HAECHI AUDIT

Pangea - Stone Staking

Smart Contract Security Analysis

Published on : 5 Dec. 2022

Version v1.1

HAECHI AUDIT

Smart Contract Audit Certificate

Pangea - Stone Staking

Security Report Published by HAECHI AUDIT

v1.1 Dec. 2022

Auditor : Andy Koo, Jade Han

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	-	-	-	-
High	2	2	-	-
Medium	-	-	-	-
Low	-	-	-	-
Tips	2	1	1	-

TABLE OF CONTENTS

ABOUT US

The most reliable web3 security partner.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. We bring together the best Web2 and Web3 experts. Security Researchers with expertise in cryptography, leaders of the global best hacker team, and blockchain/smart contract experts are responsible for securing your Web3 service.

We have secured the most well-known web3 services including 1inch, SushiSwap, Klaytn, Badger DAO, SuperRare, Netmarble, Klaytn and Chainsafe. We have secured $60B crypto assets on over 400 main-nets, Defi protocols, NFT services, P2E, and Bridges.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io

Website: audit.haechi.io

Executive Summary

Purpose of this report

This report was prepared to audit the security of the Stone Staking contracts developed by the Pangea team. HAECHI AUDIT conducted the audit focusing on whether the system created by the Pangea team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the Stone Staking contract.
In detail, we have focused on the following

- Proper calculation of reward and dividend on epoch base.
- Arithmetic over/underflow issue on reward/dividend calculation.
- Existence of Denial of Service on fee collection and distribution.
- Storage variable access control.
- Existence of known smart contract vulnerabilities

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub (https://github.com/pangea-protocol/stone-staking/tree/c601bc412a390924086686bf6f2b95b08f5aa75c).

The last commit of the code used for this Audit is "{c601bc412a390924086686bf6f2b95b08f5aa75c}".

Audit Timeline

Date	Event
2022/11/15	Audit Initiation (Stone Staking)
2022/11/29	Delivery of v1.0 report.
2022/12/05	Delivery of v1.1 report.

Findings

HAECHI AUDIT found 2 High. There are 2 Tips issues explained that would improve the code's usability or efficiency upon modification. All the issues found have been confirmed to be fixed or acknowledged.

Severity	Issue	Status
High	Dividend miscalculation when user share is updated after dividend record.	(Resolved - v1.1)
High	CollectByPage, collectFrom DOS	(Resolved - v1.1)
Tips	Check-Effect-Interaction pattern is not followed	(Resolved - v1.1)
Tips	The pendingReward sweep issue.	(Acknowledged - v1.1)

Fix

#ID	Title	Type	Severity	Difficulty	Status
1	Dividend miscalculation when user share is updated after dividend record.	Logic Error	**High**	**High**	**Resolved**
2	collectByPage, collectFrom DOS	Denial of Service	**High**	**Low**	**Resolved**
3	Check-Effect-Interaction pattern is not followed	Reentrancy	**Tips**	**Info**	**Resolved**
4	The pendingReward sweep issue.	Logic Error	**Tips**	**Info**	**Acknowledged**

Remarks

The ProtocolRevenueShare.sol contract interacts with Pangea's swap pool to collect swap fees. For the completeness of the audit we took account of the interaction with the swap pool which is not a scope of this audit.

OVERVIEW

Protocol overview

- **ProtocolRevenueShare.sol**

The contract collects swap fees that occurred from the Pangea's each swap pool and distributes collected revenues to the governance accounts. The collected fee tokens are swapped to the revenue token which is set by the MANAGER_ROLE account. The swap is executed on a whitelisted broker contract. The collection and transfer of revenue are conducted by OP_ROLE account.

- **StakedStone.sol**

Users can stake Stone tokens and receive rewards and dividends pro rata stake amount and periods. The reward is calculated on a week(7 days) basis, and the dividend is calculated by the staked amount and period between the previous epoch's record date to the current epoch's record date. The user's share and snapshot are updated by the updateUserSnapshot modifier when the user stakes or unstakes the token.

