Smart Contract Security Audit Report

For

Pandora

Date Issued: November 18, 2024

Version: v2.0

Confidentiality Level: Public

Contents

1 Abstract

This report was prepared for Pandora smart contract to identify issues and vulnerabilities in its smart contract source code. A thorough examination of Pandora smart contracts was conducted through timely communication with Pandora, 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	Pandora
Start date	October 29, 2024
End date	November 18, 2024
Contract type	Token ,DeFi
Language	Solidity

2.2 Report HASH

Name	Hash
Pandora_system_contracts_20241025.zip	6D6C6424FCA170B2CD81891AF503293459DB04A9A2123CBD5912393B6C9C12C2

3 Project contract details

3.1 Contract Overview

SortedList.sol

SortedList is a Solidity library for managing ordered lists of addresses on the blockchain. It implements a linked list structure sorted by value. Through this library, multiple addresses can be sorted in descending order to quickly get the first few addresses with the highest values. The main functions include: inserting and updating, deleting nodes, getting the first K addresses, and auxiliary functions.

WPDALock.sol

WPDATokenLock is a smart contract for block reward distribution. By setting the reward amount for each block, the reward is sent to the specified reward pool address poolAddress regularly. The main functions include: block reward setting. Block trigger distribution. Funds receiving and event recording.

WPDAToken.sol

WPDAToken is a smart contract based on the ERC20 standard, representing a redeemable token called "Wrapped PDA Token" (WPDA) that supports the deposit and withdrawal of Ethereum. The main functions include that users can call the deposit or depositTo function to deposit a certain amount of Ethereum and receive an equal amount of WPDA tokens. Users can call the withdraw or withdrawTo function to redeem WPDA tokens back to an equal amount of Ethereum.

ZEUSLock.sol

ZEUSTokenLock is a smart contract for the periodic release and reward distribution of ZEUS tokens. The contract sets the token release rules to release tokens monthly in different stages, which is suitable for scenarios with incentive mechanisms and phased releases. The main functions include: Token release plan: After calling startRelease, tokens are released monthly in four stages, releasing 10%, 5%, 3%, and 1% of the total supply respectively. Release information update: Update the number of tokens released monthly and per second every 30 days to adapt to the release needs of different stages. Reward distribution: When the mintToFarm method meets the release interval conditions, half of the released tokens are distributed to the holders and half to the community. Security control: Only the contract owner can start the release plan and set the distribution interval, and the distribution function can only be called by specific addresses.

ZEUSToken.sol

ZEUSToken is a token contract based on ERC20 that supports reward allocation and account equity management. This contract is suitable for application scenarios with token holding rewards and power limits. The main functions include: Reward allocation: Through earned and claim functions, users can obtain corresponding rewards based on the number of tokens held. Minting permissions: The lockerMint method allows the specified lockerAddress to mint tokens. Account power limit: Set the maximum power value of the account, accountPowerLimit, to control the impact of the amount of tokens held by a single account on the reward. Dynamic update: Automatically update the power value of the account after the token transfer to ensure the accuracy of the reward distribution.

Access.sol

Access is an abstract contract for role management and access control. It inherits from AccessControl and Initializable, and sets multiple permission roles in smart contracts. The main functions include: Role management: Initialize roles through __Access_init, set default administrator (DEFAULT_ADMIN_ROLE), agent (DELEGATE_ROLE), manager (MANAGER_ROLE) and developer (DEVELOPER_ROLE), and define the hierarchical relationship of roles. onlyCoinbase modifier: restricts functions to be called only by miners who produce blocks.onlyNullAddress modifier: restricts functions to be called only by zero addresses.onlySysDao modifier: restricts functions to be called only by SYSTEM_DAO contract addresses.Contract address detection: provides isContract function to determine whether an address is a contract address.

AddressTree.sol

AddressTree is a smart contract for managing address hierarchical relationships, recording the hierarchical relationship of addresses through a tree structure. The main functions include: Address relationship management: allows the specification of parent-child relationships, using methods such as makeRelation and importRelation to set one address as the parent of another address, thereby building an address tree. Hierarchical query: records the depth and parent address relationship of each address, and supports getForefathers to query the multi-layer parent relationship of the specified address. Event recording: When establishing a relationship, the addition of parent-child relationships is recorded through the AddressAdded event for easy tracking and query. Access control: inherited from the Access contract, restricts important operations (such as relationship import) to be performed only by specific roles or block miners to ensure security.

Farm.sol

Farm is a smart contract for managing reward pools, distributing rewards to token holders in a variety of ways. The main functions include: Reward pool management: supports adding multiple reward pools (addPool), each pool can set reward tokens and corresponding distribution rules. Set the contract address of the WPDA reward token through setWPDATokenAddress. Holder and community rewards: Holders and communities have independent reward calculation and collection mechanisms (earned, takeReward), and trigger the RewardTaked event when they are collected. Reward distribution: The handleBlock function distributes rewards when a new block is generated, calls the mintToFarm method of each reward pool to calculate and distribute rewards, and records distribution information through the DistributeRewards event. Access control: Inherited from the Access contract, only administrators or specific roles can call the pool management and reward distribution functions to ensure the security and operation permissions of the contract.

SystemDao.sol

SystemDao is a smart contract that manages system parameters and block event subscriptions. It supports basic parameter settings, blacklist management, and contract creator permission control. The main functions include: Parameter management: set the base gas price (baseGasPrice), block rewards (blockRewards), the minimum value of node transfer (makeNodeTransferValue), block fee rate (blockFeeRate), and the recipient. Blacklist and permission management: manage blacklist addresses (blackList) to restrict disallowed accounts. Manage the permissions of contract creators to limit who can deploy contracts. Block subscription: supports adding and removing block event subscribers (blockSubscribers), calling handleBlock when each new block is generated, triggering the subscriber's block processing logic. Deployment mode setting: can be switched to private deployment mode (isPrivateDeploymentMode), allowing only specific creators to deploy contracts.

