Smart Contract Security Audit Report

For

Galaxy

Date Issued: July 11, 2023

Version: v1.0

Confidentiality Level: Public

Contents

1 Abstract

This report was prepared for Galaxy smart contract to identify issues and vulnerabilities in its smart contract source code. A thorough examination of Galaxy smart contracts was conducted through timely communication with Galaxy, static analysis using multiple audit tools and manual auditing of their smart contract source code.

The audit process paid particular attention to the following considerations.

- A thorough review of the smart contract logic flow
- Assessment of the code base to ensure compliance with current best practice and industry standards
- Ensured the contract logic met the client's specifications and intent
- Internal vulnerability scanning tools tested for common risks and writing errors
- Testing smart contracts for common attack vectors
- Test smart contracts for known vulnerability risks
- Conduct a thorough line-by-line manual review of the entire code base

As a result of the security assessment, issues ranging from critical to informational were identified. We recommend that these issues are addressed to ensure a high level of security standards and industry practice. The recommendations we made could have better served the project from a security perspective.

- Enhance general coding practices to improve the structure of the source code.
- Provide more comments for each function to improve readability.
- Provide more transparency of privileged activities once the agreement is in place.

2 Overview

2.1 Project Summary

Project Summary	Project Information
Name	Galaxy
Start date	July 6, 2023
End date	July 11, 2023
Platform	BNB Chain
Contract type	DeFi
Language	Solidity
File	Achievement.sol, AddressTree.sol, UserInfo.sol, GalaxyHome.sol, GalaxyLevels.sol, GalaxyMine.sol, GalaxyNodes.sol

2.2 Report HASH

Name	HASH
Galaxy	8B096CF2248AEA8CD5606C95F4990457

2.3 Audit Scope

File	SHA256
Achievement.sol	BC7F05BBF74A2AD0783DA4CE706FC3C4E857E004232C94435DA2BA211F1E847D
AddressTree.sol	A670A4B7A34D3ECEA59B5DE5DC947AC52EC49B292E088CE5EEFCA10A6D1F80F6
UserInfo.sol	DFB62349AACDDD211713CDCA32FA409372188EF39BCABC4B162479B500C215E5
GalaxyHome.sol	533635625877AB13A398C797501A63B16F3DEDEDF7605E8269DB3BC00928A231
GalaxyLevels.sol	BFB5F9DA298E9B995BE1C23604287C6B7273F0E5AAB292F0015168A6AF8365FC
GalaxyMine.sol	8CAFC364267CCD723D41D2674C13EA33E06CE9A03B1A7B523C016B1C923AE455
GalaxyNodes.sol	89EDC5EF5D596AE9947A45CF7344B854A97B01E290FB380DDA0C37B3C3D5B4B2

3 Project contract details

3.1 Contract Overview

Achievement.sol

The contract is an abstract contract that implements the management functions of user performance, including the recording of deposits, the calculation of levels, the allocation of rewards etc. It provides functions for querying user's performance information, calculating reward allocation list, etc., and provides support for upgrade operations. Specific The specific logic of rank calculation and reward allocation needs to be implemented in the subcontract.

AddressTree.sol

The contract is an abstract contract that implements an address tree management function that establishes hierarchical relationships between addresses and records the address depth in the tree. It provides functions for querying the list of direct family addresses and the list of direct push-down addresses of addresses, and provides the user functions for users to add higher-level addresses.

UserInfo.sol

The contract is a library contract, which mainly implements the createPendingReward() method and implements the UserInfo structure.

Migrations.sol

The contract is a simple migration contract that manages the migration state of the contract. The owner of the contract can set the identifier of the last completed migration by calling the setCompleted function to set the identifier of the last completed migration.

GalaxyHome.sol

The contract implements a user level system where users can participate in the system by upgrading their level and receive rewards based on their level and contribution rewards. The contract defines constants, structures and variables that provide the ability to upgrade and receive rewards, and manage the distribution of rewards and asset recipient.

GalaxyLevels.sol

The contract mainly implements the function of storing the user's rank conditions and getting the actual rank of the user; and also implements the function of managing the user's performance by inheriting the Achievement contract implements the function of managing user performance.

GalaxyMine.sol

The contract implements the user's mining and reward collection functions. The user can participate in mining by setting the mining arithmetic, calculate the user's revenue based on the arithmetic and the network-wide arithmetic, and then transfer the revenue to the user's account through the receive reward function. The administrator can allocate the rewards to each miner proportionally by assigning the rewards function.

GalaxyNodes.sol

The contract implements the management and distribution functions of node rewards, including setting node arithmetic, calculating user revenue, receiving rewards and distributing rewards, etc. operations.

3.2 Code Overview

Achievement Contract

Function Name	Visibility	Modifiers
__Achievement_init	Internal	onlyInitializing
_setLevelRewardProps	Internal	-
levelOf	Public	-
childrenAchievementsOf	External	-
distrubutionRewards	External	-
distrubutionsForefathers	Public	-
_increase	Internal	-
levelUpgrade	External	-
pendingLevelOf	Public	-
whenLevelUpgraded	Internal	-

AddressTree Contract

Function Name	Visibility	Modifiers
initialize	Public	initializer
getForefathers	External	-
childrenOf	External	-
makeRelation	External	-
_makeRelationFrom	Internal	-

GalaxyHome Contract

Function Name	Visibility	Modifiers
initialize	Public	initializer
_rewardIncreasedHandle	Internal	-
setStaticRewardPerday	External	onlyRole(MANAGER_ROLE)
setAssetReceiptor	External	onlyRole(MANAGER_ROLE)
setAccountStart	External	onlyRole(MANAGER_ROLE)
_getParentOfDeep	Internal	-
upgrade	External	-
earnedStatic	Public	-
earnedTotal	Public	-
takeReward	External	-

GalaxyLevels Contract

Function Name	Visibility	Modifiers
initialize	Public	initializer
setLevelRewardProps	External	onlyRole(MANAGER_ROLE)
setAccountLevel	External	onlyRole(MANAGER_ROLE)
pendingLevelOf	Public	-
updateStartDelegate	External	onlyRole(DELEGATE_ROLE)
increaseDelegate	External	onlyRole(DELEGATE_ROLE)