The MANAGER_ROLE account can deposit the reward and dividend for the staked users, and also can cancel the reward distribution or reset the record date of the dividend.

When the user unstakes the Stone token, an unstaking request is queued on the storage variable, 7 days after, the user can withdraw the requested token.

ProtocolRevenueShare.sol and StakedStone.sol contracts both inherit the Open Zeppelin's multicall contract that enables multiple function calls on a single transaction.

Scope

contracts

```
├── ProtocolRevenueShare.sol
├── StakedStone.sol
├── interfaces
│   ├── IMasterDeployer.sol
│   ├── IProtocolFeePool.sol
│   ├── IProtocolRevenueShare.sol
│   ├── IStakedStone.sol
│   └── IWETH.sol
├── libraries
    ├── FixedPoint.sol
    └── FullMath.sol
```

Access Controls

Stone Staking contracts have the following access control mechanisms.

- onlyRole(MANAGER_ROLE)
- onlyRole(OP_ROLE)

onlyRole(MANAGER_ROLE) : modifier that controls access to variables which relate to rewards/dividend distribution, and fee collection.

- ProtocolRevenueShare#setRevenueToken()
- ProtocolRevenueShare#setGrowthFund()
- ProtocolRevenueShare#setDaoFund()
- ProtocolRevenueShare#setMinimumRevenue()
- ProtocolRevenueShare#setGrowthFundRate()
- ProtocolRevenueShare#setFactoryGrowthFundRate()
- ProtocolRevenueShare#verifyBroker()
- ProtocolRevenueShare#setApproval()
- StakedStone#depositReward()
- StakedStone#cancelReward()
- StakedStone#setDividendRecordDate()
- StakedStone#resetDividendRecordDate()
- StakedStone#depositDividend()
- StakedStone#executeDividend()
- StakedStone#setCooldownPeriod()

onlyRole(MANAGER_ROLE) : modifier that controls access to fee collection and transfer feature.

- ProtocolRevenueShare#collectByPage()
- ProtocolRevenueShare#collectFrom()
- ProtocolRevenueShare#share()

The MANAGER_ROLE has permissions that can change the crucial part of the system. It is highly recommended to maintain the private key as securely as possible and strictly monitor the system state changes.

FINDINGS

1. Dividend miscalculation when user share is updated after dividend record.

ID: Pangea-01
Severity: High
Type: Logic Error
Difficulty: High
File: contracts/StakedStone.sol

Issue

The user's share is updated when the user stakes or unstakes the stone token. When the user's share is updated between the dividend record time and execution time, the user's share is updated to the wrong dividend epoch which leads to a miscalculation of the dividend.

```
function _updateUserShare(address owner) internal {
    uint256 balance = _balanceOf[owner];

    // @dev skip to update user share
    if (balance == 0) return;

    uint256 lastRecordDate = _userLastRecordDate[owner];

    // @dev there is no previous dividend
    if (_dividendHistory.length == 0) {
        _userDividendSnapshot[owner][0].share += (block.timestamp - lastRecordDate) * balance;
        return;
    }

    uint256 index = _dividendHistory.length - 1;
    Dividend memory dividend = _dividendHistory[index];

    uint256 period = block.timestamp - Math.max(dividend.recordDate, lastRecordDate);
    _userDividendSnapshot[owner][index+1].share += period * balance;

    while (dividend.recordDate > lastRecordDate) {
        period = dividend.recordDate - Math.max(dividend.startDate, lastRecordDate);
        _userDividendSnapshot[owner][index].share += period * balance;

        if (index == 0) break;

        dividend = _dividendHistory[--index];
    }
}
```

[https://github.com/pangea-protocol/stone-staking/blob/c601bc412a390924086686bf6f2b95b08f5aa75c/contracts/StakedStone.sol#L579-L607]

The dividend is allocated when the admin calls the following functions in order, setDividendRecordDate() -> depositDividend() -> executeDividend(). The dividend epoch is the period between the record date of the previous epoch and the record date of the current epoch. When the user's share is updated between setDividendRecordDate() and executeDividend(), the updated share is added to the current epoch's user share which should be added to the next epoch's user share.

