

HAECHI AUDIT

Stake.ly

Smart Contract Security Analysis

Published on : Sep 20, 2022

Version v1.1





HAECHI AUDIT

Smart Contract Audit Certificate



Stake.ly

Security Report Published by HAECHI AUDIT

v1.1 Sep 20, 2022 / 패치 리뷰, “FKlay” - “FKlay (바뀐토큰명 stKlay)” 수정

v1.0 Sep 5, 2022 / audit report

Auditor : Jinu Lee, Paul Kim

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	1	1	-	-	-
Minor	2	2	-	-	-
Tips	-	-	-	-	-

TABLE OF CONTENTS

3 Issues (0 Critical, 1 Major, 2 Minor, 0 Tips) Found

[TABLE OF CONTENTS](#)

[ABOUT US](#)

[INTRODUCTION](#)

[SUMMARY](#)

[OVERVIEW](#)

[FINDINGS](#)

[getSharesByKlay 연산 시 버림으로 인해 이용자가 미세한 이익을 얻을 수 있습니다.](#)

[rewardBase 연산 시 버림으로 인해 이용자가 미세한 이익을 얻을 수 있으며 이로 인해 이용자에게 지급되는 reward와 누적된 reward의 차이로 DoS가 발생할 수 있습니다.](#)

[unstakeForDeregister 함수를 사용해 NodeHandler를 비활성화 시키면 Unstake가 진행중인 요청은 처리되지 않습니다.](#)

[Fix](#)

[DISCLAIMER](#)

ABOUT US

HAECHI AUDIT은 디지털 자산이 가져올 금융 혁신을 믿습니다. 디지털 자산을 쉽고 안전하게 만들기 위해 HAECHI AUDIT은 '보안'과 '신뢰'라는 가치를 제공합니다. 그래서 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 400여곳 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 Stake.ly 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/stakely-protocol/stakely-mono/tree/v0.1-2022-08-16>)에서 찾아볼 수 있습니다. Audit에 사용된 코드의 마지막 커밋은 “b23607e01e6e9cdcae4521d54c12d007a90fb2b1”입니다.

Issues HAECHI AUDIT에서는 Major 이슈 1개, Minor 이슈 2개를 발견했습니다.

Severity	Issue	Status
 MINOR	getSharesByKlay 연산 시 버림으로 인해 이용자가 미세한 이익을 얻을 수 있습니다.	(Found - v1.0)
 MAJOR	rewardBase 연산 시 버림으로 인해 이용자가 미세한 이익을 얻을 수 있으며 이로 인해 이용자에게 지급되는 reward와 누적된 reward의 차이로 DoS가 발생할 수 있습니다.	(Found - v1.0)
 MINOR	unstakeForDeregister 함수를 사용해 NodeHandler를 비활성화 시키면 Unstake가 진행중인 요청은 처리되지 않습니다.	(Found - v1.0)

OVERVIEW

Contracts subject to audit

- ❖ BuyBack.sol
- ❖ FKlay.sol
- ❖ Fluid.sol
- ❖ FluidDistributor.sol
- ❖ FluidReservoir.sol
- ❖ NodeHandler.sol
- ❖ NodeManager.sol
- ❖ SFluid.sol
- ❖ Treasury.sol
- ❖ UnstakingReceiver.sol
- ❖ VestingWallet.sol
- ❖ WFKlay.sol

Fluid에는 다음과 같은 권한이 있습니다.

- ❖ onlyOwner
- ❖ onlyRole {ROLE_PAUSER,ROLE_NODE_INFO_SETTER, ROLE_FKLAY_SETTER, ROLE_FEE_MANAGER, ROLE_MIN_SETTER, ROLE_TREASURY_SETTER}
- ❖ onlyNodeManager
- ❖ onlyFKlay
- ❖ isNodeHandler

각 권한의 제어에 대한 명세는 다음과 같습니다.