GalaxyMine Contract

Function Name	Visibility	Modifiers
initialize	Public	initializer
earned	Public	-
takeReward	External	-
setMinerDelegate	External	onlyRole(DELEGATE_ROLE)
distrubutionReward	External	-

GalaxyMine Contract

Function Name	Visibility	Modifiers
initialize	Public	initializer
earned	Public	-
takeReward	External	-
setNoderPower	External	onlyRole(DELEGATE_ROLE)
distrubutionReward	External	-

4 Audit results

4.1 Key messages

ID	Title	Severity	Status
01	Privileged role	Low	confirm
02	Redundant codes	Informational	confirm
03	Insecure order of transfers	Low	fixed
04	Possible underfunding of transfers	Informational	confirm
05	Can add superiors maliciously	Low	confirm

4.2 Audit details

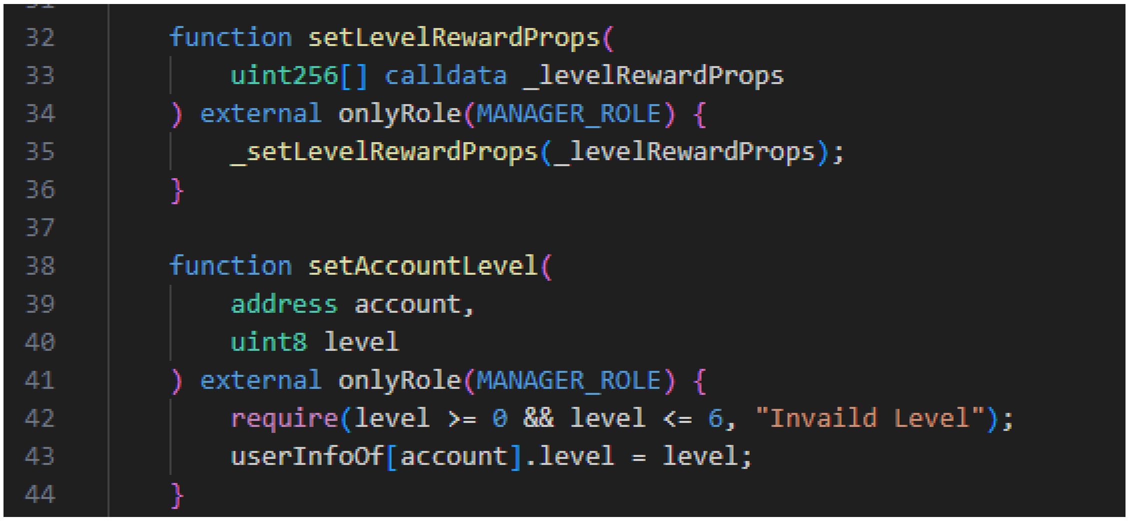
4.2.1 Privileged role

ID	Severity	Location	Status
01	Low	GalaxyLevels.sol: 32, 44	confirm

Description

Privileged roles can update the levelRewardProps variable and userInfoOf[account].level via setLevelRewardProps() and setAccountLevel(), which denote the level of rewards, respectively, and may result in larger rewards if the privileged roles are maliciously controlled or level increase.

Code location:



```
32      function setLevelRewardProps(
33          uint256[] calldata _levelRewardProps
34      ) external onlyRole(MANAGER_ROLE) {
35          _setLevelRewardProps(_levelRewardProps);
36      }
37
38      function setAccountLevel(
39          address account,
40          uint8 level
41      ) external onlyRole(MANAGER_ROLE) {
42          require(level >= 0 && level <= 6, "Invaild Level");
43          userInfoOf[account].level = level;
44      }
```

Recommendation

It is recommended to use multi-signature to manage privileged roles in project contracts.

Status

confirm.

Will recommend customers to use multi-sign contracts.

