

HAECHI AUDIT

Pangea

Smart Contract Security Analysis

Published on : July 22, 2022

Version v2.0





HAECHI AUDIT

Smart Contract Audit Certificate



Pangea

Security Report Published by HAECHI AUDIT
v2.0 July 22, 2022

Auditor : Louis Noh

Andy Koo

A handwritten signature in black ink.

A handwritten signature in black ink.

Executive Summary

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

TABLE OF CONTENTS

2 Issues (0 Critical, 1 Major, 1 Minor) Found.

TABLE OF CONTENTS

ABOUT US

INTRODUCTION

AUDIT TIMELINE

SUMMARY

OVERVIEW

FINDINGS

PoolRouter에서 경유하는 Pool의 토큰 주소 계산이 제대로 이루어지지 않습니다.

Factory는 Upgradable하므로, setPrice가 여러번 실행되어 가격을 조작 할 수 있습니다.

rangeFeeGrowth() 함수의 반환 값이 경우에 따라 포지션의 feeGrowthInside0/1 값보다 작을 수 있습니다.

TickIndex 컨트랙트의 adjust 함수는 사용자가 입력한 lowerTick, upperTick의 범위를 변경합니다.

PREVIOUS AUDIT FINDINGS

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. 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 Pangea 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.

AUDIT TIMELINE

| Date | Event |
|------------|---|
| 2022/04/15 | 1차 Audit 시작 (Concentrated Liquidity Pool & veTokenomics) (대상 : 8ee0d8707e4b34d6f37220b7d30e89d8b97827ae) |
| 2022/05/13 | 1차 Audit 보고서 전달 |
| 2022/05/16 | 2차 Audit 시작 (Bootstrap code) (대상 : c615b693905984239d294bd39cee24c1a6df416e) |
| 2022/05/27 | 2차 Audit 보고서 전달 |
| 2022/06/27 | 3차 Audit 시작 (Concentrated Liquidity Pool & Airdrop) (대상 : 2406236b28374114a3e845f460358a38e2bb1e28) |
| 2022/07/18 | 3차 Audit 보고서 전달 |
| 2022/07/21 | Follow up meeting |

SUMMARY

Audit에 사용된 코드는 GitHub

(<https://github.com/HAECHI-LABS/audit-Pangea2/tree/2406236b28374114a3e845f460358a38e2bb1e28>)에서 찾아볼 수 있습니다. Audit에 사용된 코드의 마지막 커밋은 “2406236b28374114a3e845f460358a38e2bb1e28”입니다.

Issues HAECHI AUDIT에서는 Critical 이슈 0개, Major 이슈 1개, Minor 이슈 1개를 발견하였으며 수정했을 때 코드의 사용성이나 효율성이 더 좋아질 수 있는 사항들을 2개의 Tips 카테고리로 나누어 서술하였습니다.

| Severity | Issue | Status |
|----------|--|-----------------------|
| ⚠ MAJOR | PoolRouter에서 경유하는 Pool의 토큰 주소 계산이 제대로 이루어지지 않습니다. | (Resolved - v2.0) |
| ● MINOR | Factory는 Upgradable하므로, setPrice가 여러번 실행되어 가격을 조작 할 수 있습니다. | (Resolved - v2.0) |
| 💡 TIPS | rangeFeeGrowth() 함수의 반환 값이 경우에 따라 포지션의 feeGrowthInside0/1 값보다 작을 수 있습니다. | (Acknowledged - v2.0) |
| 💡 TIPS | TickIndex 컨트랙트의 adjust 함수는 사용자가 입력한 lowerTick, upperTick의 범위를 변경합니다. | (Acknowledged - v2.0) |