Role	Functions
onlyOwner	<ul style="list-style-type: none">❖ <i>FluidReservoir#sendToVesting</i>❖ <i>BuyBack#setKlayFKlayPool</i>❖ <i>BuyBack#setFKlayFluidPool</i>❖ <i>BuyBack#burnFluid</i>❖ <i>BuyBack#emergencyWithdraw</i>❖ <i>FluidDistributor#setFluidAddress</i>❖ <i>FluidDistributor#addTarget</i>❖ <i>FluidDistributor#removeTarget</i>❖ <i>FluidDistributor#distributeFluid</i>❖ <i>FluidDistributor#emergencyFluidWithdraw</i>❖ <i>NodeHandler#updateGcStakedAmount</i>

	<ul style="list-style-type: none"> ❖ <i>NodeHandler#setGcRewardAddress</i> ❖ <i>NodeHandler#setNodeManagerAddress</i> ❖ <i>NodeHandler#setUnstakingReceiver</i> ❖ <i>SFluid#config</i> ❖ <i>Treasury#withdrawKlay</i> ❖ <i>Treasury#withdrawToken</i> ❖ <i>UnstakingReceiver#setHandler</i>
onlyRole	<ul style="list-style-type: none"> ❖ <i>FKlay#pause</i> ❖ <i>FKlay#unpause</i> ❖ <i>NodeManager#unstakeForRebalacing</i> ❖ <i>NodeManager#unstakeForDeregister</i> ❖ <i>NodeManager#claimAndRestake</i> ❖ <i>NodeManager#setFklayAddress</i> ❖ <i>NodeManager#setFeeRate</i> ❖ <i>NodeManager#setFeeDistribution</i> ❖ <i>NodeManager#setNodeLockupTime</i> ❖ <i>NodeManager#setUnstakeSplitThreshold</i> ❖ <i>NodeManager#config</i> ❖ <i>NodeManager#addNode</i> ❖ <i>NodeManager#setNodeName</i> ❖ <i>NodeManager#setNodeActive</i> ❖ <i>NodeManager#setNodeRewardAddress</i> ❖ <i>NodeManager#setTreasuryAddress</i>
onlyNodeManager	<ul style="list-style-type: none"> ❖ <i>FKlay#increaseTotalStaking</i> ❖ <i>NodeHandler#unstake</i> ❖ <i>NodeHandler#claimUnstakedTo</i> ❖ <i>NodeHandler#claimReward</i>
onlyFKlay	<ul style="list-style-type: none"> ❖ <i>NodeManager#stake</i> ❖ <i>NodeManager#unstake</i> ❖ <i>NodeManager#claim</i>
isNodeHandler	<ul style="list-style-type: none"> ❖ <i>UnstakingReceiver#withdraw</i>

FINDINGS

● MINOR

getSharesByKlay 연산 시 버림으로 인해 이용자가 미세한 이익을 얻을 수 있습니다.

(Found - v.1.0)

```
function _getSharesByKlay(uint256 klayAmount)
    private
    view
    returns (uint256)
{
    if (totalStaking == 0) return 0;

    return (totalShares * klayAmount) / totalStaking;
}
```

[<https://github.com/stakely-protocol/stakely-mono/blob/v0.1-2022-08-16/packages/contract/contracts/FKlay.sol#L462>]

```
function _stake(address user, uint256 amount)
    private
    validAddress(user)
    nonZero(amount)
    nonReentrant
{
    uint256 sharesToMint = _getSharesByKlay(amount);
    if (sharesToMint == 0) {
        // Ahh... fresh mint!
        sharesToMint = amount;
    }

    _mintShares(user, sharesToMint);
    totalStaking += amount;

    nodeManager.stake{value: amount}(user);

    emit Transfer(ZERO_ADDRESS, user, amount);
}

/**
 * @notice Requests unstaking of staked tokens
 * @param amount amount to unstake
 * Emits a `Transfer` event from msgSender to zero address
 */
function _unstake(uint256 amount) private nonZero(amount) nonReentrant {
    address user = _msgSender();

    uint256 sharesToBurn = _getSharesByKlay(amount);
    _burnShares(user, sharesToBurn);

    totalStaking -= amount;

    nodeManager.unstake(user, amount);

    emit Transfer(user, ZERO_ADDRESS, amount);
}
```

