Competitive Security Assessment

DegenReborn

Mar 10th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	DegenReborn
Platform & Language	Solidity
Codebase	• https://github.com/NirvanaLabHQ/contracts • audit commit - d96eac7783059c434c9562c7e0e9b9b540e084e0 • final commit - 54c091d20ba7338a60750901692aa6c2af1a7471
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

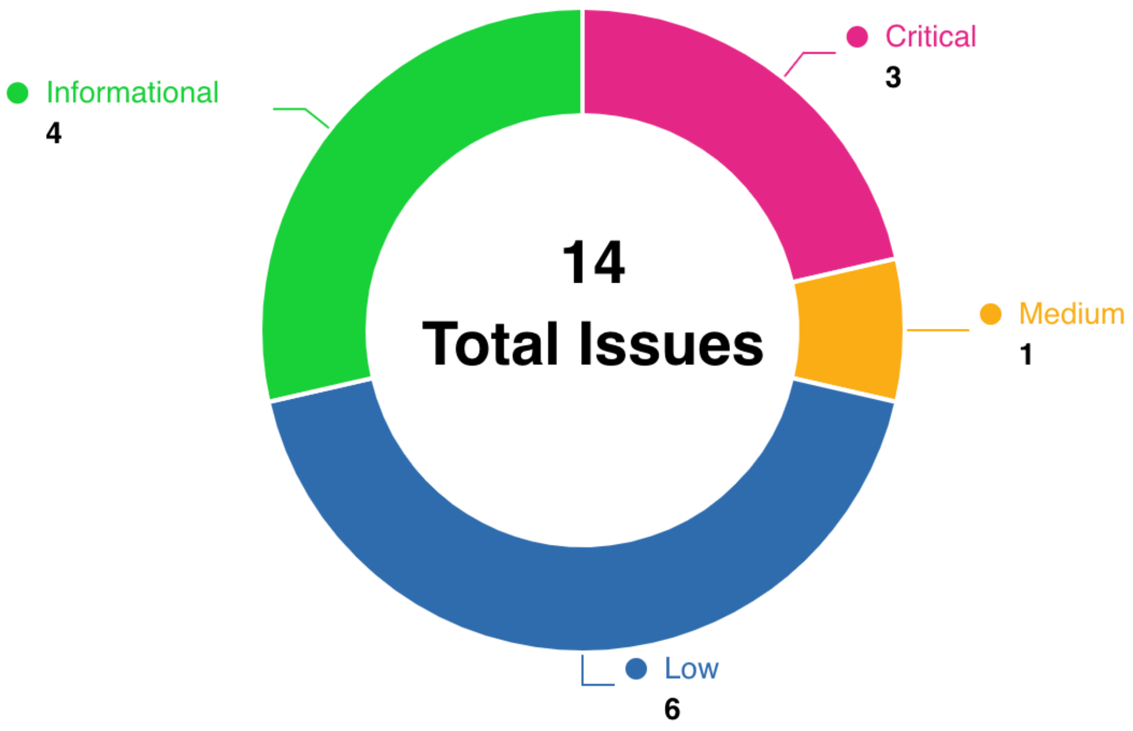
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	**3**	**0**	**0**	**3**	**0**	**0**
Medium	**1**	**0**	**1**	**0**	**0**	**0**
Low	**6**	**0**	**1**	**4**	**0**	**1**
Informational	**4**	**0**	**2**	**2**	**0**	**0**