Recommendation

- Combining the dividend allocation functions [setDividendRecordDate() -> depositDividend() -> executeDividend()] can prevent unexpected user's share update.
- Or, adding a branch statement which correctly includes the user's share on the period is desired.

Update

The issue has been resolved by restricting users' share update when the dividend process is ongoing. The updateUserSnapshot modifier checks the readyDividend.recordDate to check if the share update is available. [Commit Link]

2. CollectByPage, collectFrom DOS

ID: Pangea-02	Severity: High
Type: Denial of Service	Difficulty: Low
File: contracts/ProtocolRevenueShare.sol	

Issue

A malicious token developer can block the fee that should be distributed to Pangea users and use it as liquidity in the swap pool.

```
function collectByPage(uint256 start, uint256 limit) external onlyRole(OP_ROLE) {
    IMasterDeployer deployer = IMasterDeployer(masterDeployer);

    uint256 end = Math.min(deployer.totalPoolsCount(), start + limit);
    if (start >= end) return;

    for (uint256 i = start; i < end; i++) {
        address pool = deployer.getPoolAddress(i);
        (uint128 rev0, uint128 rev1) = IProtocolFeePool(pool).getTokenProtocolFees();

        // @dev 프로토콜 수익이 존재하지 않은 경우 스킵
        if (rev0 == 0 && rev1 == 0) continue;

        cachedPool = pool;
        IProtocolFeePool(pool).collectProtocolFee();
    }

    cachedPool = address(0);
}
```

[https://github.com/pangea-protocol/stone-staking/blob/c601bc412a390924086686bf6f2b95b08f5aa75c/contracts/ProtocolRevenueShare.sol#L237-L255]

In Pangea Swap, anyone can form a pair and create a dex pool. An admin can send a fee to a specific contract by calling the fee collection function within the pair contract. (Here, the fee is accumulated by tokens forming a pair each time you swap). When a fee is collected, it is distributed to Pangea users.

However, if the token developer executes the revert code when the to parameter of the transfer function is a contract that receives a fee, the pool does not transfer the fee to that specific contract and continues to use it as liquidity in the pool, or LPs can receive tokens including fee when the LP Token is burned.

Recommendation

When calculating the swap output in the swap pool contract, reflect the accumulated fee from the swap amount calculation to remove the motivation to perform the vulnerability mentioned in the report.

Update

The DoS issue is handled by skipping a specific swap pool. The admin can set the skipCollect list to prevent collecting fees from malicious swap pools. [Commit Link]

3. Check-Effect-Interaction pattern is not followed

ID: Pangea-03

Type: Reentrancy

File: contracts/StakedStone.sol

Severity: Tips

Difficulty: Informational

Issue

StakedStone.sol#withdraw() and StakedStone.sol#claimDividend() functions have outbound contract call, however, the check-effect-interaction(CEI) pattern is not followed.

```
function withdraw(uint256 requestId) external returns (uint256 amount) {
    UnstakingRequest memory request = unstakingRequests[requestId];
    require(!request.isClaimed, "ALREADY CLAIMED");
    require(requestOwnerOf[requestId] == msg.sender, "NOT OWNER");
    require(request.requestTs + cooldownPeriod <= block.timestamp, "NEED COOLDOWN");
    amount = request.amount;

    IERC20(stone).safeTransfer(msg.sender, amount);

    closeRequest(msg.sender, requestId);

    emit Withdraw(msg.sender, amount);
}
```