OVERVIEW

Contracts subject to audit

- ❖ PangeaBatchable
- ❖ PangeaPermit
- ❖ PangeaERC72
- ❖ MasterDeployer
- ❖ Airdrop Distributor
- ❖ FixedPoint
- ❖ DyDxMath
- ❖ SwapHelperLib
- ❖ TickMath
- ❖ TickIndex
- ❖ Ticks
- ❖ FullMath
- ❖ SafeCast
- ❖ FeeLib
- ❖ UnsafeMath
- ❖ ConcentratedLiquidityPoolManager
- ❖ PoolRouter
- ❖ ConcentratedLiquidityPool
- ❖ PoolLogger
- ❖ ConcentratedLiquidityPoolFactory
- ❖ PoolFactoryLib
- ❖ IPoolFactoryCallee
- ❖ IPoolEventStruct
- ❖ IMasterDeployer
- ❖ IAirdropDistributor
- ❖ IProtocolFeeReceiver
- ❖ IPoolFlashCallback
- ❖ IPoolFeeInfo
- ❖ IConcentratedLiquidityPoolManager
- ❖ IConcentratedLiquidityPool
- ❖ IPoolLogger
- ❖ IWETH
- ❖ INFTDescriptor
- ❖ IAirdropPool
- ❖ IPool Factory

- ❖ LPAirdropCallee
- ❖ IPositionManager
- ❖ IPoolRouter
- ❖ IConcentratedLiquidityPoolFactory

Pangea Smart contract에는 다음과 같은 권한이 있습니다.

- ❖ Owner

각 권한의 제어에 대한 명세는 다음과 같습니다.

| Role | Functions |
|-------|---|
| Owner | <ul style="list-style-type: none">❖ <i>MasterDeployer#addToWhitelistFactory()</i>❖ <i>MasterDeployer#removeFromWhitelistFactory()</i>❖ <i>MasterDeployer#setProtocolFeeTo()</i>❖ <i>MasterDeployer#setAirdropDistributor()</i> |

FINDINGS

⚠ MAJOR

PoolRouter에서 경유하는 Pool의 토큰 주소 계산이 제대로 이루어지지 않습니다.

(Resolved - v.2.0)

```
function findTokenOut(address[] calldata path, address tokenIn) internal view returns (address tokenOut) {
    tokenOut = tokenIn == address(0) ? wETH : tokenIn;
    IConcentratedLiquidityPool pool;
    for (uint256 i = 0; i < path.length; i++) {
        pool = IConcentratedLiquidityPool(path[i]);
        tokenOut = pool.token0() == tokenIn ? pool.token1() : pool.token0();
    }
    return tokenOut;
}
```

[<https://github.com/HAECHI-LABS/audit-Pangea2/blob/2406236b28374114a3e845f460358a38e2bb1e28/contractv1/pool/PoolRouter.sol#L207>]

Issue

*PoolRouter#findTokenOut()*은 입력받은 *tokenIn*에 해당하는 *tokenOut*, 즉 swap받을 토큰의 주소를 반환합니다. 이때, 경유하고자 하는 pool의 첫번째 토큰과 *tokenIn*을 비교하게되면, 같은 같은 값만 비교하게 되므로, *tokenOut*에 swap받을 토큰이 반환되지 못할 수 있습니다. 가령, wKlay와 USDC를 swap한다고 하고, 경유하고자 하는 풀이 wKlay-DAI, DAI-USDC라고 한다면, *tokenIn*은 wKlay의 주소만 담고 있으므로, *tokenOut*으로 DAI가 반환됩니다.

Recommendation

*PoolRouter#findTokenOut()*에서 현재 pool의 토큰과 경유하고자 하는 pool의 토큰들과 비교하여, 경유하고자 하는 pool의 토큰의 주소로 업데이트되어야 합니다. 반복문에서 *tokenOut* 값이 *tokenIn*과의 비교가 아닌 *tokenOut* 값과 비교되어야 합니다.

```
function findTokenOut(address[] calldata path, address tokenIn) internal view returns (address tokenOut) {
    tokenOut = tokenIn == address(0) ? wETH : tokenIn;
    IConcentratedLiquidityPool pool;
    for (uint256 i = 0; i < path.length; i++) {
        pool = IConcentratedLiquidityPool(path[i]);
        tokenOut = pool.token0() == tokenOut ? pool.token1() : pool.token0();
    }
    return tokenOut;
}
```