Audit Scope

File	Commit Hash
src/RewardDistributor.sol	d96eac7783059c434c9562c7e0e9b9b540e084e0
src/RewardVault.sol	d96eac7783059c434c9562c7e0e9b9b540e084e0
src/RBT.sol	d96eac7783059c434c9562c7e0e9b9b540e084e0
src/RankUpgradeable.sol	d96eac7783059c434c9562c7e0e9b9b540e084e0
src/RBTStorage.sol	d96eac7783059c434c9562c7e0e9b9b540e084e0
src/RebornPortalStorage.sol	d96eac7783059c434c9562c7e0e9b9b540e084e0
src/RebornPortal.sol	d96eac7783059c434c9562c7e0e9b9b540e084e0
src/interfaces/IRewardVault.sol	d96eac7783059c434c9562c7e0e9b9b540e084e0
src/interfaces/IRebornToken.sol	d96eac7783059c434c9562c7e0e9b9b540e084e0
src/interfaces/IRewardDistributor.sol	d96eac7783059c434c9562c7e0e9b9b540e084e0
src/interfaces/IRebornPortal.sol	d96eac7783059c434c9562c7e0e9b9b540e084e0

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
RBN-1	Centralization risk in `rewardDistributor` contract `withdraw` function	Privilege Related	Low	Acknowledged	Kong7ych3, helookslikeme
RBN-2	DoS risk of divided by 0 due to 0 `totalAmount` value	DoS	Critical	Fixed	Kong7ych3
RBN-3	Duplicate rank key issue	Logical	Critical	Fixed	Kong7ych3
RBN-4	Missing event record	Code Style	Informational	Fixed	Kong7ych3
RBN-5	No check for existing state	Code Style	Informational	Acknowledged	Kong7ych3

RBN-6	Redundant reward distribution issue	Gas Optimization	Informational	Acknowledged	Kong7ych3
RBN-7	Reentrancy risk in the `engrave` function of the `RebornPortal` contract	Reentrancy	Low	Declined	helookslikeme
RBN-8	Risk of incorrectly setting rebornToken	Code Style	Low	Fixed	Kong7ych3
RBN-9	The performUpkeep operation can be called ahead of time	Logical	Low	Fixed	Kong7ych3
RBN-10	Users are able to avoid paying burn fees	Logical	Medium	Acknowledged	0xac
RBN-11	`RebornPortal` and `RBT` cache array length outside loop	Gas Optimization	Low	Fixed	helookslikeme
RBN-12	`RewardVault.rebornToken` can be defined as `immutable`	Gas Optimization	Informational	Fixed	0xac
RBN-13	`performUpkeep` cannot airdrop native token	Logical	Critical	Fixed	0xac, Kong7ych3
RBN-14	`pool.accNativePerShare` divide-before-multiply loss of precision	Language Specific	Low	Fixed	0xac

RBN-1:Centralization risk in `rewardDistributor` contract `withdraw` function

Category	Severity	Code Reference	Status	Contributor
Privilege Related	Low	• code/src/RewardDistributor.sol#L94-L98	Acknowledged	Kong7ych3, helookslikeme

Code

```
94:    function withdraw() external onlyOwner {
95:        uint256 balance = (address(this)).balance;
96:        payable(msg.sender).transfer(balance);
97:        emit WithDrawn(msg.sender, balance);
98:    }
```

Description

Kong7ych3 : In RewardDistributor contracts, users can get rewards within claimPeriodEnds via the claimTokens function. However, the owner can withdraw the pending funds to be distributed in the contract at any time via the withdraw function. This will lead to an excessive risk of owner privileges, and if the owner withdraws funds before claimPeriodEnds, the user will not be able to receive the rewards properly.
helookslikeme : Administrators can directly withdraw all funds in the contract, which may cause losses to users, and there is a risk of centralization

Recommendation

Kong7ych3 : It is recommended to check the current time greater than ClaimPeriodends in the Withdraw function to avoid the owner from withdrawing funds in advance.
Consider below fix in the `RewardDistributor::withdraw()` function

```
function withdraw() external onlyOwner {
    require(block.timestamp >= claimPeriodEnds, "The claim period is not over yet");
    uint256 balance = (address(this)).balance;
    payable(msg.sender).transfer(balance);
    emit WithDrawn(msg.sender, balance);
}
```

helookslikeme : Using multi-signature functions and other solutions

Client Response

It's necessary to withdraw native token manually and distribute reward manually in the early stage. We will adopt a multi-sig solution.