Validator.sol

Validator is a smart contract for managing node delegation and reward distribution, including multi-level node and delegator incentive mechanisms. The following are the main functions of this contract:Node delegation mechanism: Users can delegate tokens to validators through the delegatePda function, increase the weight of the validator, and obtain delegation-based rewards. undelegatePda allows users to cancel delegation, retrieve their tokens and claim unwithdrawn rewards. Reward distribution: Each block automatically distributes rewards to validators, witnesses, and delegators, which is implemented through the deposit function. Rewards are distributed according to the delegation amount and the effective supply, ensuring that rewards are distributed to participants in proportion.Node and witness management: Use currentValidators and currentWitnesses to manage current node and witness snapshots, and update them at the end of each reward cycle. claimCopartnerRewards is used for witnesses to claim rewards. Deduplication prevention and security control: Use the noReentrant modifier to prevent reentry attacks and ensure the security of delegation and delegating. Configuration management: Support setDao to configure the DAO management address to ensure decentralized control of the contract.

4 Audit results

4.1 Key messages

ID	Title	Severity	Status
01	The deposit method may be exploited	High	Fixed
02	There are instant deposits and withdrawals and receive rewards	High	Fixed
03	A creator that has been changed to false cannot be changed to true again	Medium	Fixed
04	Wrong deletion location	Medium	Fixed
05	setLockAddress can be set preemptively	Medium	Ignore
06	forkInit cannot be executed	Medium	Ignore
07	The funds users vote on can be consumed by rewards	Medium	Ignore
08	The initErc20Permit() method did not find a specific implementation	Medium	Ignore
09	Preemptive Initialization	Low	Ignore
10	You can add a parent to any address	Low	Ignore
11	No checks are performed on the length of the parents and children arrays	Low	Fixed
12	It is possible to make the addresses of the upper and lower levels the same	Low	Fixed
13	Preemptive Initialization	Low	Ignore
14	There is no limit on the value of privileged role settings	Low	Ignore
15	The same subscriber address is added multiple times	Low	Fixed
16	The claimDelegatorRewards and other methods do not add modifiers to prevent reentry	Low	Ignore
17	Strict maturity height matching problem	Low	Ignore
18	The transferOrder method does not check if the receiver address is the zero address	Low	Fixed
19	initialize has the risk of preemptive initialization	Low	Fixed

20	Inflexible time or block height control mechanism	Low	Ignore
21	The delegatePda method lacks validity check for the validator	Low	Fixed
22	No check for zero address in setDao method	Low	Fixed
23	Calculation accuracy issues	Low	Ignore
24	Unrecognized chainID causes accountPowerLimitSetter to always be empty	Low	Ignore
25	The calculation of totalPower - originPower may be wrong	Low	Ignore
26	Privileged role management	Low	Ignore
27	There is no suspension mechanism in the project contract	Low	Ignore
28	The claimDelegatorRewards method can be called by any user	Low	Fixed
29	Unused onlyNullAddress modifier	Informational	Fixed
30	Lack of logging and error handling in handleBlock method	Informational	Ignore

4.2 Audit details

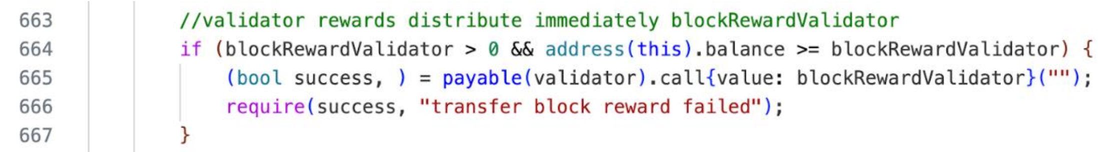
4.2.1 The deposit method may be exploited

ID	Severity	Location	Status
1	High	Validator.sol	Fixed

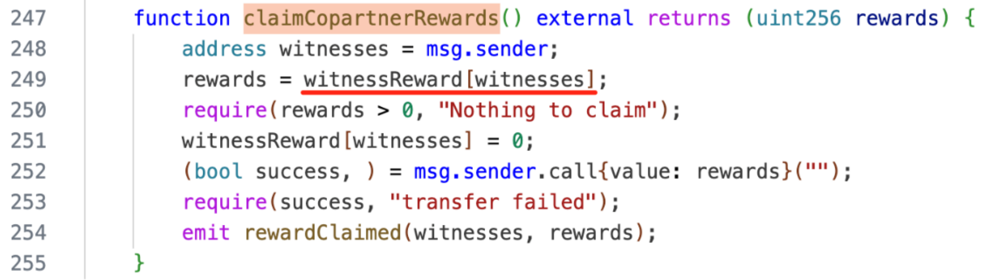
Description

The deposit method is called through block.coinbase (miner or validator). If the permission calls this method multiple times, the funds of the contract will be exhausted. This method has two fund transfers:

Code location :



```
//validator rewards distribute immediately blockRewardValidator
if (blockRewardValidator > 0 && address(this).balance >= blockRewardValidator) {
    (bool success, ) = payable(validator).call{value: blockRewardValidator}("");
    require(success, "transfer block reward failed");
}
```



```
function claimCopartnerRewards() external returns (uint256 rewards) {
    address witnesses = msg.sender;
    rewards = witnessReward[witnesses];
    require(rewards > 0, "Nothing to claim");
    witnessReward[witnesses] = 0;
    (bool success, ) = msg.sender.call{value: rewards}("");
    require(success, "transfer failed");
    emit rewardClaimed(witnesses, rewards);
}
```

Contract funds can be collected from both locations. Especially for the first transfer, as long as there is funds in the contract and blockRewardValidator is not zero, miners can continue to consume contract funds.

