HAECHI AUDIT

Key Protocol

Smart Contract Security Analysis

Published on : July 14, 2022

Version v2.0

HAECHI AUDIT

Smart Contract Audit Certificate

Key Protocol

Security Report Published by HAECHI AUDIT
v2.0 July 14, 2022

Auditor : Andy Koo

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	4	4	-	-	-
Minor	2	2	-	-	-
Tips	-	-	-	-	-

TABLE OF CONTENTS

6 Issues (0 Critical, 4 Major, 2 Minor) Found. All issues have been resolved.

ABOUT US

HAECHI AUDIT은 디지털 자산이 가져올 금융 혁신을 믿습니다. 디지털 자산을 쉽고 안전하게 만들기 위해 HAECHI AUDIT은 '보안'과 '신뢰'라는 가치를 제공합니다. 그로써 모든 사람이 디지털 자산을 부담없이 활용할 수 있는 세상을 꿈꿉니다.

HAECHI AUDIT은 글로벌 블록체인 업계를 선도하는 HAECHI LABS의 대표 서비스 중 하나로, 스마트 컨트랙트 보안 감사 및 개발을 전문적으로 제공합니다.

다년간 블록체인 기술 연구 개발 경험을 보유하고 있는 전문가들로 구성되어 있으며, 그 전문성을 인정받아 블록체인 기술 기업으로는 유일하게 삼성전자 스타트업 육성 프로그램에 선정된 바 있습니다. 또한, 이더리움 재단과 이더리움 커뮤니티 펀드로부터 기술 장려금을 수여받기도 하였습니다.

대표적인 클라이언트 및 파트너사로는 카카오 자회사인 Ground X, LG, 한화, 신한은행 등이 있으며, Sushiswap,1inch, Klaytn, Badger와 같은 글로벌 블록체인 프로젝트와도 협업한 바 있습니다. 지금까지 약 300여곳 이상의 클라이언트를 대상으로 가장 신뢰할 수 있는 스마트 컨트랙트 보안감사 및 개발 서비스를 제공하였습니다.

문의 : audit@haechi.io
웹사이트 : audit.haechi.io

INTRODUCTION

본 보고서는 Key Protocol 스마트 컨트랙트의 보안을 감사하기 위해 작성되었습니다. HAECHI AUDIT 는 스마트 컨트랙트의 구현 및 설계가 공개된 자료에 명시한 것처럼 잘 구현이 되어있고, 보안상 안전한지에 중점을 맞춰 감사를 진행했습니다.

CRITICAL	Critical 이슈는 광범위한 사용자가 피해를 볼 수 있는 치명적인 보안 결점으로 반드시 해결해야 하는 사항입니다.
MAJOR	Major 이슈는 보안상에 문제가 있거나 의도와 다른 구현으로 수정이 필요한 사항입니다.
MINOR	Minor 이슈는 잠재적으로 문제를 발생시킬 수 있으므로 수정이 요구되는 사항입니다.
TIPS	Tips 이슈는 수정했을 때 코드의 사용성이나 효율성이 더 좋아질 수 있는 사항입니다.

HAECHI AUDIT는 발견된 모든 이슈에 대하여 개선하는 것을 권장합니다. 이어지는 이슈 설명에서는 코드를 세부적으로 지칭하기 위해서 {파일 이름}#{줄 번호}, {컨트랙트 이름}#{함수/변수 이름} 포맷을 사용합니다. 예를 들면, *Sample.sol:20*은 Sample.sol 파일의 20번째 줄을 지칭하며, *Sample#fallback()* 는 Sample 컨트랙트의 fallback() 함수를 가리킵니다 보고서 작성을 위해 진행된 모든 테스트 결과는 Appendix에서 확인 하실 수 있습니다.

SUMMARY

Audit에 사용된 코드는 GitHub (https://github.com/cryptohiveteam/key-contract/tree/5941e9c327e0240b8a614168984e1d0ccc14e109)에서 찾아볼 수 있습니다. Audit에 사용된 코드의 마지막 커밋은 "5941e9c327e0240b8a614168984e1d0ccc14e109"입니다. 조치 완료 코드의 마지막 커밋은 "be9439a7b88aca4615be5694f9eabb717bc3bde2"입니다.