[recommended patch]

Update

[[PR188](#)]을 통해 해당 이슈가 해결되었음을 확인하였습니다.

```
function findTokenOut(address[] calldata path, address tokenIn) internal view returns (address
tokenOut) {
    tokenOut = tokenIn == address(0) ? wETH : tokenIn;
    IConcentratedLiquidityPool pool;
    for (uint256 i = 0; i < path.length; i++) {
        pool = IConcentratedLiquidityPool(path[i]);
        tokenOut = pool.token0() == tokenOut ? pool.token1() : pool.token0();
    }
    return tokenOut;
}
```

[Patched code]

▣ MINOR

Factory는 Upgradable하므로, setPrice가 여러번 실행되어 가격을 조작 할 수 있습니다.

(Resolved - v.2.0)

```
function setPrice(uint160 _price) external {
    if (msg.sender != factory) revert NotAuthorized();
    TickMath.validatePrice(_price);
    price = _price;
}
```

[<https://github.com/HAECHI-LABS/audit-Pangea2/blob/2406236b28374114a3e845f460358a38e2bb1e28/contractv1/pool/ConcentratedLiquidityPool.sol#L142>]

Issue

*ConcentratedLiquidityPool#setPrice()*는 Pool이 deploy될 때, 가격을 설정하는 함수로 한번만 설정되어야 합니다. 그리고 *ConcentratedLiquidityPoolFactory* 컨트랙트에 의해서만 실행 될 수 있습니다. 그러나, *ConcentratedLiquidityPoolFactory* 컨트랙트는 Upgradable하게 구현되어있어, 이후에 업그레이드를 통해 *ConcentratedLiquidityPool#setPrice()*를 여러번 실행할 수 있습니다.

Recommendation

*ConcentratedLiquidityPool#setPrice()*가 한번만 호출될 수 있도록, *price*값이 0인지 아닌지 확인하는 과정이 필요합니다.

```
...
error AlreadySetPrice();
...

function setPrice(uint160 _price) external {
    if (msg.sender != factory) revert NotAuthorized();
    if (price != 0) revert AlreadySetPrice();
    TickMath.validatePrice(_price);
    price = _price;
}
```

[recommended patch]

Update

[[a7994](#)] commit을 통해 이슈가 해결되었음을 확인하였습니다.

```
function setPrice(uint160 _price) external {
    if (msg.sender != factory) revert NotAuthorized();
    if (price != 0) revert AlreadyPriceInitialized();
    TickMath.validatePrice(_price);
    price = _price;
}
```

[Patched code]

💡 TIPS

`rangeFeeGrowth()` 함수의 반환 값이 경우에 따라 포지션의 `feeGrowthInside0/1` 값보다 작을 수 있습니다.

```
function positionFees(uint256 positionId)
public
view
returns (
    uint256 token0amount,
    uint256 token1amount,
    uint256 feeGrowthInside0,
    uint256 feeGrowthInside1
)
{
    Position memory position = positions[positionId];

    (feeGrowthInside0, feeGrowthInside1) =
IConcentratedLiquidityPool(position.pool).rangeFeeGrowth(position.lower, position.upper);

    token0amount =
        FullMath.mulDiv(feeGrowthInside0 - position.feeGrowthInside0, position.liquidity,
FixedPoint.Q128) +
        position.feeOwed0;

    token1amount =
        FullMath.mulDiv(feeGrowthInside1 - position.feeGrowthInside1, position.liquidity,
FixedPoint.Q128) +
        position.feeOwed1;
}
```