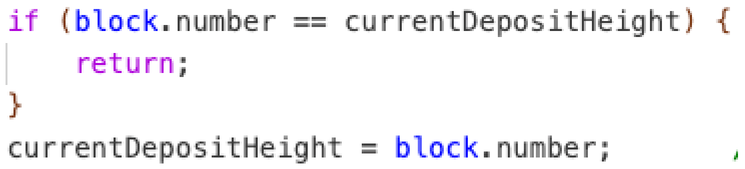
Recommendation

Add permissions and judgments to avoid multiple withdrawals of funds.

Status

Fixed.

Added logic to prevent duplicate calls within a block.



```
if (block.number == currentDepositHeight) {
    return;
}
currentDepositHeight = block.number;
```

4.2.2 There are instant deposits and withdrawals and receive rewards

ID	Severity	Location	Status
2	High	Validator.sol	Fixed

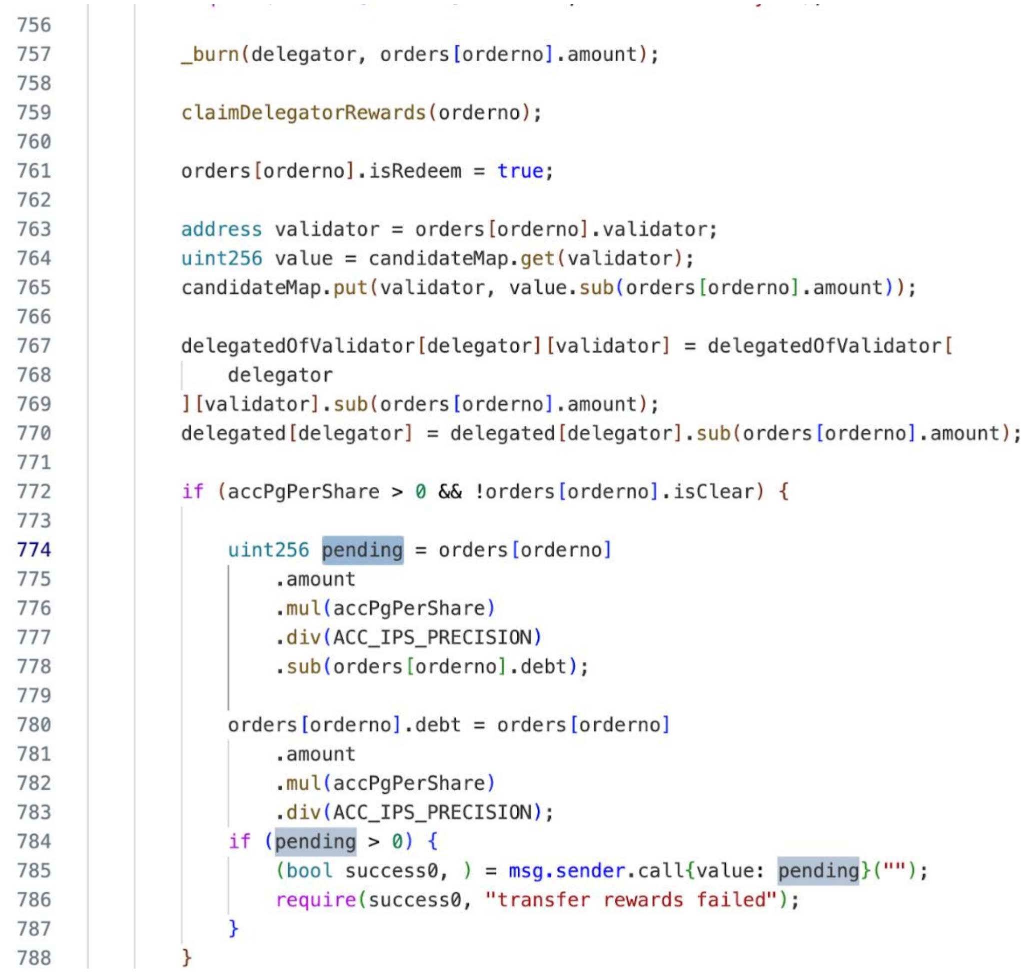
Description

The delegatePda method allows users to vote and create a voting order.

The undelegatePda method allows users to revoke their votes and receive rewards.

There is a situation where users can continue to call the undelegatePda method after calling the delegatePda method to vote in a transaction. Users can revoke their votes and receive rewards. In this case, the funds of the contract can be exhausted.

Code location:



```
        _burn(delegator, orders[orderno].amount);

        claimDelegatorRewards(orderno);

        orders[orderno].isRedeem = true;

        address validator = orders[orderno].validator;
        uint256 value = candidateMap.get(validator);
        candidateMap.put(validator, value.sub(orders[orderno].amount));

        delegatedOfValidator[delegator][validator] = delegatedOfValidator[
            delegator
        ][validator].sub(orders[orderno].amount);
        delegated[delegator] = delegated[delegator].sub(orders[orderno].amount);

        if (accPgPerShare > 0 && !orders[orderno].isClear) {

            uint256 pending = orders[orderno]
                .amount
                .mul(accPgPerShare)
                .div(ACC_IPS_PRECISION)
                .sub(orders[orderno].debt);

            orders[orderno].debt = orders[orderno]
                .amount
                .mul(accPgPerShare)
                .div(ACC_IPS_PRECISION);
            if (pending > 0) {
                (bool success0, ) = msg.sender.call{value: pending}("");
                require(success0, "transfer rewards failed");
            }
        }
```

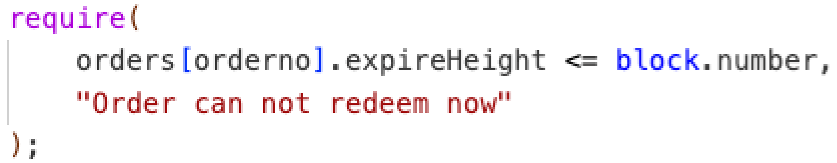
Recommendation

Add a mechanism to avoid instantaneous collection.