RBN-2:DoS risk of divided by 0 due to 0 `totalAmount` value

Category	Severity	Code Reference	Status	Contributor
DoS	**Critical**	• **code/src/RebornPortal.sol#L389-L390** • **code/src/RebornPortal.sol#L414-L418**	**Fixed**	**Kong7ych3**

Code

```
389:                    (_dropConf._rebornDropEthAmount * 1 ether * PERSHARE_BASE) /
390:                    pool.totalAmount;

414:                pool.accNativePerShare +=
415:                    (((_dropConf._nativeDropRatio * address(this).balance * 3) / 200) *
416:                        PERSHARE_BASE) /
417:                    PERCENTAGE_BASE /
418:                    pool.totalAmount;
```

Description

Kong7ych3 : In the RebornPortal contract, `_dropReborn` and `_dropNative` functions are used to update accPerShare to the top 100 tvl pool, which will divide `pool.totalAmount` when calculating accPerShare. totalAmount is the amount of $REBORN tokens staked by the user in this pool, but users can stake $REBORN tokens to other pools through the `switchPool` function, so `pool.totalAmount` may become 0.

When `pool.totalAmount` is 0, it will be divided by 0 when calculating accPerShare, which will cause the entire performUpkeep operation to be reverted, resulting in the failure of all pool rewards to be updated normally, resulting in reward distribution DoS.

Recommendation

Kong7ych3 : It is recommended that in `_dropReborn` and `_dropNative` functions, when `pool.totalAmount is` 0, continue operation is performed to avoid performUpkeep operation revert.

Client Response

Fixed

RBN-3:Duplicate rank key issue

Category	Severity	Code Reference	Status	Contributor
Logical	**Critical**	• **code/src/RankUpgradeable.sol#L20-L22**	**Fixed**	**Kong7ych3**

Code

```
20:         if (value == 0) {
21:             _exit(tokenId);
22:         }
```

Description

Kong7ych3 : In the RankUpgradeable contract, the `_enter` function is used to update the rank tree. When the value passed in is 0 (the `pool.totalAmount` passed in is 0), it will clear the corresponding rank through the `_exit` function. But when the execution of `_exit` is completed, it does not end the call of `_enter`, but continues to execute the `_rank.add` operation, which will cause the rank removed in `_exit` to be added back. And when the `pool.totalAmount` of the same tokenId is greater than 0 again, this tokenId will be added again when the rank is updated through `_enter`, that is, the same tokenId will exist in the rank tree at the same time.

Here is a simple exploit scenario: We assume that there are only three pools at this time, and the tokenIds are 1, 2 and 3 respectively. At this point `_getTopNTokenId(100)` will return [3,2,1,0,0...0,0] At this time, the `pool.totalAmount` of 1 tokenId is reduced to 0, and after the `_enter(1, 0)` operation, the return value of `_getTopNTokenId(100)` will be [2,3,1,0,... ,0,0] When a user stakes 100 $REBORN on 1 tokenId again, the `_enter(1, 100)` operation will be executed, and finally `_getTopNTokenId(100)` will return [1,2,3,1,0,..., 0,0] We can find that tokenId 1 appears twice in the return value of the `_getTopNTokenId` operation, which will have a serious impact on the protocol when the pool is less than 100. The same tokenId will be rewarded multiple times, resulting in The reward for this tokenId is much larger than expected.

Recommendation

Kong7ych3 : It is recommended to return immediately after the execution of `_exit` in the `_enter` function.