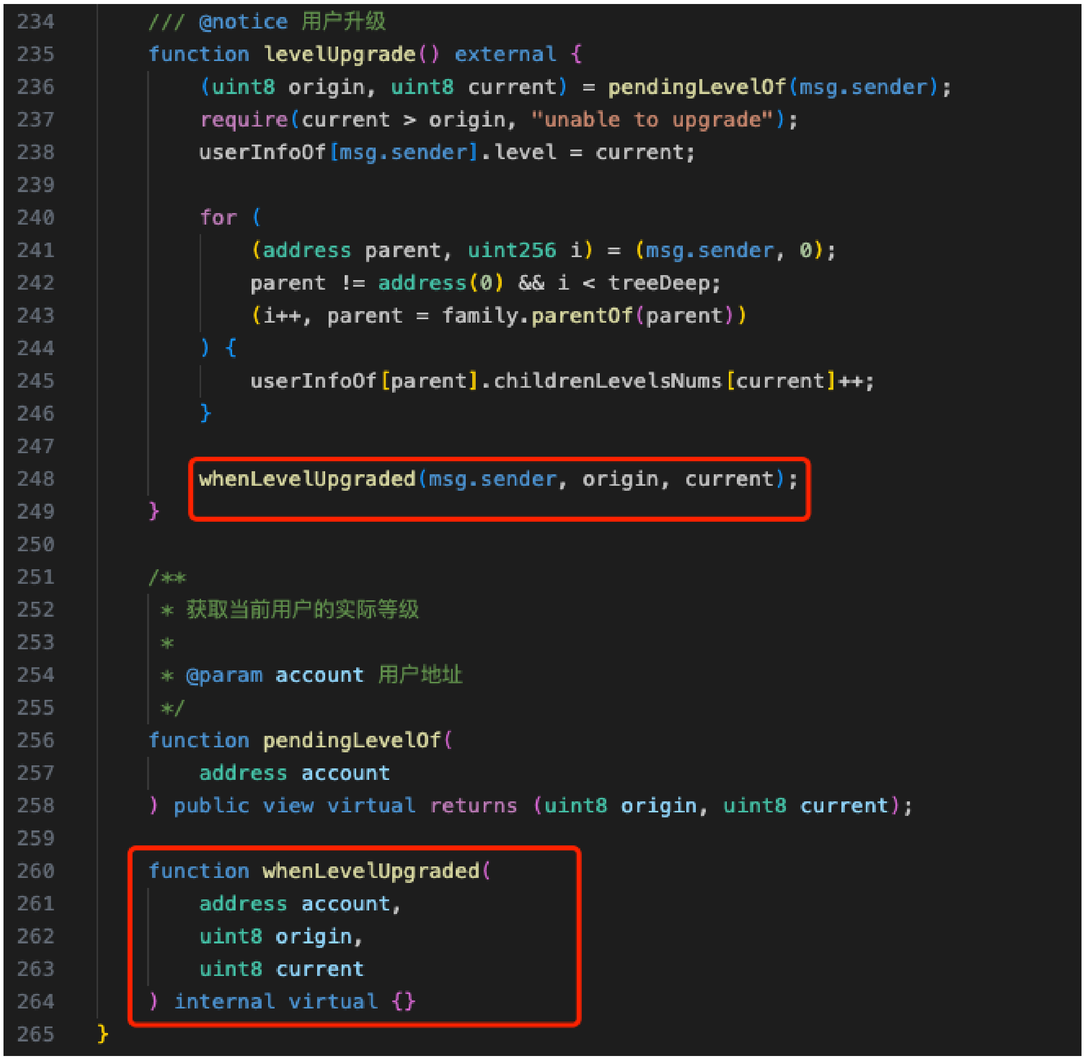
4.2.2 Redundant codes

ID	Severity	Location	Status
02	Informational	Achievement.sol: 260, 264	confirm

Description

The levelUpgrade() method calls the whenLevelUpgraded() method, but there is no contract-specific logic in that method, and if no logic is written for that method, the method is redundant code.

Code location:



```
234      /// @notice 用户升级
235      function levelUpgrade() external {
236          (uint8 origin, uint8 current) = pendingLevelOf(msg.sender);
237          require(current > origin, "unable to upgrade");
238          userInfoOf[msg.sender].level = current;
239
240          for (
241              (address parent, uint256 i) = (msg.sender, 0);
242              parent != address(0) && i < treeDeep;
243              (i++, parent = family.parentOf(parent))
244          ) {
245              userInfoOf[parent].childrenLevelsNums[current]++;
246          }
247
248          whenLevelUpgraded(msg.sender, origin, current);
249      }
250
251      /**
252       * 获取当前用户的实际等级
253       *
254       * @param account 用户地址
255       */
256      function pendingLevelOf(
257          address account
258      ) public view virtual returns (uint8 origin, uint8 current);
259
260      function whenLevelUpgraded(
261          address account,
262          uint8 origin,
263          uint8 current
264      ) internal virtual {}
265  }
```

Recommendation

It is recommended to remove method code in the contract that does not implement the logic.

Status

confirm

Achievement.sol is an abstract contract and can't be used directly, the official code used in this project is GalaxyLevel.sol, whenLevelUpgraded is implemented as a dummy method in the sub-contract, so it's not redundant code. Its role is to handle some additional business logic after the user has finished upgrading.