Issues HAECHI AUDIT에서는 Critical 이슈 0개, Major 이슈 4개, Minor 이슈 2개를 발견하였습니다. 발견된 이슈는 모두 조치가 완료되었습니다.

Severity	Issue	Status
⚠ MAJOR	lock 상태를 확인하는 조건이 잘못 적용되었습니다.	(Resolved - v2.0)
⚠ MAJOR	safeTrasfer/safeTransferFrom을 통한 veNFT 수신이 revert됩니다.	(Resolved - v2.0)
⚠ MAJOR	unlock시 positionId 값이 올바르지 않습니다.	(Resolved - v2.0)
⚠ MAJOR	UnlockDrop 컨트랙트와 PangeaLpDepositor 컨트랙트의 상호작용이 올바르지 않습니다.	(Resolved - v2.0)
MINOR	불필요한 safeApprove 함수가 존재합니다.	(Resolved - v2.0)
MINOR	safeTransfer/safeTransfer가 아닌 transfer/transferFrom을 사용합니다.	(Resolved - v2.0)

OVERVIEW

Contracts subject to audit

- FeeDistributor.sol
- KeyMinter.sol
- KeyToken.sol
- StakingRewards.sol
- TokenLocker.sol
- IKlayswap.sol
- ILPStaker.sol
- IPangeLpDepositor.sol
- IPangeaAuction.sol
- IPangeaLockDrop.sol
- KlayswapLPBurnerLib.sol
- PangeaLockDrop.sol
- PublicSale.sol
- TokenVesting.sol
- UnlockDrop.sol
- KeyFixedPoint.sol
- KeyFullMath.sol
- KeyPangeaVoter.sol
- LpDepositToken.sol
- LpDepositorHelper.sol
- LpRewards.sol
- PangeaLpDepositor.sol
- VeDepositor.sol

Key Protocol Smart contract에는 다음과 같은 권한이 있습니다.

- Owner
- Minter

각 권한의 제어에 대한 명세는 다음과 같습니다.

Role	Functions
Owner	❖ *FeeDistributor#whitelistToken* ❖ *FeeDistributor#dewhitelistToken* ❖ *KeyMinter#getReady* ❖ *KeyMinter#removeLocker* ❖ *KeyMinter#KeyToken* ❖ *KeyMinter#setEmergencySkip* ❖ *KeyToken#setMinter* ❖ *KeyToken#removeMinter* ❖ *PangeaLockDrop#delegateToAuction* ❖ *PangeaLockDrop#setKeyAmount* ❖ *PangeaLockDrop#claimStoneToStaker* ❖ *PangeaLockDrop#withdraw* ❖ *PangeaLockDrop#setPoolAddress* ❖ *PangeaLockDrop#claimStoneToAuction* ❖ *PangeaLockDrop#claimStoneToLockdrop* ❖ *PangeaLockDrop#setKeyAddresses* ❖ *PangeaLockDrop#setPangeaAddresses* ❖ *PangeaLockDrop#setTokenAddresses* ❖ *PangeaLockDrop#setTime* ❖ *PangeaLockDrop#setPoolAddress* ❖ *PublicSale#setTime* ❖ *TokenVesting#revoke* ❖ *UnlockDrop#addRewardPool* ❖ *UnlockDrop#setTime* ❖ *UnlockDrop#setPangeaPool* ❖ *UnlockDrop#setRewardKeyAmount* ❖ *UnlockDrop#migrateToPool* ❖ *UnlockDrop#claimStoneToLpDepositor* ❖ *UnlockDrop#claimKeyToLpDepositor* ❖ *KeyPangeaVoter#getPoolAddresses* ❖ *LpDepositToken#mint* ❖ *LpDepositToken#burn* ❖ *PangeaLpDepositor#setAddresses* ❖ *VeDepositor#setAddresses*
Minter	❖ *KeyToken#mint*

FINDINGS

⚠ **MAJOR**

lock 상태를 확인하는 조건이 잘못 적용되었습니다.