Consider below fix in the `RankUpgradeable::_enter()` function

```
function _enter(uint256 tokenId, uint256 value) external {
    if (value == 0) {
        _exit(tokenId);
+       return;
    }

    // remove old value from the rank, keep one token Id only one value
    if (_tokenIdOldValue[tokenId] != 0) {
        _rank.remove(tokenId, _tokenIdOldValue[tokenId]);
    }
    _rank.add(tokenId, value);
    _tokenIdOldValue[tokenId] = value;
}
```

Client Response

Fixed

RBN-4:Missing event record

Category	Severity	Code Reference	Status	Contributor
Code Style	**Informational**	• **code/src/RebornPortal.sol#L199**	**Fixed**	**Kong7ych3**

Code

```
199:        vault = vault_;
```

Description

Kong7ych3 : In the RebornPortal contract, the owner can set the vault address through the setVault function, but the event is not recorded.

Recommendation

Kong7ych3 : It is recommended to record events when modifying sensitive parameters for community review and self-examination.

Client Response

Fixed

RBN-5:No check for existing state

Category	Severity	Code Reference	Status	Contributor
Code Style	**Informational**	• **code/src/RBT.sol#L49-L61** • **code/src/RebornPortal.sol#L215-L227**	**Acknowledged**	**Kong7ych3**

Code

```
49:    function updateMinter(
50:        address[] calldata toAdd,
51:        address[] calldata toRemove
52:    ) external onlyOwner {
53:        for (uint256 i = 0; i < toAdd.length; i++) {
54:            minters[toAdd[i]] = true;
55:            emit MinterUpdate(toAdd[i], true);
56:        }
57:        for (uint256 i = 0; i < toRemove.length; i++) {
58:            delete minters[toRemove[i]];
59:            emit MinterUpdate(toRemove[i], false);
60:        }
61:    }

215:    function updateSigners(
216:        address[] calldata toAdd,
217:        address[] calldata toRemove
218:    ) external onlyOwner {
219:        for (uint256 i = 0; i < toAdd.length; i++) {
220:            signers[toAdd[i]] = true;
221:            emit SignerUpdate(toAdd[i], true);
222:        }
223:        for (uint256 i = 0; i < toRemove.length; i++) {
224:            delete signers[toRemove[i]];
225:            emit SignerUpdate(toRemove[i], false);
226:        }
227:    }
```

Description

Kong7ych3 : In the RBT contract, the owner can update the status of the minter role in batches through the updateMinter function. In the RebornPortal contract, the owner can update the status of the signe role in batches through the updateSigners function, but it does not check whether the incoming toAdd and toRemove lists have the same address , and does not check whether the status of toAdd address is false and whether the status of toRemove address is true.

Recommendation

Kong7ych3 : It is recommended to check that the addresses in the toAdd and toRemove lists are not the same, or that care should be taken to ensure that the addresses in the two lists are different when constructing off-chain. And check that the status of toAdd address is false and the status of toRemove address is true before setting the status.

Client Response

It's a function with low frequency. We will check it carefully before calling it.

RBN-6:Redundant reward distribution issue

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	**Informational**	• **code/src/RebornPortal.sol#L474-L478** • **code/src/RebornPortal.sol#L503-L507**	**Acknowledged**	**Kong7ych3**

Code

```
474:            address ref1,
475:            uint256 ref1Reward,
476:            address ref2,
477:            uint256 ref2Reward
478:        ) = _calculateReferReward(account, amount, RewardType.RebornToken);

503:            address ref1,
504:            uint256 ref1Reward,
505:            address ref2,
506:            uint256 ref2Reward
507:        ) = _calculateReferReward(account, amount, RewardType.NativeToken);
```

Description

Kong7ych3 : In the RebornPortal contract, the `_vaultRewardToRefs` and `_sendRewardToRefs` functions are used to distribute rewards to the referrer, but they have not checked whether the referrer address returned by `_calculateReferReward` is 0 address, if it is 0 address, continuing to execute the reward distribution logic will be in vain consumes unnecessary gas.

Recommendation

Kong7ych3 : It is recommended to check whether `ref1` and `ref2` returned by `_calculateReferReward` are 0 addresses. If the referrer's address is 0, the reward distribution logic will no longer be executed.