[<https://github.com/HAECHI-LABS/audit-Pangea2/blob/2406236b28374114a3e845f460358a38e2bb1e28/contractv1/pool/ConcentratedLiquidityPoolManager.sol#L334>]

```
function rangeFeeGrowth(int24 lowerTick, int24 upperTick) public view returns (uint256
feeGrowthInside0, uint256 feeGrowthInside1) {
    int24 currentTick = TickMath.getTickAtSqrtRatio(price);

    Tick memory lower = ticks[lowerTick];
    Tick memory upper = ticks[upperTick];

    // Calculate fee growth below & above.
    uint256 _feeGrowthGlobal0;
    uint256 _feeGrowthGlobal1;
    {
        (uint256 remain0, uint256 remain1) = airdropGrowthRemain(liquidity);
        _feeGrowthGlobal0 = swapFeeGrowthGlobal0 + airdropGrowthGlobal0 + remain0;
        _feeGrowthGlobal1 = swapFeeGrowthGlobal1 + airdropGrowthGlobal1 + remain1;
    }

    uint256 feeGrowthBelow0;
    uint256 feeGrowthBelow1;
    uint256 feeGrowthAbove0;
    uint256 feeGrowthAbove1;
    if (lowerTick <= currentTick) {
        feeGrowthBelow0 = lower.feeGrowthOutside0;
        feeGrowthBelow1 = lower.feeGrowthOutside1;
    } else {
        feeGrowthBelow0 = _feeGrowthGlobal0 - lower.feeGrowthOutside0;
```

```

        feeGrowthBelow1 = _feeGrowthGlobal1 - lower.feeGrowthOutside1;
    }

    if (currentTick < upperTick) {
        feeGrowthAbove0 = upper.feeGrowthOutside0;
        feeGrowthAbove1 = upper.feeGrowthOutside1;
    } else {
        feeGrowthAbove0 = _feeGrowthGlobal0 - upper.feeGrowthOutside0;
        feeGrowthAbove1 = _feeGrowthGlobal1 - upper.feeGrowthOutside1;
    }

    feeGrowthInside0 = _feeGrowthGlobal0 - feeGrowthBelow0 - feeGrowthAbove0;
    feeGrowthInside1 = _feeGrowthGlobal1 - feeGrowthBelow1 - feeGrowthAbove1;
}

```

[<https://github.com/HAECHI-LABS/audit-Pangea2/blob/2406236b28374114a3e845f460358a38e2bb1e28/contractv1/pool/ConcentratedLiquidityPool.sol#L802>]

Issue

*ConcentratedLiquidityPoolManager#positionFees()*은 LP의 특정 tick 구간 내에 모인 fee를 LP가 제공한 Liquidity의 비율로 계산해 주는 함수입니다. 이 과정에서 *ConcentratedLiquidityPool#rangeFeeGrowth()*를 사용하여 현재까지 구간 내에 모인 fee를 조회합니다. 해당 함수의 반환값 *feeGrowthInside0/1* 포지션의 값 보다 크다는 가정으로 코드가 진행되고 있습니다. 그러나 swap 과정에서 현재 tick이 MAX_TICK, MIN_TICK 인 경우, *ConcentratedLiquidityPool#rangeFeeGrowth()* 값이 *position.feeGrowthInside{0, 1}* 보다 작아질 수 있습니다. 이 경우, integer underflow가 발생하여 함수 호출에 실패 할 수 있습니다. *ConcentratedLiquidityPoolManager#positionFees()* 함수가 실행되지 않는다면, *burn()*, *collect()* 함수가 실행되지 않아 유저가 자금을 인출 할 수 없게 됩니다.

Recommendation

비록, 해당 이슈가 유저의 자금을 잡글 수 있으나 TIPS레벨로 서술된 이유는 해치랩스에서 분석한 결과, 해당 이슈가 발생할 확률이 높지 않으며, 부득이한 경우 추가적으로 liquidity를 공급함으로써 자금을 인출 할 수 있기 때문입니다.
따라서, 프로토콜에서 최악의 경우를 가정하고 자금 인출이 우선이신 경우 *positionFees()* 가, *rangeFeeGrowth()* 값에 따라 다르게 작동하는 방식을 채택하는 것을 추천드립니다. 이는 비록 자금 인출에 대한 안정성은 확보하지만, 정산받아야하는 일부 수수료에 대해서는 받을 수 없게 되는 부작용이 있습니다.