Status

Fixed.

Added lock-up period judgment, and redemption can only be made after the lock-up height has been reached.



```
require(
    orders[orderno].expireHeight <= block.number,
    "Order can not redeem now"
);
```

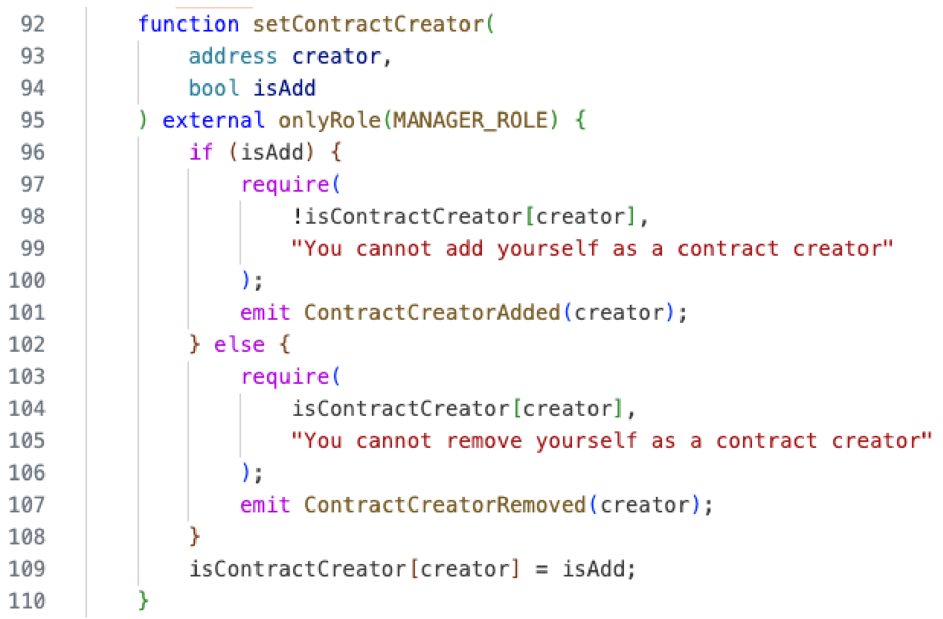
4.2.3 A creator that has been changed to false cannot be changed to true again

ID	Severity	Location	Status
03	Medium	SystemDao.sol	Fixed

Description

Once isContractCreator[creator] is set to false, even if you call setContractCreator later to try to set it to true, the require condition will be triggered and fail because isContractCreator[creator] is false at this time.

Code location:



```
function setContractCreator(
    address creator,
    bool isAdd
) external onlyRole(MANAGER_ROLE) {
    if (isAdd) {
        require(
            !isContractCreator[creator],
            "You cannot add yourself as a contract creator"
        );
        emit ContractCreatorAdded(creator);
    } else {
        require(
            isContractCreator[creator],
            "You cannot remove yourself as a contract creator"
        );
        emit ContractCreatorRemoved(creator);
    }
    isContractCreator[creator] = isAdd;
}
```

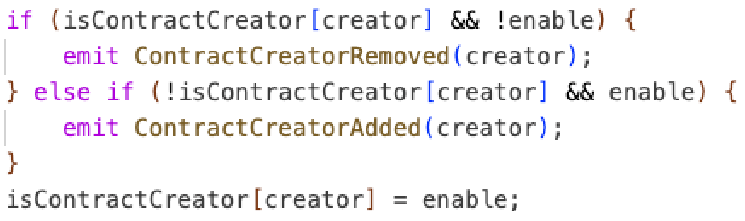
Recommendation

It is recommended to adjust the require logic so that the require check when isAdd is true only throws an error when isContractCreator[creator] == true.

Status

Fixed.

The client has changed the setup logic.



```
if (isContractCreator[creator] && !enable) {
    emit ContractCreatorRemoved(creator);
} else if (!isContractCreator[creator] && enable) {
    emit ContractCreatorAdded(creator);
}
isContractCreator[creator] = enable;
```

4.2.4 Wrong deletion location

ID	Severity	Location	Status
4	Medium	SystemDao.sol	Fixed

Description

The logic of the removeBlockSubscriber method is as follows:

The for loop traverses the blockSubscribers array and finds the subscriber that matches the passed parameter subscriber. If a subscriber that matches the passed subscriber is found at a certain position i in the array,

replace blockSubscribers[i] (the currently matched subscriber) with blockSubscribers[blockSubscribers.length - 1] (the last element in the array). This means that the currently found subscriber will be replaced by the last element of the array. Then, use the pop() method to delete the last element in the array. This step actually deletes the last element used to replace it, not the original matched element.

Example:

Suppose the blockSubscribers array has 5 elements and a length of 5:

blockSubscribers = [S1, S2, S3, S4, S5]

Suppose we want to delete S2 (that is, blockSubscribers[1] == subscriber). The code will do the following:

Find the matching element blockSubscribers[1] == S2.

Assign blockSubscribers[4] (i.e. S5) to blockSubscribers[1], replacing S2.