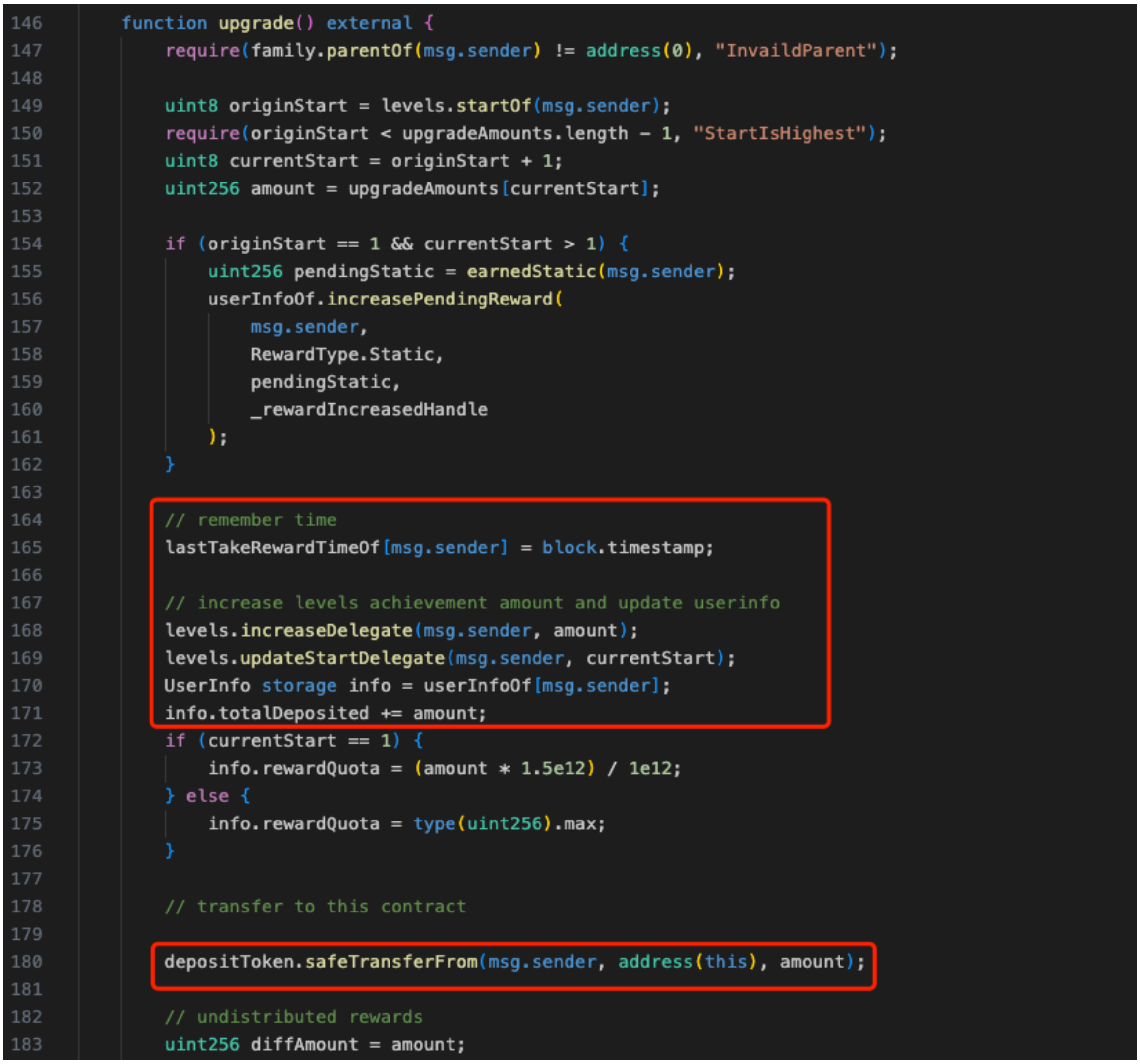
4.2.3 Insecure order of transfers

ID	Severity	Location	Status
03	Low	GalaxyHome.sol: 165, 180	fixed

Description

The user transfer is received in the upgrade() method, but since the mode of this method is similar to a deposit operation, updating the deposit status when the user has not transferred the funds to the contract may result in a reentry situation. So it is safer to update the variable when the user funds are transferred.

Code location:



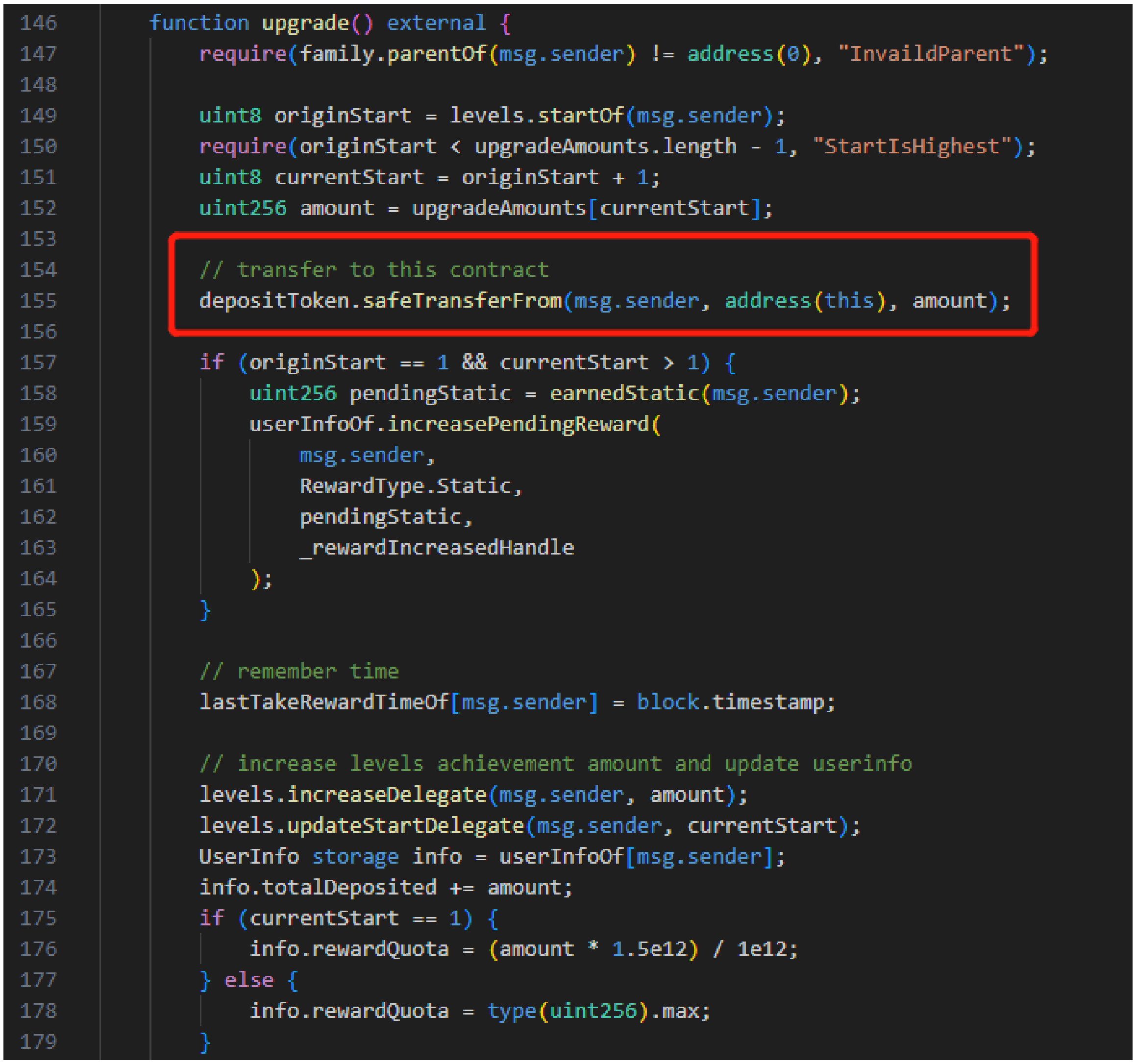
```
146     function upgrade() external {
147         require(family.parentOf(msg.sender) != address(0), "InvaildParent");
148 
149         uint8 originStart = levels.startOf(msg.sender);
150         require(originStart < upgradeAmounts.length - 1, "StartIsHighest");
151         uint8 currentStart = originStart + 1;
152         uint256 amount = upgradeAmounts[currentStart];
153 
154         if (originStart == 1 && currentStart > 1) {
155             uint256 pendingStatic = earnedStatic(msg.sender);
156             userInfoOf.increasePendingReward(
157                 msg.sender,
158                 RewardType.Static,
159                 pendingStatic,
160                 _rewardIncreasedHandle
161             );
162         }
163 
164         // remember time
165         lastTakeRewardTimeOf[msg.sender] = block.timestamp;
166 
167         // increase levels achievement amount and update userinfo
168         levels.increaseDelegate(msg.sender, amount);
169         levels.updateStartDelegate(msg.sender, currentStart);
170         UserInfo storage info = userInfoOf[msg.sender];
171         info.totalDeposited += amount;
172         if (currentStart == 1) {
173             info.rewardQuota = (amount * 1.5e12) / 1e12;
174         } else {
175             info.rewardQuota = type(uint256).max;
176         }
177 
178         // transfer to this contract
179 
180         depositToken.safeTransferFrom(msg.sender, address(this), amount);
181 
182         // undistributed rewards
183         uint256 diffAmount = amount;
```

Recommendation

It is recommended to transfer the money first and then go ahead and update the user's status to avoid any security issues.