[https://github.com/pangea-protocol/stone-staking/blob/c601bc412a390924086686bf6f2b95b08f5aa75c/contracts/StakedStone.sol#L399-L411]

```
function claimDividend(uint256 epoch) external updateUserSnapshot(msg.sender) {
    DividendSnapshot memory snapshot = _userDividendSnapshot[msg.sender][epoch];
    require(!snapshot.isPaid, "ALREADY PAID");
    require(snapshot.share > 0, "NO SHARE");

    Dividend memory epochDividend = _dividendHistory[epoch];
    uint256 epochTotalShare = epochDividend.totalShare;

    for (uint256 i = 0; i < epochDividend.tokens.length; i++) {
        address token = epochDividend.tokens[i];
        uint256 amount = epochDividend.amounts[i];

        uint256 userAmount = FullMath.mulDiv(
            amount, snapshot.share, epochTotalShare
        );

        IERC20(token).safeTransfer(msg.sender, userAmount);

        emit ClaimDividend(msg.sender, epoch, token, userAmount);
    }

    _userDividendSnapshot[msg.sender][epoch].isPaid = true;
}
```

[https://github.com/pangea-protocol/stone-staking/blob/c601bc412a390924086686bf6f2b95b08f5aa75c/contracts/StakedStone.sol#L470-L492]

The CEI pattern prevents the contract from being victimized by a reentrancy attack. The StakedStone.sol#withdraw() function calls to the stone token and StakedStone.sol#claimDividend() calls to the reward token. The storage variable update is implemented after the outbound call is executed. In case the malicious user hooks the call flow and makes a call to the function again, drainage of all stones or reward tokens may occur.

Currently, the outbound calls are only made to the stone token and USDT(reward token). These tokens have no callback function that can make use of a reentrancy attack. However, following the CEI pattern prevents possible reentrancy attacks in the future.

Recommendation

Short term, follow the CEI pattern. That means closeRequest() function should be implemented before the safeTransfer call in the StakedStone.sol#withdraw() function and _userDividendSnapshot should be updated before safeTransfer call on the reward token on the StakedStone.sol#claimDividend() function.

Long term, be assured that the interacting tokens have no callback function which can be used for reentrancy attack.

Update

Check-Effect-Interaction pattern is implemented to the withdraw and claimDividend function. [Commit Link]

4. The pendingReward sweep issue.

ID: Pangea-04	Severity: Tips
Type: Denial of Service	Difficulty: Informational
File: contracts/ProtocolRevenueShare.sol	

Issue

The first person that stake token can monopolize pendingReward. The pendingReward is not distributed per user, but can be sweeped by a single user.

```
modifier updateUserSnapshot(address owner) {
        uint256 growthGlobal = _updateGrowthGlobal();
        _updateRewardSnapshot(owner, growthGlobal);
        _updateDividendSnapshot(owner);

        _;
    }

function _updateGrowthGlobal() internal returns (uint256 growthGlobal) {
        uint256 _checkpoint = checkpoint;
        uint256 amount = _calculateRewardToDistribute();

        amount = _updatePendingReward(amount);
        growthGlobal = _rewardGrowthGlobal(amount);
        rewardGrowthGlobalLast = growthGlobal;

        // @dev Skip if the block has been updated in advance. (gas efficient policy)
        if (_checkpoint >= block.timestamp) {
            return growthGlobal;
        }

        totalShare += (block.timestamp - _checkpoint) * _totalSupply;
        checkpoint = block.timestamp;
    }

function _updatePendingReward(uint256 amount) internal returns (uint256) {
        if (_totalSupply == 0) {
            // @dev Rewards accumulated while there is no staked supply
            // are distributed later
            pendingReward += amount;
            return 0;
        }

        if (pendingReward > 0) {
            // @dev add pendingReward if it remains
            amount += pendingReward;
            pendingReward = 0;
        }

        return amount;
    }
```

[https://github.com/pangea-protocol/stone-staking/blob/c601bc412a390924086686bf6f2b95b08f5aa75c/contracts/StakedStone.sol#L399-L411]

Recommendation

Estimate the risk of the sweep of pending rewards at the discretion of the project.