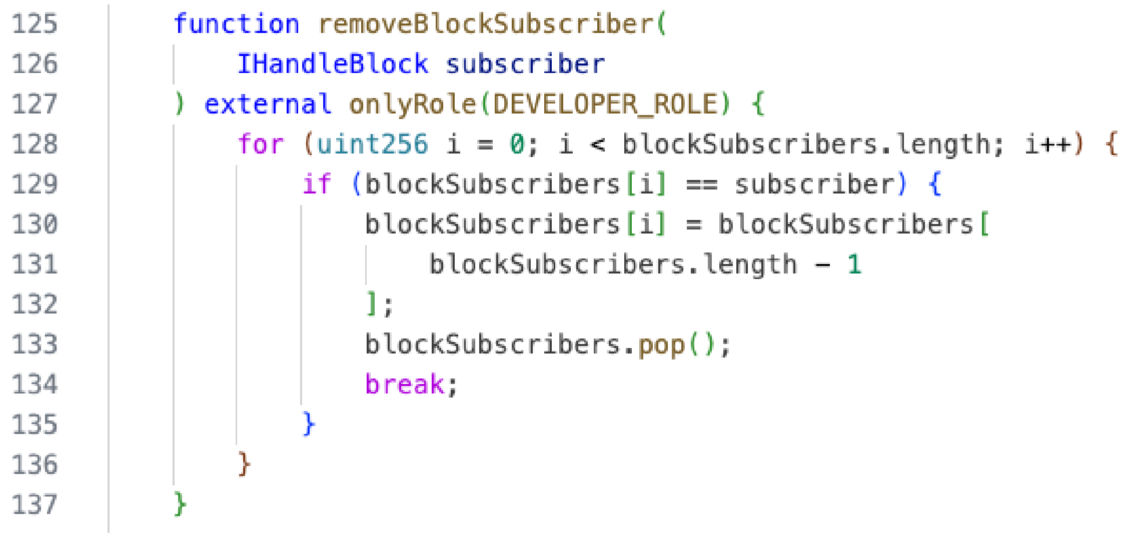
blockSubscribers = [S1, S5, S3, S4, S5]

Then pop() deletes the last element, and blockSubscribers becomes:

blockSubscribers = [S1, S5, S3, S4]

As you can see, the last element (S5) in the array is actually deleted, not the initially matched element S2..

Code location:



```
    function removeBlockSubscriber(
        IHandleBlock subscriber
    ) external onlyRole(DEVELOPER_ROLE) {
        for (uint256 i = 0; i < blockSubscribers.length; i++) {
            if (blockSubscribers[i] == subscriber) {
                blockSubscribers[i] = blockSubscribers[
                    blockSubscribers.length - 1
                ];
                blockSubscribers.pop();
                break;
            }
        }
    }
```

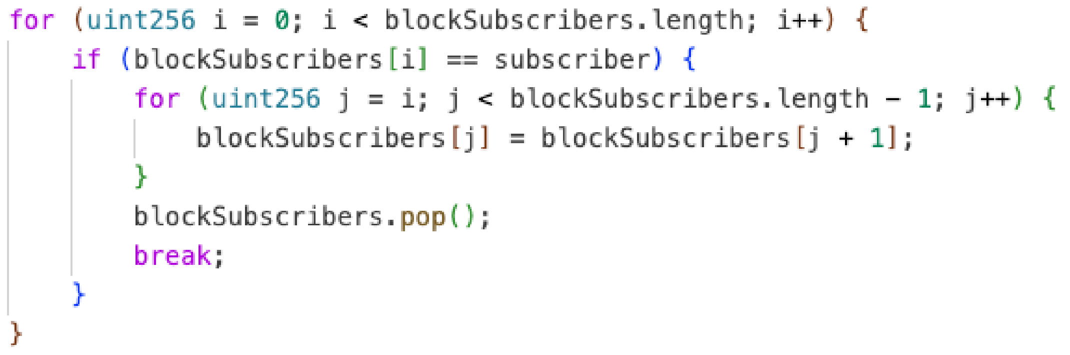
Recommendation

It is recommended to iterate over the array and shift all elements to the left, ensuring exact deletion, or use another structure.

Status

Fixed.

The client has fixed the deletion logic.



```
for (uint256 i = 0; i < blockSubscribers.length; i++) {
    if (blockSubscribers[i] == subscriber) {
        for (uint256 j = i; j < blockSubscribers.length - 1; j++) {
            blockSubscribers[j] = blockSubscribers[j + 1];
        }
        blockSubscribers.pop();
        break;
    }
}
```

4.2.5 setLockAddress can be set preemptively

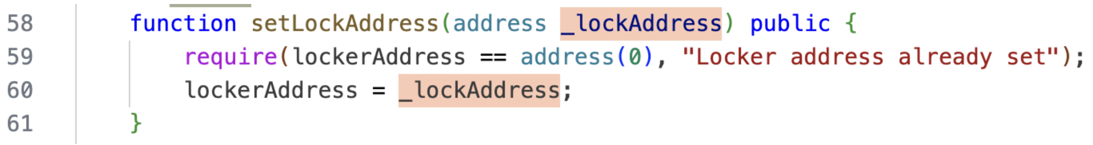
ID	Severity	Location	Status
5	Medium	tokens/ZEUSToken.sol	Ignore

Description

Since any address can call this method for the first time and set lockerAddress, lockerAddress may be set by a malicious address, thus having arbitrary mint power.

It is recommended to limit the permissions of the setLockAddress method to the contract deployer or onlyOwner to ensure that only authorized persons can set lockerAddress.

Code location:



```
function setLockAddress(address _lockAddress) public {
    require(lockerAddress == address(0), "Locker address already set");
    lockerAddress = _lockAddress;
}
```

Recommendation

It is recommended to limit the permissions of the setLockAddress method to the contract deployer or onlyOwner to ensure that only authorized persons can set lockerAddress.

Status

Ignore.

The client replied that the setting should be done immediately after the contract is deployed, and the setLockAddress method should be called at the same time..

