee811bea

```
123    function getNextUnlock() checkStatus(status == VestingStatus.ACTIVE)
       virtual public view returns (uint256) {
124        require(block.timestamp < end,
                   "FantoVesting: there's no remaining unlock time");
125
126        if (block.timestamp <= start) {
127            return start + UNLOCK_PERIOD;
128        }
129        if (block.timestamp >= end) {
130
131        }
132        return start + ((block.timestamp - start) / UNLOCK_PERIOD + 1)
                  * UNLOCK_PERIOD;
133    }
```

Details

getNextUnlock의 129에서 131번째 줄은 빈 조건문 블록입니다. 무의미하므로 삭제되어야 합니다.

Mitigation

빈 조건문 블록을 삭제하십시오.

Update

Fanto 팀은 빈 조건문 블록을 삭제하는 것을 통해 문제를 해결하였습니다.

Note **[Resolved] 무의미한 SafeMath의 도입으로 가스비가 낭비되고 있습니다.**

File Name	FantoFarm.sol
File Location	/contracts/farm/FantoFarm.sol
SHA1	983892045e90f9d386762e44fbc4a94d84a5c771

```
18    contract FantoFarm is ERC20Upgradeable {
19          using SafeMath for uint256;
```

Details

Solidity 0.8에서 Overflow와 Underflow를 Runtime에서 차단하도록 수정되었습니다. Klaytn에서도 이제는 Solidity 0.8을 지원하므로, Overflow/Underflow를 막기 위하여 SafeMath를 도입하는 것은 가스의 낭비입니다.

Mitigation

SafeMath의 사용을 중지하고, 일반적인 연산자로 연산을 대체하십시오.

Update

Fanto 팀은 본 문서에서 제안한 대로 SafeMath의 사용을 중지하고, 일반적인 연산자로 연산을 대체하는 것을 통해 문제를 해결하였습니다.

✔ (Verified) SWC-101

Details

정수형 데이터에 대한 오버플로우와 언더플로우 문제는 발생하지 않는 것을 확인하였습니다.

✔ (Verified) SWC-104

Details

호출 데이터에 대한 예외 처리가 포함되어 있습니다.

✔ (Verified) SWC-108

Details

컨트랙트 클래스에 대한 설계가 잘 되어 있습니다.

✔ (Verified) SWC-113

Details

DoS는 발생하지 않습니다.

✔ (Verified) SWC-118

Details

생성자 이름은 모두 온전하게 작성되어 있습니다.

2.4 결론

Fanto 팀에서 개발한 Fanto 컨트랙트는 이해하기 쉽게 명명되고 용도와 쓰임에 따라 잘 설계되어 있습니다. 대부분 모범 사례를 따르고 있습니다. 컨트랙트들은 매우 잘 설계되었고 그 구현 또한 훌륭하였음을 강조하고 싶습니다.

코드 검사 결과 발견된 이슈는 **Critical 2개, High 2개, Medium 1개, Low 2개, Note 3개** 입니다. 발견된 10개 이슈 중 9개 이슈가 팀에 의해 해소되었음을 알려드립니다. 꾸준한 코드 감사를 통해 컨트랙트의 안정을 도모하고 잠재적인 취약점에 대한 분석을 하는 것을 추천드립니다.

About SOOHO

SOOHO는 블록체인 생태계의 안전성과 신뢰도를 높이기 위한 기술을 연구하고 제공하고자 시작하였습니다. SOOHO는 자체적으로 개발한 취약점 분석기와 오픈 소스 분석기로 스마트 컨트랙트 취약점을 검증합니다. 더하여, SOOHO의 보안팀은 Defcon, Nuit du Hack, 화이트햇, SamsungCTF 등 국내외의 해킹 대회에서 수상하고, 보안분야 박사 학위와 같은 학문적 배경을 가지는 등 우수한 해킹 실력과 경험을 가진 보안 전문 인력들로 구성되어 있습니다. SOOHO의 전문 전문가들은 알려진 1-DAY 취약점부터 0-DAY 취약점으로부터 고객사의 스마트 컨트랙트를 보호하고자 합니다.

Contact us :)

Twitter: SOOHO_AUDIT

E-mail: audit@sooho.io

Website: https://audit.sooho.io/