Update

Uninitialized tick의 mint 이전에 `ConcentratedLiquidityPool#rangeFeeGrowth()`를 호출하면, mint 전/후의 `position.feeGrowthInside{0, 1}`에 차이가 발생할 수 있으므로, mint 이후에 포지션의 Growth fee를 계산하도록 변경이 이루어졌습니다. 또한, Growth fee 계산에 오류가 발생하더라도 Liquidity를 인출할 수 있도록 조건문을 만들어 Safeguard를 추가하였습니다.

[[PR191](#)]

```
function _mint(
    IConcentratedLiquidityPool pool,
    int24 lowerOld,
    int24 lower,
    int24 upperOld,
    int24 upper,
    uint128 amount0Desired,
    uint128 amount1Desired,
    uint256 minLiquidity,
    uint256 positionId
) internal returns (uint256 _positionId) {
    (lowerOld, lower, upperOld, upper) = pool.adjust(lowerOld, lower, upperOld, upper);

    uint128 liquidityMinted = uint128(
        pool.mint(
            IConcentratedLiquidityPoolStruct.MintParams({
                lowerOld: lowerOld,
                lower: lower,
                upperOld: upperOld,
                upper: upper,
                amount0Desired: amount0Desired,
                amount1Desired: amount1Desired
            })
        )
    );
    if (liquidityMinted < minLiquidity) revert TooLittleReceived();

    if (positionId == 0) {
        // We mint a new NFT.
        _positionId = nftCount.minted;
        (uint256 feeGrowthInside0, uint256 feeGrowthInside1) = pool.rangeFeeGrowth(lower,
upper);
        positions[_positionId] = Position({
            pool: address(pool),
            liquidity: liquidityMinted,
            lower: lower,
            upper: upper,
            latestAddition: uint32(block.timestamp),
            feeGrowthInside0: feeGrowthInside0,
            feeGrowthInside1: feeGrowthInside1,
            feeOwed0: 0,
            feeOwed1: 0
        });
        mint(msg.sender);
    }
}
```

[Patched _mint code]

```
function positionFees(uint256 positionId)
    public
    view
    returns (
        uint256 token0amount,
        uint256 token1amount,
        uint256 feeGrowthInside0,
        uint256 feeGrowthInside1
    )
{
    Position memory position = positions[positionId];

    (feeGrowthInside0, feeGrowthInside1) =
    IConcentratedLiquidityPool(position.pool).rangeFeeGrowth(position.lower, position.upper);

    token0amount = feeGrowthInside0 > position.feeGrowthInside0
        ? FullMath.mulDiv(feeGrowthInside0 - position.feeGrowthInside0, position.liquidity,
    FixedPoint.Q128) + position.feeOwed0
        : position.feeOwed0;

    token1amount = feeGrowthInside1 > position.feeGrowthInside1
        ? FullMath.mulDiv(feeGrowthInside1 - position.feeGrowthInside1, position.liquidity,
    FixedPoint.Q128) + position.feeOwed1
        : position.feeOwed1;
}

...

```

[Patched positionFees code]

💡 TIPS

TickIndex 컨트랙트의 `adjust` 함수는 사용자가 입력한 `lowerTick`, `upperTick`의 범위를 변경합니다.

```
function adjust(
    IConcentratedLiquidityPool pool,
    int24 lowerOld,
    int24 lower,
    int24 upperOld,
    int24 upper
)
external
view
returns (
    int24,
    int24,
    int24,
    int24
)
{
    (lower, upper) = _adjustLowerAndUpper(pool, lower, upper);
    bool needToInitLower = needToInitialize(pool, lower);

    if (!needToInitLower) {
        lowerOld = pool.ticks(lower).previousTick;
    } else if (atWrongPlace(pool, lower, lowerOld)) {
        lowerOld = findIndex(pool, lower, lowerOld);
    }

    if (atUpperNext(pool, lowerOld, upper)) {
        return (lowerOld, lower, lower, upper);
    }

    bool needToInitUpper = needToInitialize(pool, upper);

    if (!needToInitUpper) {
        upperOld = pool.ticks(upper).previousTick;
    } else if (atWrongPlace(pool, upper, upperOld)) {
        upperOld = findIndex(pool, upper, upperOld);
    }

    return (lowerOld, lower, upperOld, upper);
}
```