4.2.6 forkInit cannot be executed

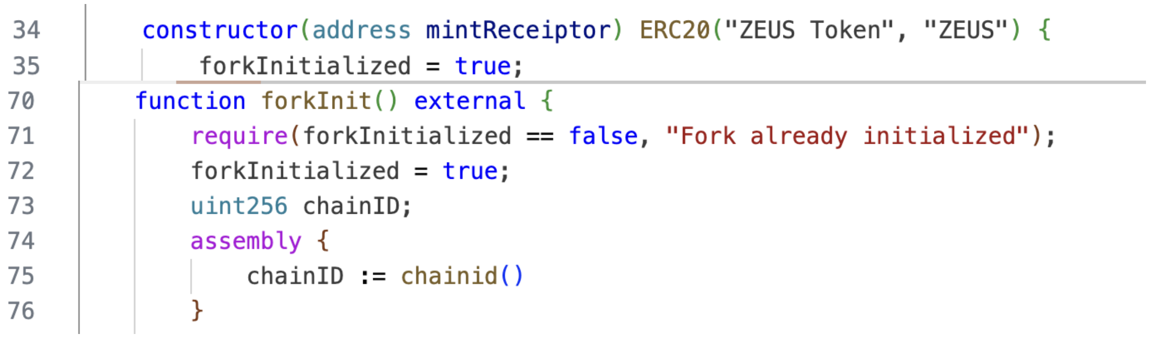
ID	Severity	Location	Status
6	Medium	tokens/ZEUSToken.sol	Ignore

Description

Initialize forkInitialized to true in the contract constructor, so that forkInit can never be executed.

If the logic of forkInit is necessary, it is recommended to set forkInitialized to false initially and set it to true only after forkInit is successfully called.

Code location:



```
    constructor(address mintReceiptor) ERC20("ZEUS Token", "ZEUS") {
        forkInitialized = true;
    function forkInit() external {
        require(forkInitialized == false, "Fork already initialized");
        forkInitialized = true;
        uint256 chainID;
        assembly {
            chainID := chainid()
        }
```

Recommendation

It is recommended to set forkInitialized to false initially and set it to true only after forkInit has been called successfully.

Status

Ignore.

The client replied that the actual environment used is that after reaching the specified block, all nodes replace the implementation of the contract at the same time and call the forkInit method at the same time. The node program ensures the continuity and call of this process.

4.2.7 The funds users vote on can be consumed by rewards

ID	Severity	Location	Status
7	Medium	Validator.sol	Ignore

Description

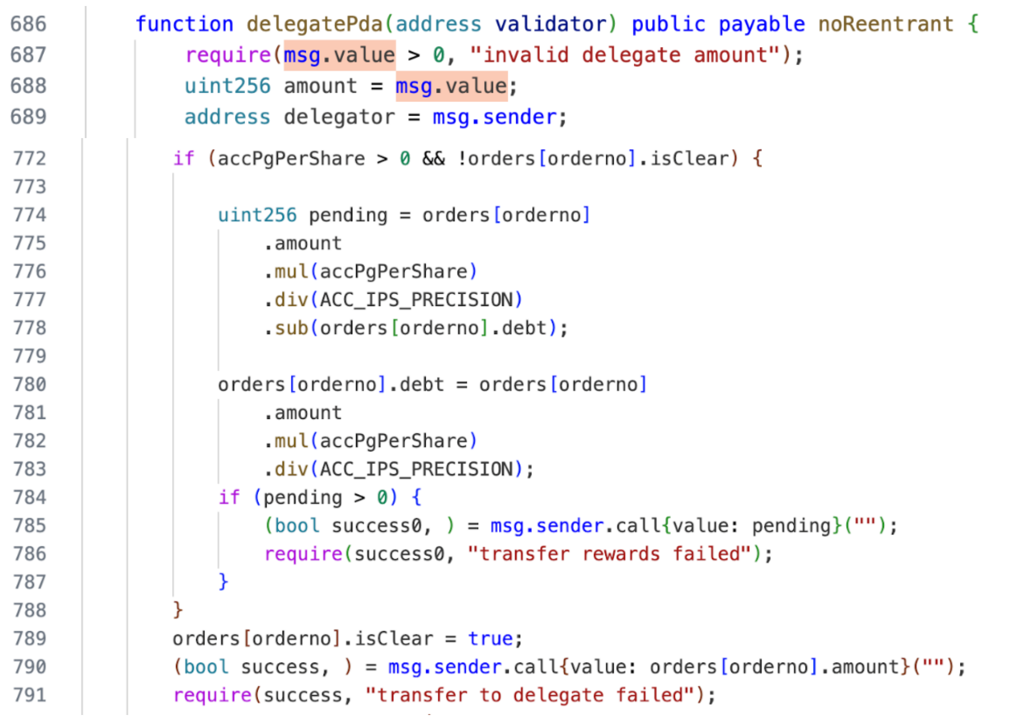
Only the delegatePda method in the contract will receive the user's voting funds, but many rewards in the contract use this fund, which will lead to the following situations:

1. When very few users participate, after the user's funds are sent as rewards, the user will not be able to obtain the funds, and the contract can only have other funds coming in.

2. If the rewards are received in large quantities, the principal of some users in the contract, and in extreme cases, all users, will be exhausted.

It is recommended to calculate the user's voting funds separately to avoid the use of user funds for contract rewards.

Code location:



```
    function delegatePda(address validator) public payable noReentrant {
        require(msg.value > 0, "invalid delegate amount");
        uint256 amount = msg.value;
        address delegator = msg.sender;
```

```
        if (accPgPerShare > 0 && !orders[orderno].isClear) {

            uint256 pending = orders[orderno]
                .amount
                .mul(accPgPerShare)
                .div(ACC_IPS_PRECISION)
                .sub(orders[orderno].debt);

            orders[orderno].debt = orders[orderno]
                .amount
                .mul(accPgPerShare)
                .div(ACC_IPS_PRECISION);
            if (pending > 0) {
                (bool success0, ) = msg.sender.call{value: pending}("");
                require(success0, "transfer rewards failed");
            }
        }
        orders[orderno].isClear = true;
        (bool success, ) = msg.sender.call{value: orders[orderno].amount}("");
        require(success, "transfer to delegate failed");
```

Recommendation

It is recommended to calculate the user's voting funds separately to avoid using user funds for contract rewards.