Client Response

Checking address is included in `_calculateReferReward`

RBN-7:Reentrancy risk in the `engrave` function of the `RebornPortal` contract

Category	Severity	Code Reference	Status	Contributor
Reentrancy	**Low**	• **code/src/RebornPortal.sol#L88**	**Declined**	**helookslikeme**

Code

```
88:        _safeMint(user, tokenId);
```

Description

helookslikeme : The `_safeMint()` function will call `_safeMint()` in the ERC721 contract, thereby calling the `_checkOnERC721Received()` function of the transferred address. If the `onERC721Received()` of the forwarded address contains malicious code, the attack can be carried out.

Given the function can only be called by `onlySigner`, the severity is low

Recommendation

helookslikeme : consider use reentrant lock `nonReentrant` modifier - https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/ReentrancyGuard.sol

Client Response

Declined. Reentrancy would not happen as we control the caller.

RBN-8:Risk of incorrectly setting rebornToken

Category	Severity	Code Reference	Status	Contributor
Code Style	**Low**	• **code/src/RebornPortal.sol#L37**	**Fixed**	**Kong7ych3**

Code

```
37:        rebornToken = rebornToken_;
```

Description

Kong7ych3 : The rebornToken address is set in the initialize function of the RebornPortal contract, but it does not check whether the rebornToken address passed in by the user is a non-zero address. If the user passes in the 0 address by mistake, since there is no interface for resetting the rebornToken address in this contract, this will make this contract unavailable.

Recommendation

Kong7ych3 : It is recommended to check that the `rebornToken_` parameter passed in by the user is not a 0 address in the initialize function.

Client Response

Fixed

RBN-9:The performUpkeep operation can be called ahead of time

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/src/RebornPortal.sol#L378 • code/src/RebornPortal.sol#L394 • code/src/RebornPortal.sol#L404 • code/src/RebornPortal.sol#L422	Fixed	Kong7ych3

Code

```
378:        bool dropReborn = block.timestamp >

394:                _toLastHour(block.timestamp)

404:        bool dropNative = block.timestamp >

422:                _toLastHour(block.timestamp)
```

Description

Kong7ych3 : In the RebornPortal contract, when performing `_dropReborn` and `_dropNative` operations, the drop interval will be checked first, and accPerShare and DropLastUpdate will be updated only after the check is passed.
When updating `_rebornDropLastUpdate` and `_nativeDropLastUpdate`, it performs a `_toLastHour` operation on `block.timestamp`, which will make the updated `*DropLastUpdate` value smaller than the current time. And the drop interval check still uses `block.timestamp` for comparison. This will cause the `block.timestamp > *DropLastUpdate + *DropInterval` check to be passed even if the real world has experienced less time than the drop interval.
Here is an example: Suppose the current `block.timestamp` is 1677752589, `*DropInterval` is 86400. At this time, when the performUpkeep operation is successfully executed, `*DropLastUpdate` will be updated to `_toLastHour(1677752589) == 1677751200` Theoretically, the performUpkeep operation should not be performed again until the timestamp is greater than `1677752589+86400=1677838989`, but in reality, the performUpkeep operation can be performed again if the current timestamp is greater than `1677751200+86400=1677837600`.

Recommendation

Kong7ych3 : It is recommended to perform the `_toLastHour` operation on the current time when checking the drop Interval.

Consider below fix in `RebornPortal::_dropReborn()`, `RebornPortal::_dropNative()` and `RebornPortal::checkUpkeep()` function

```
uint40(_toLastHour(block.timestamp)) > _dropConf._rebornDropLastUpdate +
_dropConf._rebornDropInterval
&
uint40(_toLastHour(block.timestamp)) > _dropConf._nativeDropLastUpdate +
_dropConf._nativeDropInterval
```

Client Response

Fixed

RBN-10:Users are able to avoid paying burn fees

Category	Severity	Code Reference	Status	Contributor
Logical	**Medium**	• **code/src/RebornPortal.sol#L550**	**Acknowledged**	**0xac**