[<https://github.com/stakely-protocol/stakely-mono/blob/v0.1-2022-08-16/packages/contract/contracts/FKlay.sol#L325>]

Issue

FKlay (바뀐토큰명 stKlay)는 클레이를 스테이킹 하면 발행되는 토큰입니다. 스테이킹 보상은 클레이로 쌓이며, 이를 비율로 관리하기 위해 shares와 totalstaking 두가지 값을 관리합니다. totalShares 비율에 맞게 발행된 총 FKlay (바뀐토큰명 stKlay) 수량이며 totalStaking은 {스테이킹 클레이 + 리워드 클레이} 수량입니다.

stake/unstake 함수에서 shares - klay의 비율을 계산할 때 _getSharesByKlay 함수를 사용합니다. _getSharesByKlay 함수는 연산 시 나눗셈에서 버림이 발생하기 때문에 입력한 값 보다 결과 값이 같거나 적게 반환 됩니다.

- stake
 - mint 되는 값은 _getSharesByKlay(amount)로 값이 버림 연산 되어 이용자가 미세한 손실을 입을 수 있습니다.
 - stake 함수를 보면 _getSharesByKlay(amount) 값이 0일 때는 sharesToMint 값을 amount로 설정하는데 최초의 스테이킹 상황을 가정하고 코드를 구현했을 것이라 추측됩니다. 하지만 reward가 지급 된 적이 있거나 스테이킹이 한번 이상 된 상태여서 totalStaking 값이 0이 아닌 상황에서도 amount 값이 작으면 $(totalShares * klayAmount) / totalStaking$ 연산의 결과가 0이 되어 _getSharesByKlay 함수의 반환 값이 0일 수 있습니다. 이 상황에서는 이용자가 스테이킹 하는 Klay 수량을 FKlay (바뀐토큰명 stKlay)로 변환해보면 0 이하이기 때문에 FKlay (바뀐토큰명 stKlay)를 발행해주면 안되지만, 스테이킹 하는 Klay 수량 만큼 FKlay (바뀐토큰명 stKlay)를 발행하게 되기 때문에 이용자는 **shares - klay 비율을 무시하고 FKlay (바뀐토큰명 stKlay)를 발행할 수 있습니다.**
- unstake: burn 되는 값은 _getSharesByKlay(amount)로 값이 버림 연산 되었지만 실제 unstake 보상은 amount 입력 값을 그대로 사용하기 때문에 **이용자가 미세한 이득을 볼 수 있습니다.**

Recommendation

- stake 함수에서 _getSharesByKlay 함수의 반환 값을 사용해 0과 비교하지 않고 totalShares가 0인지 비교하도록 코드를 수정합니다.

```
function _stake(address user, uint256 amount)
    private
    validAddress(user)
    nonZero(amount)
    nonReentrant
{
```

```

uint256 sharesToMint;
if (totalShares == 0 ) {
    // Ahh... fresh mint!
    sharesToMint = amount;
}else {
    sharesToMint = _getSharesByKlay(amount);
    if (sharesToMint == 0) {
        revert("FKlay: insufficient amount");
    }
}

_mintShares(user, sharesToMint);
totalStaking += amount;

nodeManager.stake{value: amount}(user);

emit Transfer(ZERO_ADDRESS, user, amount);
}

```

- _getSharesByKlayRoundUp 함수를 구현해 unstake 함수에서 사용하면 이용자가 손해를 보도록 기능을 구현할 수 있습니다.