(Found - v.1.0)

```
function lock(address _lpToken, uint8 _week, uint256 _amount) external {
        require(0 < _week && _week < 5);
        require(startsAt <= block.timestamp);
        require(block.timestamp < endsAt);
        require(isRewardPool[_lpToken]);

        IERC20(_lpToken).safeApprove(address(this), _amount);
        IERC20(_lpToken).safeTransferFrom(msg.sender, address(this), _amount);

        UserLockup storage _userLockup = userLockups[_lpToken][msg.sender];
        UserLockupSum storage _lpWeight = lockupSums[_lpToken];

        // if user already locked up
        if (_userLockup.locked == 0) {
            _lpWeight.weightLocked -= _weightOf(_userLockup.locked, _userLockup.week);
        }
        _userLockup.locked += _amount;
        _userLockup.week = _week;
        _lpWeight.weightLocked += _weightOf(_amount, _week);
}
```

[https://github.com/cryptohiveteam/key-contract/blob/5941e9c327e0240b8a614168984e1d0ccc14e109/contracts/launch/UnlockDrop.sol#L148]

Issue

UnlockDrop#lock() 함수는 사용자가 기존에 예치했던 토큰이 존재할 경우 기존 전체 weight에서 사용자의 기존 weight를 제거하고 이후 새로 추가합니다. 기존 예치 이력이 존재할 경우 *_userLockup.locked* 값은 0이 아니게 되므로 조건문의 적용이 잘못된 상태입니다. 또한, 전체 weight를 업데이트 할 때 *_amount* 가 아닌, 사용자의 전체 토큰 값을 더해 주어야 합니다.

Recommendation

조건문의 등호를 변경하고 전체 weight 업데이트시 사용자가 예치한 토큰양이 반영될 수 있도록 수정이 필요합니다.

```
function lock(address _lpToken, uint8 _week, uint256 _amount) external {
        require(0 < _week && _week < 5);
        require(startsAt <= block.timestamp);
        require(block.timestamp < endsAt);
        require(isRewardPool[_lpToken]);

        IERC20(_lpToken).safeApprove(address(this), _amount);
        IERC20(_lpToken).safeTransferFrom(msg.sender, address(this), _amount);

        UserLockup storage _userLockup = userLockups[_lpToken][msg.sender];
        UserLockupSum storage _lpWeight = lockupSums[_lpToken];

        // if user already locked up
        if (_userLockup.locked != 0) {
            _lpWeight.weightLocked -= _weightOf(_userLockup.locked, _userLockup.week);
        }
        _userLockup.locked += _amount;
        _userLockup.week = _week;
        _lpWeight.weightLocked += _weightOf(_userLockup.locked, _week);
}
```

[Modified code]

Update

UnlockDrop.sol 컨트랙트는 서비스에 사용되지 않을 예정입니다.

⚠ MAJOR

safeTrasfer/safeTransferFrom을 통한 veNFT 수신이 revert됩니다.

(Found - v.1.0)

```
function onERC721Received(
        address _operator,
        address _from,
        uint256 _tokenID,
        bytes calldata
    ) external returns (bytes4) {
        require(msg.sender == address(votingEscrow), "Can only receive Magma NFTs");
        uint256 amount = votingEscrow.locked__amount(_tokenID);
        uint256 end = votingEscrow.locked__end(_tokenID);

        if (tokenID == 0) {
            tokenID = _tokenID;
            unlockTime = end;
            keyVoter.setTokenID(tokenID);
            votingEscrow.safeTransferFrom(address(this), address(lpDepositor), _tokenID);
            _mint(_operator, amount);
            extendLockTime();
        } else {
            merge(_tokenID);
        }

        return bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"));
}


function merge(uint256 _tokenID) internal returns (bool) {
        require(tokenID != _tokenID, "Invalid token");
        address _owner = votingEscrow.ownerOf(_tokenID);
        require(_owner == msg.sender, "Not owner");
        uint256 end = votingEscrow.locked__end(_tokenID);

        uint256 amount;
        if (end <= block.timestamp) { // if lockup period is expired, token is transferred. so check
the delta of token balance
            votingEscrow.transferFrom(_owner, address(this), _tokenID);
            amount = token.balanceOf(address(this));
            votingEscrowDist.claim(_tokenID); // lock rebase amounti
            amount = token.balanceOf(address(this)) - amount;
        }

        amount += votingEscrow.locked__amount(_tokenID);
        votingEscrow.merge(_tokenID, tokenID);
        if (end > unlockTime) unlockTime = end;

        emit Merged(_owner, _tokenID, amount);

        _mint(_owner, amount);
        extendLockTime();

        return true;
}
```