Status

Ignore.

If the customer replies that the demand is like this, the operator will recharge enough PDA in advance when going online as a reward. If it is insufficient, you can also temporarily replenish it and then withdraw it.

4.2.8 The initErc20Permit() method did not find a specific implementation

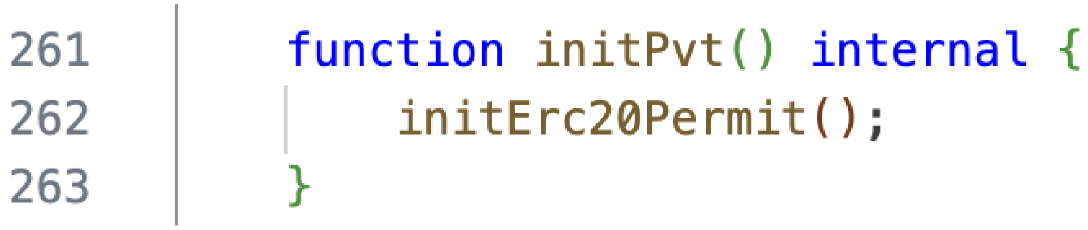
ID	Severity	Location	Status
8	Medium	Validator.sol	Ignore

Description

The initErc20Permit() method did not find a specific implementation. The definition of this method is missing in the current contract or dependent library, and calling it may cause an error.

Check or confirm whether it is necessary to use this method, or ensure that the relevant dependencies are imported correctly.

Code location:



```
    function initPvt() internal {
        initErc20Permit();
    }
```

Recommendation

Check or confirm whether the method is necessary, or ensure that the relevant dependencies are imported correctly.

Status

Ignore.

The client responded by directly changing the local ERC20Permit.sol contract, node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol.

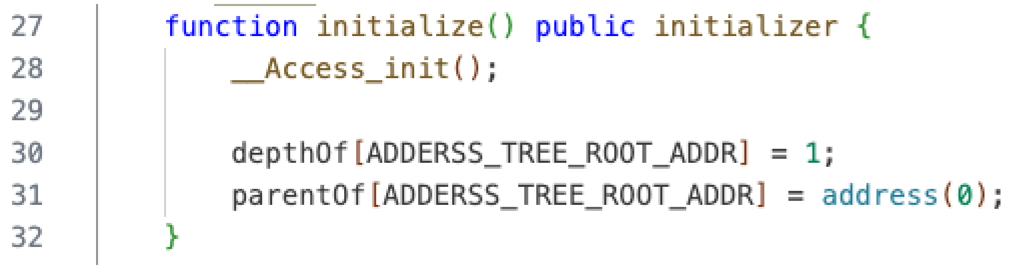
4.2.9 Preemptive Initialization

ID	Severity	Location	Status
09	Low	AddressTree.sol	Ignore

Description

The initializer modifier is used to ensure that initialize is only executed on the initial call to prevent secondary initialization. If the initialize method is preemptively executed during deployment or calling, the root address of rootAddress and related status data may be incorrectly initialized. It is recommended to add an administrator permission or use the _disableInitializers method of the proxy contract.

Code location:



```
    function initialize() public initializer {
        __Access_init();

        depthOf[ADDERSS_TREE_ROOT_ADDR] = 1;
        parentOf[ADDERSS_TREE_ROOT_ADDR] = address(0);
    }
```

Recommendation

It is recommended to add an administrator permission or use the _disableInitializers method of the proxy contract.

Status

Ignore.

As a built-in contract of the system, AddressTree.sol completes the initialize() operation by the node in the first block, and this process is continuous.

4.2.10 You can add a parent to any address

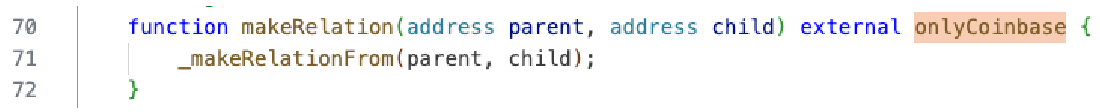
ID	Severity	Location	Status
10	Low	AddressTree.sol	Ignore

Description

The onlyCoinbase modifier is controlled by block.coinbase, and miners or validators can add parents to any address.

It is recommended to add additional access control, introduce multi-signature or specific role permissions, such as OWNER_ROLE, to ensure that only specific accounts can call it.

Code location:



```
function makeRelation(address parent, address child) external onlyCoinbase {
    _makeRelationFrom(parent, child);
}
```

Recommendation

It is recommended to add additional access control, introduce multi-signature or specific role permissions.

Status

Ignore.

The customer explained that the makeRelation method needs to be combined with the node program to complete the addition of the superior. The logic here is bound to the node program. If a Byzantine node attempts to write too many superior relations, the system will judge it as a Byzantine node and will not recognize the data of the block.