⚠ MAJOR

rewardBase 연산 시 버림으로 인해 이용자가 미세한 이익을 얻을 수 있으며 이로 인해 이용자에게 지급되는 reward와 누적된 reward의 차이로 DoS가 발생할 수 있습니다.

(Found - v.1.0)

```
function stake(address user, uint256 amount)
    external
    nonReentrant
    whenNotPaused
{
    require(amount > 0, "zero amount");
    if (totalSupply() > 0) {
        rewardRate +=
            ((address(this).balance - lastKlayBalance) * 1e18) /
            totalSupply();
    }
    lastKlayBalance = address(this).balance;
    rewardBase[user] += (amount * rewardRate) / 1e18;

    _mint(user, amount);

    bool success = fluid.transferFrom(msg.sender, address(this), amount);
    require(success, "transferFrom failed");

    emit Stake(msg.sender, user, amount, balanceOf(user));
}
```

[<https://github.com/stakely-protocol/stakely-mono/blob/v0.1-2022-08-16/packages/contract/contracts/SFluid.sol#L87>]

```
function _claimReward(address user) private {
    (uint256 rate, uint256 reward) = _claimableReward(user);

    if (reward > 0) {
        rewardRate = rate;
        rewardBase[user] += reward;
        lastKlayBalance = address(this).balance - reward;

        cumulativeClaimedReward[user] += reward;
        payable(user).sendValue(reward);
        emit ClaimReward(user, reward);
    }
}
```

[<https://github.com/stakely-protocol/stakely-mono/blob/v0.1-2022-08-16/packages/contract/contracts/SFluid.sol#L280>]

```
function _claimableReward(address user)
    internal
    view
    returns(uint256 rate, uint256 reward)
{
    uint256 totalSupply = totalSupply();
    if (totalSupply == 0) return (rewardRate, 0);
```

```

    rate =
        rewardRate +
        (guardedSub(address(this).balance, lastKlayBalance) * 1e18) /
        totalSupply;

    reward = guardedSub((balanceOf(user) * rate) / 1e18, rewardBase[user]);
}

```

[<https://github.com/stakely-protocol/stakely-mono/blob/v0.1-2022-08-16/packages/contract/contracts/SFluid.sol#L255>]

Issue

SFluid의 stake 함수에서 rewardBase[user] 더해주는 과정에서 곱셈 이후 나눗셈 연산을 거치게 됩니다. 문제는 여기서 나눗셈 연산으로 인한 round down이 발생하면서 rewardBase가 실제보다 적게 증가하게 됩니다.

이후 unstake 과정에서 _claimReward를 호출하게 됩니다. _claimReward를 보면, _claimableReward에서 리턴받은 reward를 컨트랙트 밸런스에서 빼주는 것으로 lastKlayBalance를 계산하게 됩니다.

reward의 연산 과정을 수식으로 표현하면 $\text{balanceOf}(\text{user}) * \text{rate} / 1\text{e}18 - \text{rewardBase}[\text{user}]$ 입니다. 여기서 중요한 점은 stake 함수가 호출 되었을 때, rewardBase가 내림 되었기 때문에 당연히 reward는 실제 reward보다 큰 값이 리턴되게 됩니다.

위와 같은 원리로 lastKlayBalance를 계산하는 과정에서 컨트랙트가 가진 Balance에서 실제 reward보다 큰 값이 뺄셈 연산 되면서, lastKlayBalance가 잘못 계산되거나 integer underflow로 인한 DoS가 발생할 수 있습니다.

Recommendation

stake함수에서 rewardBase[user] 계산 시 Round Up 해주는 코드 추가를 권장합니다.

● MINOR

unstakeForDeregister 함수를 사용해 NodeHandler를 비활성화 시키면 Unstake가 진행중인 요청은 처리되지 않습니다.