[https://github.com/cryptohiveteam/key-contract/blob/5941e9c327e0240b8a614168984e1d0ccc14e109/contracts/pangea/VeDepositor.sol#L135]

Issue

VeDepositor#onERC721Received() 함수는 safeTransfer/safeTransferFrom을 통해 veNFT를 수신 받을 때 호출됩니다. 해당 함수에서 수신 받은 veNFT를 컨트랙트가 보유한 veNFT와 합치기 위해 *merge()*를 호출합니다. 이 때 msg.sender가 votingEscrow 주소이면서 동시에 수신된 veNFT의 owner이어야 합니다. 해당 조건을 동시에 만족할 수 없으므로 veNFT 전송은 revert 됩니다.

Recommendation

onERC721Received() 로 veNFT를 전송 받은 경우, owner에 의해 허용된 전송이므로 바로 merge될 수 있도록 구현할 수 있습니다.

Update

safeTransfer/safeTransferFrom을 통해 veNFT를 전달 받은 경우, owner가 컨트랙트의 주소인지 여부를 체크하는 로직을 추가하여 패치가 이루어졌습니다.

```
function onERC721Received(
        address _operator,
        address _from,
        uint256 _tokenID,
        bytes calldata
    ) external returns (bytes4) {
        require(address(votingEscrow) == msg.sender, "Can only receive Magma NFTs");
        uint256 amount = votingEscrow.locked__amount(_tokenID);
        uint256 end = votingEscrow.locked__end(_tokenID);

        if (tokenID == 0) {
            tokenID = _tokenID;
            unlockTime = end;
            keyVoter.setTokenID(tokenID);
            votingEscrow.safeTransferFrom(address(this), address(lpDepositor), _tokenID);
            _mint(_operator, amount);
            extendLockTime();
        } else {
            merge(_tokenID, _from);
        }

        return bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"));
}


function merge(uint256 _tokenID, address to) internal returns (bool) {
        require(tokenID != _tokenID, "Invalid token");
        address _owner = votingEscrow.ownerOf(_tokenID);
        require(_owner == msg.sender || _owner == address(this), "Not owner nor itself");
        uint256 end = votingEscrow.locked__end(_tokenID);

        uint256 amount;
        if (end <= block.timestamp) { // if lockup period is expired, token is transferred. so check
```

```
the delta of token balance
            votingEscrow.transferFrom(_owner, address(this), _tokenID);
            amount = token.balanceOf(address(this));
            votingEscrowDist.claim(_tokenID); // lock rebase amounti
            amount = token.balanceOf(address(this)) - amount;
        }

        amount += votingEscrow.locked__amount(_tokenID);
        votingEscrow.merge(_tokenID, tokenID);
        if (end > unlockTime) unlockTime = end;

        emit Merged(_owner, _tokenID, amount);

        _mint(to, amount);
        extendLockTime();

        return true;
}
```

[https://github.com/cryptohiveteam/key-contract/blob/be9439a7b88aca4615be5694f9eabb717bc3bde2/contracts/pangea/VeDepositor.sol#L135]

⚠ MAJOR

unlock시 positionId 값이 올바르지 않습니다.