[<https://github.com/HAECHI-LABS/audit-Pangea2/blob/2406236b28374114a3e845f460358a38e2bb1e28/contractv1/libraries/TickIndex.sol#L13>]

Issue

일반적으로 사용자가 입력한 `lowerTick`, `upperTick`은 범위가 잘못되었을 경우에, `revert`를 내어 사용자에게 잘못된 tick의 범위를 입력했다는 것을 알립니다. 하지만, Pangea에서 `TickIndex#adjust()`에 의해 사용자가 입력한 범위가 잘못되었을 경우 이를 미세조정함으로써 올바른 Tick에 유동성을 제공하도록 합니다. 이는 사용자가 원하지 않는 연산일 수도 있습니다.

PREVIOUS AUDIT FINDINGS

💡 TIPS

(1차 audit)

Push를 하기 전에 중복여부 검사를 하지 않습니다.

```
function _registerFee(address token) internal {
    feeTokens.push(token);
    isFee[token] = true;
    rewardTokens.push(token);
    isReward[token] = true;
    _initSnapshot(token);
}
```

[<https://github.com/HAECHI-LABS/audit-Pangea/blob/8ee0d8707e4b34d6f37220b7d30e89d8b97827ae/contracts/bribe/Bribe.sol#L108>]

```
function _addTokenToDepositList(address token) private {
    depositTokensIndex[token] = depositTokens.length;
    depositTokens.push(token);
}
```

[<https://github.com/HAECHI-LABS/audit-Pangea/blob/8ee0d8707e4b34d6f37220b7d30e89d8b97827ae/contracts/bribe/Bribe.sol#L423>]

```
function _updateLatestRewards(address token) internal {
    if (isLatestRewards[token]) {
        return;
    }

    // LatestRewards is a round robin list
    if (latestRewards.length < MAXIMUM_LATEST_REWARDS) {
        latestRewards.push(token);
    } else {
        uint256 _currIndex = lIndex % MAXIMUM_LATEST_REWARDS;
        address _prev = latestRewards[_currIndex];
        isLatestRewards[_prev] = false;
        latestRewards[_currIndex] = token;
    }

    isLatestRewards[token] = true;
    lIndex++;
}
```

[<https://github.com/HAECHI-LABS/audit-Pangea/blob/8ee0d8707e4b34d6f37220b7d30e89d8b97827ae/contracts/bribe/Bribe.sol#L404>]

Issue

Bribe#_registerFee(), Bribe#_addTokenToDepositList(), Bribe#_updateLastestRewards()

는 입력으로 받은 token의 주소를 어떠한 중복 여부 검사를 수행하지 않고, push를 하고 있습니다. 이는 Contract로 하여금 예기치 못한 동작을 야기할 수 있습니다.

Update

세 함수 모두 앞단 로직에서 중복여부를 검사하고 있음을 확인하였습니다.

Bribe 컨트랙트가 프로젝트에 사용되지 않음에 따라 해당 이슈가 해결되었습니다.

TIPS

Concentrated Liquidity Pool 특성상, swap이 일어나는 가격과 시장 가격간 큰 차이가 발생할 수 있습니다.

(1차 audit)

Issue

Stable Coin에 대한 Concentrated Liquidity Pool을 제공하는 경우, 많은 유동성 공급자들이 게으른 유동성 문제에 대응하기 위해 특정 구간에만 유동성을 제공하여, 해당 구간에만 유동성이 쌓일 수 있습니다.