Comment by Pangea Team

As the occurrence of a pending reward is an unusual situation, taking the pending reward by the single person who stakes first is expected behavior.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix. A

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	**Medium**	**High**
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	▪ Integer under/overflow vulnerability ▪ floating point and rounding accuracy
Access & Privilege Control	▪ Manager functions for emergency handle ▪ Crucial function and data access ▪ Count of calling important task, contract state change, intentional task delay
Denial of Service	▪ Unexpected revert handling ▪ Gas limit excess due to unpredictable implementation
Miner Manipulation	▪ Dependency on the block number or timestamp. ▪ Frontrunning
Reentrancy	▪Proper use of Check-Effect-Interact pattern. ▪Prevention of state change after external call ▪ Error handling and logging.
Low-level Call	▪ Code injection using delegatecall ▪ Inappropriate use of assembly code
Off-standard	▪ Deviate from standards that can be an obstacle of interoperability.
Input Validation	▪ Lack of validation on inputs.
Logic Error/Bug	▪ Unintended execution leads to error.
Documentation	▪Coherency between the documented spec and implementation
Visibility	▪ Variable and function visibility setting
Incorrect Interface	▪ Contract interface is properly implemented on code.

Appendix. B

POC of PANGEA-01

```
    function test_dividend_calculation_test_multiple_dividend_in_epoch0() public {
        stoneFaucet(user1, 10e18);
        stoneStake(user1, 10e18);

        vm.warp(block.timestamp + 1 days);
        vm.prank(admin);
        stakedStone.setDividendRecordDate();

        vm.warp(block.timestamp + 2 days);
        stoneFaucet(user2, 10e18);
        stoneStake(user2, 10e18);

        stoneFaucet(user1, 10e18);
        stoneStake(user1, 10e18);

        kusdtFaucet(admin, 70e18);
        vm.startPrank(admin);
        kusdt.approve(address(stakedStone), 70e18);
        stakedStone.depositDividend(address(kusdt), 70e18);
        vm.stopPrank();

        vm.warp(block.timestamp + 1 days);
        stoneFaucet(user3, 10e18);
        stoneStake(user3, 10e18);

        vm.warp(block.timestamp + 1 days);
        vm.prank(admin);
        stakedStone.executeDividend();

        vm.prank(user1);
        stakedStone.claimDividend(0);

        console.log(kusdt.balanceOf(address(user1)));
    }
```

```
...
    ├─ [35578] stakedStone::claimDividend(0)
    │   ├─ [1008] usdt::transfer(user1: [0x7E5F4552091A69125d5DfCb7b8C2659029395Bdf],
210000000000000000000)
    │   │   └─ ← "EvmError: Revert"
    │   └─ ← "SafeERC20: low-level call failed"
    └─ ← "SafeERC20: low-level call failed"

Test result: FAILED. 0 passed; 1 failed; finished in 3.35ms
```