(Found - v.1.0)

```
function unlock(address _lpToken) external returns (uint256 positionId) {
        // 비율에 맞는 stone, key
        claimRewardByUser(_lpToken);

        // 자신의 LP 토큰
        UserLockup storage _userLockup = userLockups[_lpToken][msg.sender];
        uint256 _weight = _weightOf(_userLockup.locked, _userLockup.week);
        UserLockupSum storage _lpWeight = lockupSums[_lpToken];
        uint256 _totalWeightNow = _lpWeight.weightLocked - _lpWeight.weightUnlocked;

        IConcentratedLiquidityPoolManager.Position memory position =
poolManager.positions(positionId);

        uint256 amount = KeyFullMath.mulDiv(position.liquidity, _weight, _totalWeightNow);

        (uint256 token0Amount, uint256 token1Amount) =
lpDepositor.removeLiquidity(poolInfos[_lpToken].positionId, uint128(amount));

        positionId = voter.mintAndDeposit(poolInfos[_lpToken].pangeaPool, TICK_LOWEST, TICK_LOWEST,
TICK_HIGHEST, TICK_HIGHEST, uint128(token0Amount), uint128(token1Amount), 0, EMPTY_ARR);
}
```

[https://github.com/cryptohiveteam/key-contract/blob/5941e9c327e0240b8a614168984e1d0ccc14e109/contracts/launch/UnlockDrop.sol#L274]

Issue

UnlockDrop#unlock() 함수는 positionId 를 burn할 수 있습니다. 이 과정에서 positionId 값이 0으로 설정되어 함수의 정상 동작이 불가합니다.

Recommendation

positionId가 함수의 인자 또는 스토리지에 저장되어 유효한 값이 될 수 있도록 수정할 수 있습니다.

Update

UnlockDrop.sol 컨트랙트는 서비스에 사용되지 않을 예정입니다.

⚠ MAJOR

UnlockDrop 컨트랙트와 PangeaLpDepositor 컨트랙트의 상호작용이 올바르지 않습니다.

(Found - v.1.0)

```
function migrateToPool(address _lpToken) external onlyOwner {
        (
            address token0,
            address token1,
            uint256 amount0,
            uint256 amount1
        ) = KlayswapLPBurnerLib.burnAll(_lpToken);

        //Lower & upper 계산하는 로직
        uint256 positionId = voter.mintAndDeposit(
            poolInfos[_lpToken].pangeaPool,
            TICK_LOWEST,
            TICK_LOWEST,
            TICK_HIGHEST,
            TICK_HIGHEST,
            uint128(amount0),
            uint128(amount1),
            0,
            EMPTY_ARR
        );
        lpDepositor.deposit(positionId);
    }
```

[https://github.com/cryptohiveteam/key-contract/blob/5941e9c327e0240b8a614168984e1d0ccc14e109/contracts/launch/UnlockDrop.sol#L200]

```
function _deposit(uint256 positionId) internal {
        IConcentratedLiquidityPoolManager.Position memory position =
poolManager.positions(positionId);
        address pool = address(position.pool);
        address _owner = poolManager.ownerOf(positionId);
        require(msg.sender == _owner);

        address gauge = gaugeForPool[pool];

        if (gauge == address(0)) {
            gauge = pangeaVoter.gauges(pool);
            if (gauge == address(0)) {
                gauge = pangeaVoter.createGauge(pool);
            }
            gaugeForPool[pool] = gauge;
            bribeForPool[pool] = pangeaVoter.bribes(gauge);
            poolManager.setApprovalForAll(gauge, true);
        }

        poolManager.transferFrom(_owner, address(this), positionId);
        uint256[] memory veIDs = new uint256[](1);
        veIDs[0] = tokenID;
        IGauge(gauge).deposit(address(this), positionId, veIDs);
        ILpDepositToken(lpDepositToken).mint(_owner, positionId);
        emit Deposited(_owner, pool, positionId);
    }
```