Status

fixed.



```
    function upgrade() external {
        require(family.parentOf(msg.sender) != address(0), "InvaildParent");

        uint8 originStart = levels.startOf(msg.sender);
        require(originStart < upgradeAmounts.length - 1, "StartIsHighest");
        uint8 currentStart = originStart + 1;
        uint256 amount = upgradeAmounts[currentStart];

        // transfer to this contract
        depositToken.safeTransferFrom(msg.sender, address(this), amount);

        if (originStart == 1 && currentStart > 1) {
            uint256 pendingStatic = earnedStatic(msg.sender);
            userInfoOf.increasePendingReward(
                msg.sender,
                RewardType.Static,
                pendingStatic,
                _rewardIncreasedHandle
            );
        }

        // remember time
        lastTakeRewardTimeOf[msg.sender] = block.timestamp;

        // increase levels achievement amount and update userinfo
        levels.increaseDelegate(msg.sender, amount);
        levels.updateStartDelegate(msg.sender, currentStart);
        UserInfo storage info = userInfoOf[msg.sender];
        info.totalDeposited += amount;
        if (currentStart == 1) {
            info.rewardQuota = (amount * 1.5e12) / 1e12;
        } else {
            info.rewardQuota = type(uint256).max;
        }
```

4.2.4 Possible underfunding of transfers

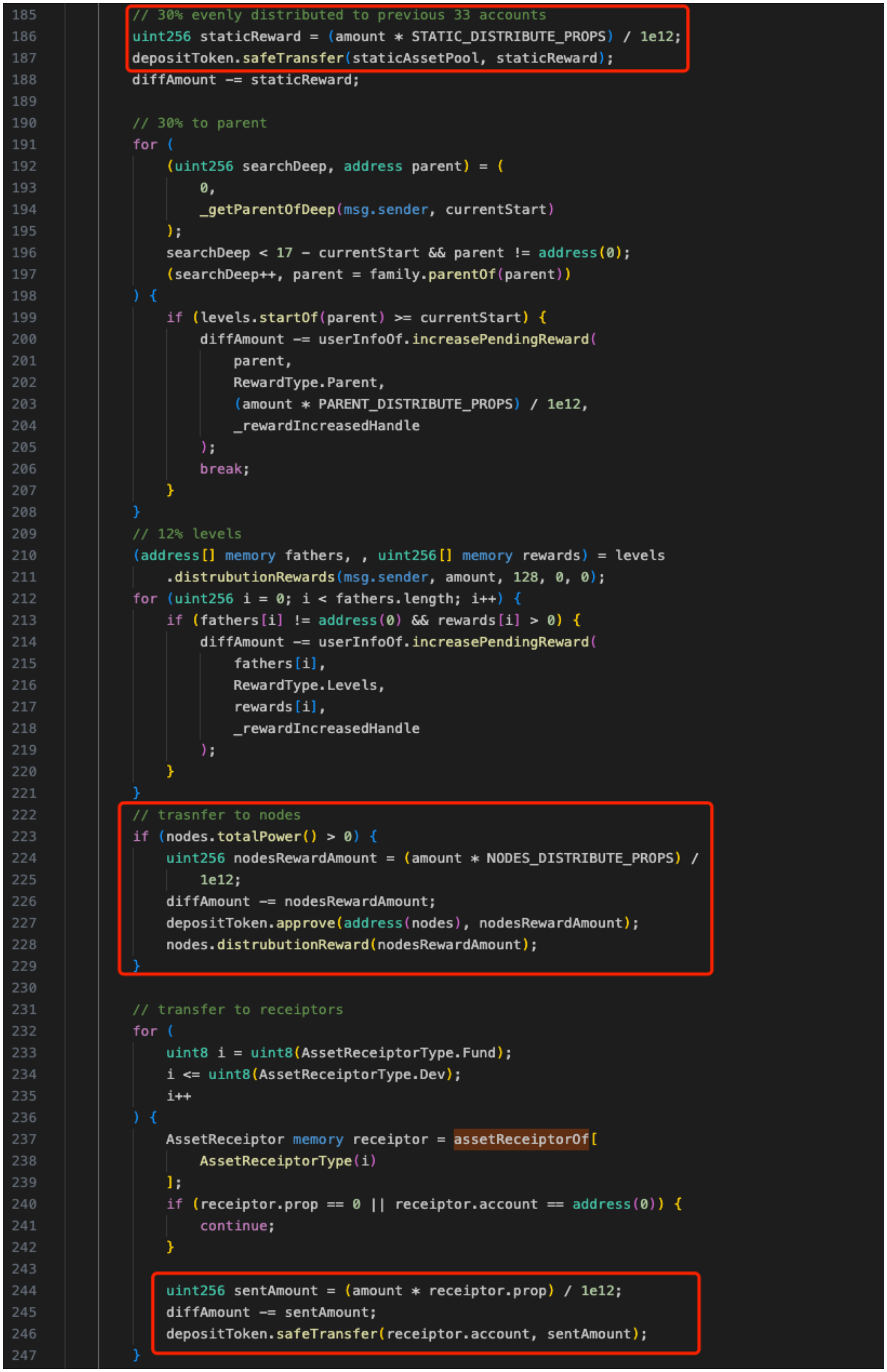
ID	Severity	Location	Status
04	Informational	GalaxyHome.sol: 185, 247	confirm

Description

If receiptor.prop is greater than 50%, it may lead to transfer away other address allocation funds, currently the contract will first transfer 30% to staticAssetPool, trasnfer to nodes funds for 20%, if here receiptor.prop is greater than 50%, due to the trasnfer to nodes funds are only authorized, the funds have not been transferred, so transferring funds at 50%-70% can transfer trasnfer to nodes funds as well. If the transferred funds are more than 70%, it may cause the transfer to fail.