Code

```
550:        uint256 burnAmount = (amount * 5) / 100;
```

Description

0xac : When user calls `switchPool()` function, it calls `_increaseToPool()` function to burn 5% RBT amount as fee. However, user can avoid paying burn fees by set the amount less than 19. If `(amount * 5)` is less than 100, `burnAmount` will be floored to 0. User can repeat this operation any number of times to avoid paying burn fees.

```
function _increaseToPool(uint256 tokenId, uint256 amount) internal {
        uint256 burnAmount = (amount * 5) / 100;
...
    }
```

Recommendation

0xac : Suggest to ensure the `burnAmount` is not equal to 0.

Consider below fix in the `RebornPortal._increaseToPool()` function

```
function _increaseToPool(uint256 tokenId, uint256 amount) internal {
    uint256 burnAmount = (amount * 5) / 100;

    require(burnAmount > 0);

    uint256 restakeAmount = amount - burnAmount;

    _increasePool(tokenId, restakeAmount);

    emit IncreaseToPool(msg.sender, tokenId, restakeAmount);
}
```

Client Response

The `amount` has a decimal of 10e18. Only when the real amount is low than 20 * 10e-18 will this case happen. It doesn’t matter.

RBN-11: `RebornPortal` and `RBT` cache array length outside loop

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Low	• code/src/RBT.sol#L53 • code/src/RBT.sol#L57 • code/src/RebornPortal.sol#L157 • code/src/RebornPortal.sol#L219 • code/src/RebornPortal.sol#L223	Fixed	helookslikeme

Code

```
53:         for (uint256 i = 0; i < toAdd.length; i++) {

57:         for (uint256 i = 0; i < toRemove.length; i++) {

157:         for (uint256 i = 0; i < tokenIds.length; i++) {

219:         for (uint256 i = 0; i < toAdd.length; i++) {

223:         for (uint256 i = 0; i < toRemove.length; i++) {
```

Description

helookslikeme : Caching the array length outside a loop saves reading it on each iteration, as long as the array's length is not changed during the loop.

Recommendation

helookslikeme : Avoid unnecessary read of array length in for loops can save gas

Consider below fix in the `sample.test()` function

```
uint256 len = toAdd.length
for (uint256 i = 0; i < len; i++) {
    // invariant: array's length is not changed
}
```

Client Response

Fix one case, as in the other cases, the length of the array would mostly be one.

RBN-12: `RewardVault.rebornToken` can be defined as `immutable`

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/src/RewardVault.sol#L13	Fixed	0xac

Code

```
13:    address public rebornToken;
```

Description

0xac : `rebornToken` variable is only assigned in constructor. It can reduce gas cost by define as `immutable`.

```
address public rebornToken;

constructor(address owner_, address rebornToken_) {
    if (rebornToken_ == address(0)) revert ZeroAddressSet();
    _transferOwnership(owner_);
    rebornToken = rebornToken_;
}
```

Recommendation

0xac : Suggest to redefine the `rebornToken` variable as `immutable`.

Consider below fix in the `RewardVault.rebornToken` variable

```
address public immutable rebornToken;
```

Client Response

Fixed

RBN-13: `performUpkeep` cannot airdrop native token

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/src/RebornPortal.sol#L173-L182 • code/src/RebornPortal.sol#L180	Fixed	0xac, Kong7ych3

Code

```
173:    function performUpkeep(
174:        bytes calldata performData
175:    ) external override whenNotPaused {
176:        uint256 t = abi.decode(performData, (uint256));
177:        if (t == 1) {
178:            _dropReborn();
179:        } else if (t == 2) {
180:            _dropReborn();
181:        }
182:    }

180:            _dropReborn();
```