[https://github.com/cryptohiveteam/key-contract/blob/5941e9c327e0240b8a614168984e1d0ccc14e109/contracts/pangea/PangeaLpDepositor.sol#L290]

Issue

UnlockDrop#migrateToPool() 함수는 KlaySwap의 LP를 burn하고 이를 Pangea에 예치합니다. 이 과정에서 사용되는 *voter#mintAndDeposit()* 함수는 Pangea에 토큰을 예치하고 pool LP토큰을 수령하여 Gauge에 예치하는 기능을 합니다.
위 과정이 완료되면 *LpDepositor#deposit()* 함수를 호출합니다.
LpDepositor(PangeaLpDepositor)에는 노출된 deposit() 함수가 없어 호출이 불가합니다.
또한, *PangeaLpDepositor#_deposit()* internal 함수는 pool LP토큰을 Gauge에 예치하고 LpDepositToken을 발행해 주는 함수로, 이미 *voter#mintAndDeposit()*에 의해 gauge에 예치된 positionId를 전달하면 revert가 발생합니다.

Recommendation

UnlockDrop#migrateToPool() 함수 진행 과정에서 필요한 *LpDepositor#deposit()* 함수의 구현이 필요합니다.

Update

UnlockDrop.sol 컨트랙트는 서비스에 사용되지 않을 예정입니다.

MINOR

불필요한 safeApprove 함수가 존재합니다.

(Found - v.1.0)

```
function lock(address _lpToken, uint8 _week, uint256 _amount) external {
        require(0 < _week && _week < 5);
        require(startsAt <= block.timestamp);
        require(block.timestamp < endsAt);
        require(isRewardPool[_lpToken]);

        IERC20(_lpToken).safeApprove(address(this), _amount);
        IERC20(_lpToken).safeTransferFrom(msg.sender, address(this), _amount);

        UserLockup storage _userLockup = userLockups[_lpToken][msg.sender];
        UserLockupSum storage _lpWeight = lockupSums[_lpToken];

        // if user already locked up
        if (_userLockup.locked == 0) {
            _lpWeight.weightLocked -= _weightOf(_userLockup.locked, _userLockup.week);
        }
        _userLockup.locked += _amount;
        _userLockup.week = _week;
        _lpWeight.weightLocked += _weightOf(_amount, _week);
}
```

[https://github.com/cryptohiveteam/key-contract/blob/5941e9c327e0240b8a614168984e1d0ccc14e109/contracts/launch/UnlockDrop.sol#L148]

Issue

UnlockDrop#lock() 함수의 실행 과정에서 해당 컨트랙트의 주소를 approve하는 코드가 존재합니다. 해당 코드는 *UnlockDrop#lock()* 함수의 구현에 불필요한 코드이며 예상치 못한 오류를 야기할 수 있습니다.

Recommendation

“safeApprove” 코드를 제거하시길 권고 드립니다.

Update

UnlockDrop.sol 컨트랙트는 서비스에 사용되지 않을 예정입니다.

MINOR

safeTransfer/safeTransfer가 아닌 transfer/transferFrom을 사용합니다.

(Found - v.1.0)

Issue

일부 함수의 실행 과정에서 safeTransfer/safeTransferFrom이 아닌 transfer/transferFrom이 사용되고 있습니다.

Recommendation

다음의 코드에 대하여 safeTransfer/safeTransferFrom 으로의 변경을 권고드립니다.

```
PangeaLockDrop.sol#L117
PangeaLockDrop.sol#L144
PangeaLockDrop.sol#L171
PangeaLockDrop.sol#L219
PangeaLockDrop.sol#L233
UnlockDrop.sol#L191
```

[transfer/transferFrom 사용 코드]

Update

UnlockDrop.sol, *PangeaLockDrop.sol* 컨트랙트는 서비스에 사용되지 않을 예정입니다.

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 메인넷 상의 결함 등 발견되지 않은 문제들이 있을 수 있습니다. 안전한 스마트 컨트랙트를 작성하기 위해서는 발견된 문제들에 대한 수정과 충분한 테스트가 필요합니다.

End of Document