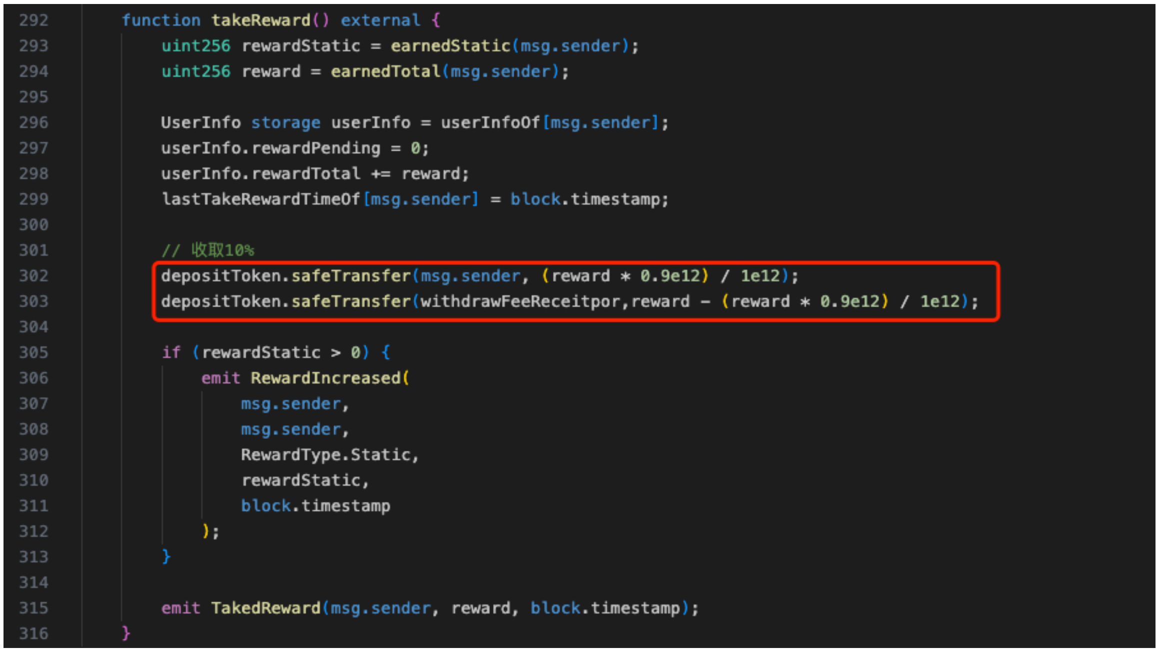
The current receiptor.prop is set by the privileged role, it is recommended to set the value not to exceed 50%.

Code location:



```
// 30% evenly distributed to previous 33 accounts
uint256 staticReward = (amount * STATIC_DISTRIBUTE_PROPS) / 1e12;
depositToken.safeTransfer(staticAssetPool, staticReward);
diffAmount -= staticReward;

// 30% to parent
for (
    (uint256 searchDeep, address parent) = (
        0,
        _getParentOfDeep(msg.sender, currentStart)
    );
    searchDeep < 17 - currentStart && parent != address(0);
    (searchDeep++, parent = family.parentOf(parent))
) {
    if (levels.startOf(parent) >= currentStart) {
        diffAmount -= userInfoOf.increasePendingReward(
            parent,
            RewardType.Parent,
            (amount * PARENT_DISTRIBUTE_PROPS) / 1e12,
            _rewardIncreasedHandle
        );
        break;
    }
}
// 12% levels
(address[] memory fathers, , uint256[] memory rewards) = levels
    .distrubutionRewards(msg.sender, amount, 128, 0, 0);
for (uint256 i = 0; i < fathers.length; i++) {
    if (fathers[i] != address(0) && rewards[i] > 0) {
        diffAmount -= userInfoOf.increasePendingReward(
            fathers[i],
            RewardType.Levels,
            rewards[i],
            _rewardIncreasedHandle
        );
    }
}
// trasnfer to nodes
if (nodes.totalPower() > 0) {
    uint256 nodesRewardAmount = (amount * NODES_DISTRIBUTE_PROPS) /
        1e12;
    diffAmount -= nodesRewardAmount;
    depositToken.approve(address(nodes), nodesRewardAmount);
    nodes.distrubutionReward(nodesRewardAmount);
}

// transfer to receiptors
for (
    uint8 i = uint8(AssetReceiptorType.Fund);
    i <= uint8(AssetReceiptorType.Dev);
    i++
) {
    AssetReceiptor memory receiptor = assetReceiptorOf[
        AssetReceiptorType(i)
    ];
    if (receiptor.prop == 0 || receiptor.account == address(0)) {
        continue;
    }

    uint256 sentAmount = (amount * receiptor.prop) / 1e12;
    diffAmount -= sentAmount;
    depositToken.safeTransfer(receiptor.account, sentAmount);
}
```