이러한 Concentrated Liquidity Pool 특성은 Pool에 Liquidity가 있더라도, 현재 시장 가격에 대한 Liquidity가 없다면 Pool에서 swap이 일어나는 가격과 시장 가격 간에 큰 차이를 야기할 수 있습니다. 가령, Stable Coin의 pegging이 깨져 해당 유동성이 집중된 가격 구간을 벗어나는 경우, 시장 가격의 변동성을 Pangea Concentrated Liquidity Pool에서 따라 가지 못할 수 있습니다. 이는 코드의 문제는 아니지만, 해당 리스크에 대해 인지하시길 바랍니다.

TIPS

sandwich/MEV 공격이 가능합니다.

(1차 audit)

Issue

큰 규모의 거래는 시장 가격을 낮추기 위한 사전 매각 혹은 거래 금액에 같은 양을 더하는 테일게이트 바이백에 의해서 샌드위치 공격에 당할 가능성이 있습니다. 이러한 공격은 손실로 이어질 수 있으며, 실제 얻는 이익이 본래 얻어야 할 이익보다 작을 수도 있습니다. 이러한 프론트 러닝 공격이 발생할 수 있음을 인지하시길 바랍니다.

Recommendation

위의 프론트 러닝 공격에 대비하기 위해 off chain에서 값을 계산하여야 함을 유의 하시기 바랍니다.

Update

해당 내용에 대해 인지하고 있으며, Slippage를 체크하고 revert하는 로직이 존재하는 상태입니다.

💡 TIPS

initPool()에서 중복된 풀의 생성에 대한 고려가 없습니다.

(2차 audit)

```
function initPool(address factory, InitializationParams calldata params)
external
returns (
    uint256 positionId,
    uint256 liquidity,
    uint256 remain0,
    uint256 remain1
)
{
    require(params.token0 < params.token1, "WRONG ORDER");
    require(params.amount0 > 0 && params.amount1 > 0, "INVALID AMOUNT");

    // [1] create pool
    address pool = masterDeployer.deployPool(factory, deployData(params)); // @check existing pool

    // [2] transfer token to poolManager
    relayToManager(params.token0, params.amount0);
    relayToManager(params.token1, params.amount1);

    // [3] mint position
    (int24 lower, int24 upper) = priceRange(pool, params.isStable);
    positionId = poolManager.mint(pool, lower, lower, upper, upper, uint128(params.amount0),
uint128(params.amount1), 0, 0);
    liquidity = poolManager.positions(positionId).liquidity;

    // [4] transfer positionId
    poolManager.transferFrom(address(this), msg.sender, positionId);

    // [4] sweep remain tokens
    (remain0, remain1) = sweepTokens(params);
}
```

[https://github.com/HAECHI-LABS/audit-Pangea_Bootstrap/blob/c615b693905984239d294bd39cee24c1a6df416e/contracts/bootstrap/PoolInitializer.sol#L30]

Issue

*Poolinitializer#initPool()*은 deployData에 의해서 풀을 생성합니다. 그러나, 풀을 생성하는 과정중에는 난수 값이 사용되지 않습니다. 즉, 같은 deployData로 두번이상 *Poolinitializer#initPool()*를 호출하게 되면, 예기치 않은 오류가 발생하게 됩니다.

Recommendation

DeployData를 통해서 이미 생성된 풀인지 검사해야합니다.

Comment

MasterDeployer에서 풀 팩토리를 화이트리스트 관리합니다. 화이트리스트가 아니면 풀 팩토리를 통해 풀을 생성할 수 없습니다.

풀 팩토리를 마이그레이션 시점에 whitelist처리하여 공격자가 저희보다 먼저 Pool Deploy하는 것을 방지합니다.

마이그레이션 와중에 공격자가 풀 팩토리에 들어갈 경우, 신규 풀 팩토리를 생성하여 해당 팩토리를 통해 마이그레이션 합니다.