Description

0xac : The `performUpkeep` function can only airdrop the RBT in `if..else if` code segment. It could not airdrop the native token. In `checkUpkeep` fucntion indecates that 1 means airdrop RBT, and 2 means airdrop native token. Finally, `_dropNative` fucntion has not been used.

```
function performUpkeep(
    bytes calldata performData
) external override whenNotPaused {
    uint256 t = abi.decode(performData, (uint256));
    if (t == 1) {
        _dropReborn();
    } else if (t == 2) {
        _dropReborn();
    }
}
```

```
function checkUpkeep(
    bytes calldata /* checkData */
)
    external
    view
    override
    returns (bool upkeepNeeded, bytes memory performData)
{
    if (_dropConf._dropOn == 1) {
        if (
            block.timestamp >
            _dropConf._rebornDropLastUpdate + _dropConf._rebornDropInterval
        ) {
            upkeepNeeded = true;
            performData = abi.encode(1);
        } else if (
            block.timestamp >
            _dropConf._nativeDropLastUpdate + _dropConf._nativeDropInterval
        ) {
            upkeepNeeded = true;
            performData = abi.encode(2);
        }
    }
}
```

Kong7ych3 : In the RebornPortal contract, the performUpkeep function is used to perform airdrops of Reborn tokens and Native tokens to the top 100 tvl pool. However, the _dropNative function is not executed in the performUpkeep function, which will cause the pool's accNativePerShare to fail to update.

Recommendation

0xac : Suggest to replace `_dropReborn` function to `_dropNative` function in `else if (t == 2)` code segment. Consider below fix in the `RebornPortal.performUpkeep()` function

```
function performUpkeep(
    bytes calldata performData
) external override whenNotPaused {
    uint256 t = abi.decode(performData, (uint256));
    if (t == 1) {
        _dropReborn();
    } else if (t == 2) {
        _dropNative();
    }
}
```

Kong7ych3 : It is recommended to perform `_dropNative` operation when performData is 2.

Consider below fix in the `RebornPortal::performUpkeep()` function

```
    function performUpkeep(
        bytes calldata performData
    ) external override whenNotPaused {
        uint256 t = abi.decode(performData, (uint256));
        if (t == 1) {
            _dropReborn();
        } else if (t == 2) {
            _dropNative();
        }
    }
```

Client Response

Fixed

RBN-14: `pool.accNativePerShare` divide-before-multiply loss of precision

Category	Severity	Code Reference	Status	Contributor
Language Specific	Low	• code/src/RebornPortal.sol#L414-L418	Fixed	0xac

Code

```
                pool.accNativePerShare +=
                    (((_dropConf._nativeDropRatio * address(this).balance * 3) / 200) *
                        PERSHARE_BASE) /
                    PERCENTAGE_BASE /
                    pool.totalAmount;
```

Description

0xac : The formula of calculating pool.accNativePerShare is divided first and then multiplied, resulting in loss of accuracy of the result. Assume `(_dropConf._nativeDropRatio * address(this).balance * 3)` is equal to 2199, then `((_dropConf._nativeDropRatio * address(this).balance * 3) / 200)` is equal to 10. And `((_dropConf._nativeDropRatio * address(this).balance * 3) / 200) * PERSHARE_BASE` is equal to 10000

However, `(_dropConf._nativeDropRatio * address(this).balance * 3) * PERSHARE_BASE` is equal to 21990000, then `((_dropConf._nativeDropRatio * address(this).balance * 3) * PERSHARE_BASE / 200)` is equal to 109950.

Consider below POC contract

```
pool.accNativePerShare +=
    (((_dropConf._nativeDropRatio * address(this).balance * 3) / 200) *
        PERSHARE_BASE) /
    PERCENTAGE_BASE /
    pool.totalAmount;
```

Recommendation

0xac : Consider below fix in the `RebornPortal._dropNative()` function

```
pool.accNativePerShare +=
    (((_dropConf._nativeDropRatio * address(this).balance * 3) ) * PERSHARE_BASE) /
    200 /
    PERCENTAGE_BASE /
    pool.totalAmount;
```

Client Response

Fixed

Disclaimer