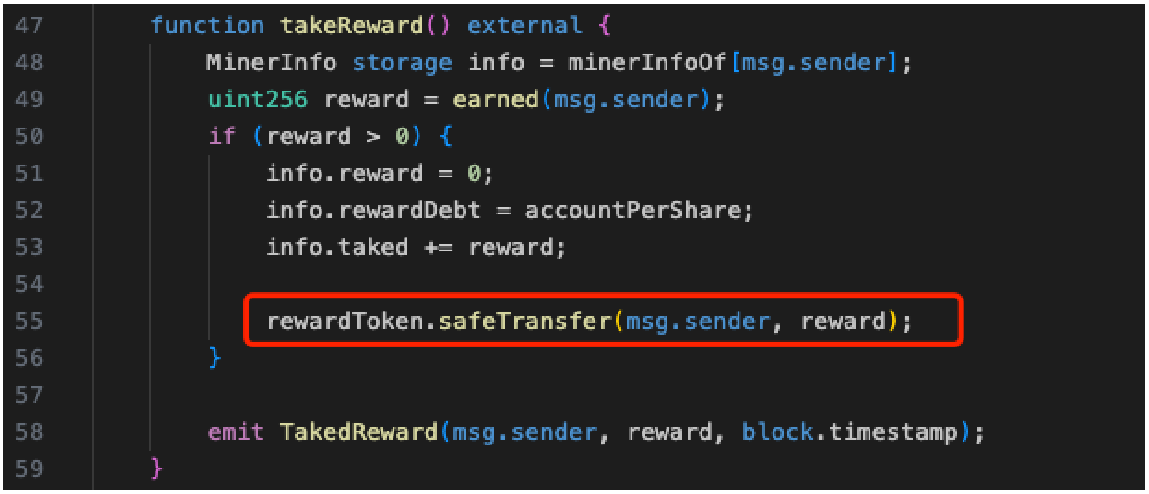
When getting rewards through this contract, since the rewards are sent directly from the contract, it is necessary to determine that the contract has a sufficient amount of funds to provide the transfer. Avoid having insufficient funds in the contract, which may result in a transfer failure.



```
    function takeReward() external {
        uint256 rewardStatic = earnedStatic(msg.sender);
        uint256 reward = earnedTotal(msg.sender);

        UserInfo storage userInfo = userInfoOf[msg.sender];
        userInfo.rewardPending = 0;
        userInfo.rewardTotal += reward;
        lastTakeRewardTimeOf[msg.sender] = block.timestamp;

        // 收取10%
        depositToken.safeTransfer(msg.sender, (reward * 0.9e12) / 1e12);
        depositToken.safeTransfer(withdrawFeeReceitpor,reward - (reward * 0.9e12) / 1e12);

        if (rewardStatic > 0) {
            emit RewardIncreased(
                msg.sender,
                msg.sender,
                RewardType.Static,
                rewardStatic,
                block.timestamp
            );
        }

        emit TakedReward(msg.sender, reward, block.timestamp);
    }
```

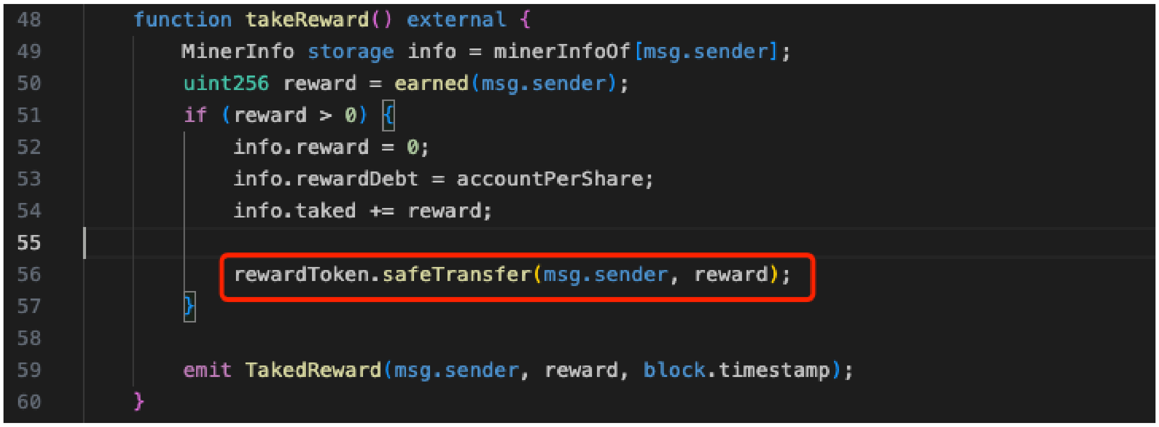
In addition to this, the GalaxyMine contract and GalaxyNodes contract also suffer from this issue.

GalaxyMine.sol



```
    function takeReward() external {
        MinerInfo storage info = minerInfoOf[msg.sender];
        uint256 reward = earned(msg.sender);
        if (reward > 0) {
            info.reward = 0;
            info.rewardDebt = accountPerShare;
            info.taked += reward;

            rewardToken.safeTransfer(msg.sender, reward);
        }

        emit TakedReward(msg.sender, reward, block.timestamp);
    }
```

GalaxyNodes.sol



```
    function takeReward() external {
        MinerInfo storage info = minerInfoOf[msg.sender];
        uint256 reward = earned(msg.sender);
        if (reward > 0) {
            info.reward = 0;
            info.rewardDebt = accountPerShare;
            info.taked += reward;

            rewardToken.safeTransfer(msg.sender, reward);
        }

        emit TakedReward(msg.sender, reward, block.timestamp);
    }
```

Recommendation

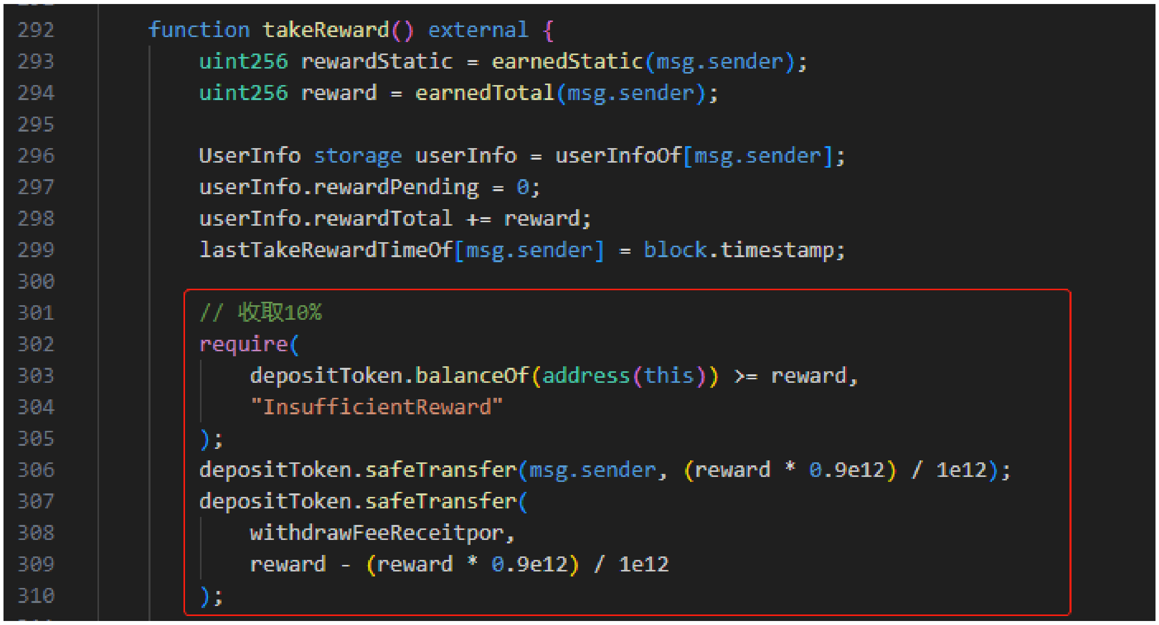
It is recommended that you subtract the funds already allocated before transferring, and judge whether the contract is fully funded at the time of transfer.