▣ MINOR

Bootstrap 과정에서도 동일한 유동성을 제공하고, 더 적은 fee와 reward를 수취할 수 있음을 인지하셔야 합니다.

(2차 audit)

Issue

Concentrated Liquidity Pool의 특성상 동일한 유동성을 제공하고도 더 많은 fee와 reward를 얻을 수도, 얻지 못할 수도 있습니다. Bootstrap 과정에서 *LPStaker*는 Concentrated Liquidity Pool과 같은 방식으로, 얻을 수 있는 fee와 reward를 계산합니다.

따라서, Bootstrap 과정에서도 동일한 현상이 발생할 수 있음을 인지하여 주시길 바랍니다.

💡 TIPS

`addIncentives()` 함수로 인해 요구사항 밖의 시나리오가 발생할 수 있습니다.

(2차 audit)

```
function addIncentives(uint256 amount) external { // @audit why anyone?
    require(block.timestamp <= endTime);

    stone.safeTransferFrom(msg.sender, address(this), amount);

    totalIncentives += amount;
}
```

[https://github.com/HAECHI-LABS/audit-Pangea_Bootstrap/blob/c615b693905984239d294bd39cee24c1a6df416e/contracts/bootstrap/BootstrappingAuction.sol#L140]

Issue

`BootstrappingAuction#addIncentives()`를 이용하여, Phase 0, 1에 참여하지 않은 사용자가 거버넌스 토큰인 `stone`을 얻어, incentive를 얻는 요구사항 밖의 시나리오가 발생할 수 있습니다. 가령, Phanse 0 혹은 Phase 1에 참여한 사용자 A가 거버넌스 토큰 `stone`을 얻어, 이를 Phase 0 혹은 Phase 1에 참여하지 않은 사용자 B에 전송할 수 있습니다. 이를 받은 사용자 B는 `BootstrappingAuction#addIncentives()`를 이용하여, Incentive를 얻을 수 있습니다. 이는 요구사항 밖의 시나리오가 될 수 있으므로, 인지하여 주시길 바랍니다.

Comment

phase0 혹은 phase1에서 STONE을 수령하기 위해서는 launch 이후에 수령이 가능한데, launch 시점은 위의 `block.timestamp <= endTime` 조건으로 걸리기 때문에 얻은 토큰으로 `addIncentive`하는 케이스는 존재하지 않습니다. 그리고 시간 조건은 세팅 시에 조건을 두고 있기 때문에 행여 시간이 역전되는 케이스는 발생하지 않도록 예방하였습니다.

💡 TIPS

Airdrop에 참여한 recipient에게 stone을 전송해야 합니다.

(2차 audit)

```
function sweepUnclaimedRewards(address recipient) external onlyOwner {  
    require(block.timestamp >= claimEndTime, "NOT ENDED");  
    uint256 unclaimedRewards = stone.balanceOf(address(this));  
    stone.safeTransfer(recipient, unclaimedRewards);  
}
```

[https://github.com/HAECHI-LABS/audit-Pangea_Bootstrap/blob/c615b693905984239d294bd39cee24c1a6df416e/contracts/bootstrap/Airdrop.sol#L119]

Issue

요구사항에 따르면, Airdrop 단계에서 거버넌스 토큰인 stone을 분배하는 것은 Airdrop에 참가한 사용자에게 분배되어야 합니다. 하지만, *Airdrop#sweepUnclaimedRewards()*에서 stone을 분배받는 recipient가 Airdrop에 참가한 recipient가 아닐수 있습니다.

Comment

owner만이 회수할 수 있도록 제약 조건을 두고, 이후에 airdrop 이벤트가 종료되고 나서 받아가지 않은 물량에 대해서는 거버넌스의 의사결정에 따라 처리하기 위함입니다
airdrop이벤트 기간내에 받아가지 않으면 회수한다는 것을 Docs에 명시하고 있습니다.

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

End of Document