(Found - v.1.0)

```
function unstakeForDeregister(uint256 nodeId)
    external
    nonReentrant
    onlyRole(ROLE_NODE_INFO_SETTER)
{
    INodeHandler nodeHandler = nodeInfos[nodeId].nodeHandler;
    _setActive(nodeId, false);

    //slither-disable-next-line unused-return
    nodeHandler.unstake(address(this), nodeHandler.protocolNetStaking());
}
```

[<https://github.com/stakely-protocol/stakely-mono/blob/v0.1-2022-08-16/packages/contract/contracts/NodeManager.sol#L298>]

Issue

NodeManager에 구현된 unstakeForDeregister 함수는 특정 노드의 모든 것을 언스테이킹하고 비활성화 하기 위한 함수입니다. NodeInfo에 저장된 노드의 Active 정보를 false로 바꾸고, NodeManager를 이용해 스테이킹 된 모든 값(protocolStaking)에서 언스테이킹 중인 값(unstakingRequested)을 뺀 만큼 unstake 요청을 합니다. unstake 요청이 되었다면, 7일 이후 claimUnstakedTo 함수를 실행해야 unstake 신청 금액을 수령할 수 있습니다.

NodeHandler의 claimUnstakedTo 함수를 보면 onlyNodeManager modifier를 사용해 NodeManager만 호출할 수 있게 권한을 설정했습니다. NodeManager가 NodeHandler의 claimUnstakedTo 함수를 호출하는 경우는 아래와 같이 두 가지가 존재합니다.

1. ROLE_NODE_INFO_SETTER 권한을 가진 Account가 claimAndRestake 함수를 호출하는 경우: claimUnstakedTo 함수의 인자로 address(this) 값을 전달함
노드의 Active 여부와 관계 없이 claimUnstakedTo 함수를 호출함
2. FKlay (바뀐토큰명 stKlay) 컨트랙트를 통해 claim 함수를 호출하는 경우:
claimUnstakedTo 함수의 인자로 claim 함수 인자 값을 전달함
노드가 Active 상태여야 claimUnstakedTo 함수를 호출함

만약 Fluid 컨트랙트를 사용하는 이용자의 unstake 요청이 존재하는 NodeHandler가 비활성화 된다면 이용자는 claimUnstakedTo 함수를 호출할 수 없게 되고 unstake 신청 금액을 수령할 수 없게 됩니다.

Recommendation

비활성화 된 노드에서도 보상을 수령할 수 있게 수정합니다.

Fix

Last Update: 2022.09.19

#ID	Title	Severity	Status
1	getSharesByKlay 연산 시 버림으로 인해 이용자가 미세한 이익을 얻을 수 있습니다.	Minor	Fixed
2	rewardBase 연산 시 버림으로 인해 이용자가 미세한 이익을 얻을 수 있으며 이로 인해 이용자에게 지급되는 reward와 누적된 reward의 차이로 DoS가 발생할 수 있습니다.	Major	Fixed
3	unstakeForDeregister 함수를 사용해 NodeHandler를 비활성화 시키면 Unstake가 진행중인 요청은 처리되지 않습니다.	Minor	Fixed

Fix Comment

[01] [PR-149](#) Fixed

[02] [PR-169](#) Fixed

[03] [PR-169](#) Fixed

NodeHandler를 비활성화 시킬 때 NodeManager에서 isUnstakingBlocked 값을 업데이트 하고, 클레임 가능한 기간 동안 기다렸다가 unstakeForDeregister 함수를 호출하도록 패치했습니다.

ROLE_NODE_INFO_SETTER가 클레임 가능한 기간¹(~ +2 weeks)동안 기다렸다가 unstakeForDeregister 함수를 직접 호출해야 하기 때문에 이에 대한 관리 혹은 자동화가 필요합니다.

1

<https://github.com/klaytn/klaytn/blob/442f01f650eaae5c489ddd2476a665e77a48e790/contracts/cnstaking/CnStakingContract.sol#L548>

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