POC of PANGEA-02

```
pragma solidity ^0.8.13;

import "forge-std/Script.sol";
import "forge-std/console.sol";

import "../src/test/MaliciousToken1.sol";
import "../src/ProtocolRevenueShare.sol";
import "../src/DepositRelayer.sol";

import "forge-std/StdStorage.sol";

interface IMyMasterDeployer {
    function deployPool(address factory, bytes calldata data) external returns (address);
    function setProtocolFeeTo(address _protocolFeeTo) external;
}

interface IPool {
    function getReserves() external view returns (uint128 _reserve0, uint128 _reserve1);
    function getTokenProtocolFees() external view returns (uint128 _token0ProtocolFee, uint128
_token1ProtocolFee);
}

contract ContractScript is Script {
    using stdStorage for StdStorage;

    MaliciousToken1 public MAL1;
    ProtocolRevenueShare public ShareContract;

    address public masterDeployer = 0xEB4B1CE03bb947Ce23ABd1403dF7C9B86004178d;
    address public factory = 0x3d94b5E3b83CbD52B9616930D33515613ADfAd67;

    address public USDT = 0xceE8FAF64bB97a73bb51E115Aa89C17FfA8dD167;
    address public wKlay = 0xFF3e7cf0C007f919807b32b30a4a9E7Bd7Bc4121;

    address public poolAddr;
    function setUp() public {
        console.log("%d", 7 days);
        console.log("setUp called...");
        MAL1 = new MaliciousToken1("MaliciousToken1", "MAL1");
        ShareContract = new ProtocolRevenueShare();

        console.log("MAL1 Token Address : %s", address(MAL1));

        bytes memory deployData =
hex"000000000000000000000000c7f2cf4845c6db0e1a1e91ed41bcd0fcc1b0e141000000000000000000000000ff3e7cf0
c007f919807b32b30a4a9e7bd7bc4121000000000000000000000000000000000000000000000000000000000000007d000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000014";
        (address tokenA, address tokenB, uint24 swapFee, uint160 price, uint24 tickSpacing) =
abi.decode(
            deployData,
            (address, address, uint24, uint160, uint24)
        );

        console.log("tokenA : %s", tokenA);
        console.log("tokenB : %s", tokenB);
        console.log("swapFee : %d", swapFee);
        console.log("price : %d", price);
        console.log("tickSpacing : %d", tickSpacing);
        if (tokenA > tokenB) console.log("error point");
```

```
        uint size;
        address a = masterDeployer;
        assembly {
            size := extcodesize(a)
        }
        console.log("masterDeployer contract size : %d", size);
        poolAddr = IMyMasterDeployer(masterDeployer).deployPool(factory, deployData);

        console.log("customPool address : %s", poolAddr);
        console.log("ShareContract address : %s", address(ShareContract));
        MAL1.setBlacklist(address(ShareContract));

        ShareContract.initialize(masterDeployer, USDT, wKlay);
        ShareContract.grantRole(keccak256(abi.encode("OP")), address(this));
        ShareContract.grantRole(keccak256(abi.encode("MANAGER")), address(this));

        ShareContract.setGrowthFund(address(this));
        ShareContract.setDaoFund(address(this));


        vm.startPrank(0x2A2F23ff33671361010D357529BDF0adca9416Fc);
        IMyMasterDeployer(masterDeployer).setProtocolFeeTo(address(ShareContract));
        vm.stopPrank();

    }

    function run() public {
        (uint256 reserve0, uint256 reserve1) = IPool(poolAddr).getReserves();
        (uint256 fee0, uint256 fee1) = IPool(poolAddr).getTokenProtocolFees();

        console.log("pool reserve0 : %d, reserve1 : %d", reserve0, reserve1);
        console.log("pool fee0 : %d, fee1 : %d", fee0, fee1);


        vm.store(poolAddr, bytes32(uint256(11)),
bytes32(uint256(340282366920938463463374607431768211457))); // fee0-1, fee1-1
        vm.store(poolAddr, bytes32(uint256(12)),
bytes32(uint256(340282366920938463463374607431768211457))); // reserve0-1, reserve1-1
        (reserve0, reserve1) = IPool(poolAddr).getReserves();
        (fee0, fee1) = IPool(poolAddr).getTokenProtocolFees();

        console.log("storage change~~~");
        console.log("pool reserve0 : %d, reserve1 : %d", reserve0, reserve1);
        console.log("pool fee0 : %d, fee1 : %d", fee0, fee1);
        console.log("--------------");

        DepositRelayer relayerOriginal = new
DepositRelayer{salt:bytes32(uint256(uint160(32333141241)) << 96)}();
        console.log("relayerOriginal : %s", address(relayerOriginal));
        //ShareContract.collectFrom(poolAddr);
        //revert();
    }
}
```

End of Document