Status

confirm.

We added some assertions appropriately, but not all of them, we think the transfer fails and the transaction should roll, not adding too many assertions is in the gas consideration, because balanceOf to determine the balance is an external call that generates gas.

GalaxyHome.takeReward()-fixed



```
    function takeReward() external {
        uint256 rewardStatic = earnedStatic(msg.sender);
        uint256 reward = earnedTotal(msg.sender);

        UserInfo storage userInfo = userInfoOf[msg.sender];
        userInfo.rewardPending = 0;
        userInfo.rewardTotal += reward;
        lastTakeRewardTimeOf[msg.sender] = block.timestamp;

        // 收取10%
        require(
            depositToken.balanceOf(address(this)) >= reward,
            "InsufficientReward"
        );
        depositToken.safeTransfer(msg.sender, (reward * 0.9e12) / 1e12);
        depositToken.safeTransfer(
            withdrawFeeReceitpor,
            reward - (reward * 0.9e12) / 1e12
        );
```

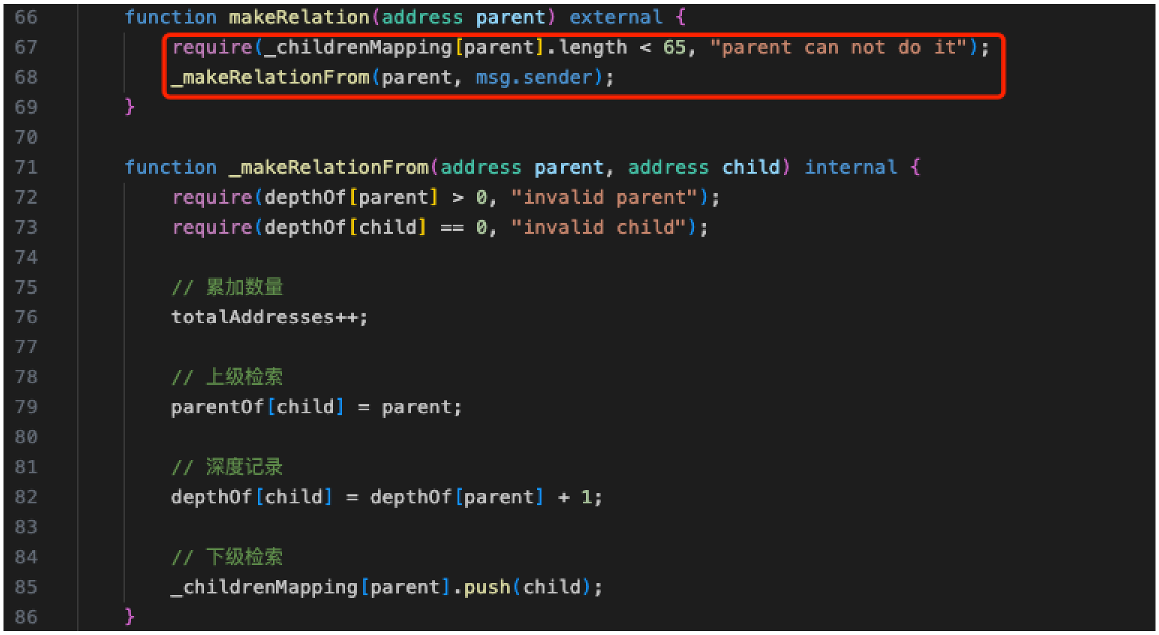
4.2.5 Can add superiors maliciously

ID	Severity	Location	Status
05	Low	AddressTree.sol: 66, 86	confirm

Description

The makeRelation() method is used for users to add superiors, but since there is a length limitation for adding superiors, if a malicious user adds a certain superior in bulk, it may result in other users not being able to continue adding.

Code location:



```
66      function makeRelation(address parent) external {
67          require(_childrenMapping[parent].length < 65, "parent can not do it");
68          _makeRelationFrom(parent, msg.sender);
69      }
70
71      function _makeRelationFrom(address parent, address child) internal {
72          require(depthOf[parent] > 0, "invalid parent");
73          require(depthOf[child] == 0, "invalid child");
74
75          // 累加数量
76          totalAddresses++;
77
78          // 上级检索
79          parentOf[child] = parent;
80
81          // 深度记录
82          depthOf[child] = depthOf[parent] + 1;
83
84          // 下级检索
85          _childrenMapping[parent].push(child);
86      }
```

Recommendation

Modify the logic of adding superiors to avoid a situation where a superior address is used maliciously.

Status

confirm.

Keep it the same, malicious attacks cost money. If a malicious attack still occurs, the contract can be updated to remove this restriction, or the restriction can be removed outright, which needs to be determined with the requirements.

5 Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Disclaimer

This report is issued in response to facts that occurred or existed prior to the issuance of this report, and liability is assumed only on that basis.
Shield Security cannot determine the security status of this program and assumes no responsibility for facts occurring or existing after the date of this report. The security audit analysis and other content in this report is based on documents and materials provided to Shield Security by the information provider through the date of the insurance report. in Shield Security's opinion. The information provided is not missing, falsified, deleted or concealed. If the information provided is missing, altered, deleted, concealed or not in accordance with the actual circumstances, Shield Security shall not be liable for any loss or adverse effect resulting therefrom. shield Security will only carry out the agreed security audit of the security status of the project and issue this report. shield Security is not responsible for the background and other circumstances of the project. Shield Security is not responsible for the background and other circumstances of the project.