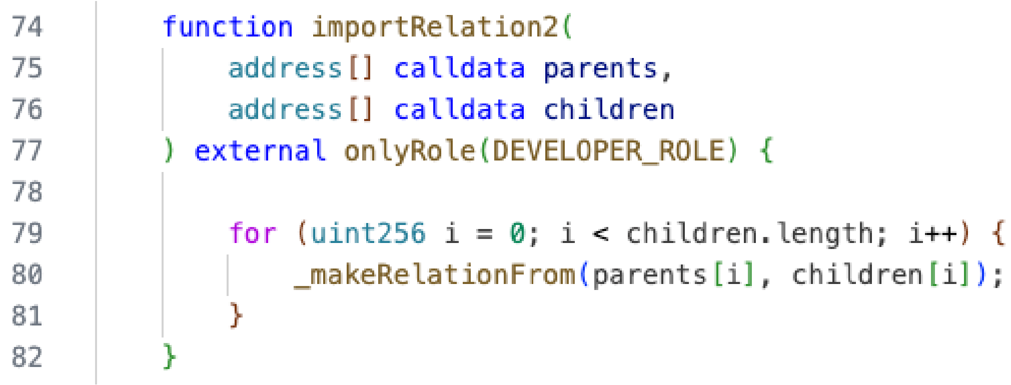
4.2.11 No checks are performed on the length of the parents and children arrays.

ID	Severity	Location	Status
11	Low	AddressTree.sol	Fixed

Description

The onlyNullAddress modifier is not used. If you need to use it, you need to specify the location. If you don't use it, it is recommended to delete it.

Code location:



```
    function importRelation2(
        address[] calldata parents,
        address[] calldata children
    ) external onlyRole(DEVELOPER_ROLE) {

        for (uint256 i = 0; i < children.length; i++) {
            _makeRelationFrom(parents[i], children[i]);
        }
    }
```

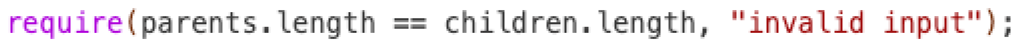
Recommendation

It is recommended to add array length checks to ensure that parents and children are of equal length.

Status

Fixed.

The client has added assertions to check for length equality.



```
require(parents.length == children.length, "invalid input");
```

4.2.12 It is possible to make the addresses of the upper and lower levels the same

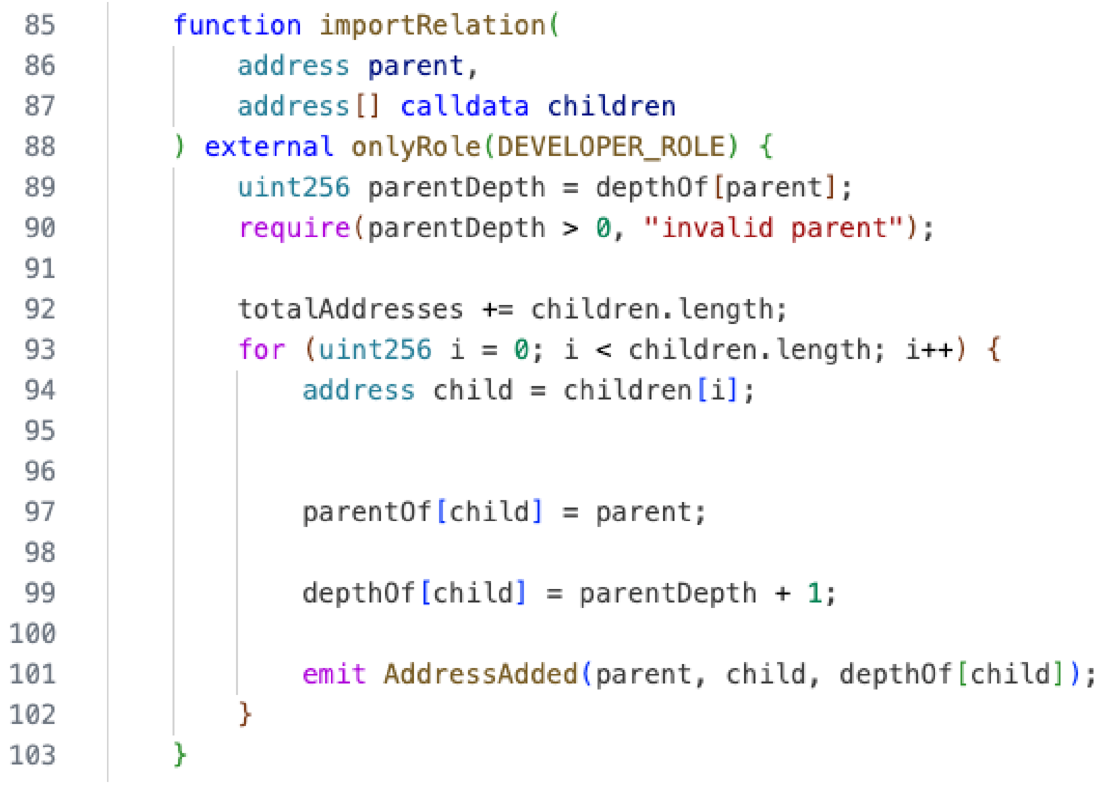
ID	Severity	Location	Status
12	Low	AddressTree.sol	Fixed

Description

It is possible to make the addresses of the parent and child the same.

It is recommended to add a check condition in the importRelation method to prevent the parent and child addresses from being the same.

Code location:



```
    function importRelation(
        address parent,
        address[] calldata children
    ) external onlyRole(DEVELOPER_ROLE) {
        uint256 parentDepth = depthOf[parent];
        require(parentDepth > 0, "invalid parent");

        totalAddresses += children.length;
        for (uint256 i = 0; i < children.length; i++) {
            address child = children[i];


            parentOf[child] = parent;

            depthOf[child] = parentDepth + 1;

            emit AddressAdded(parent, child, depthOf[child]);
        }
    }
```

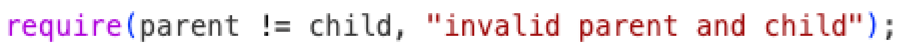
Recommendation

It is recommended to add a check condition in the importRelation method to prevent the parent and child addresses from being the same.

Status

Fixed.

Client Added Added assertion that parent and child cannot be the same.



```
require(parent != child, "invalid parent and child");
```

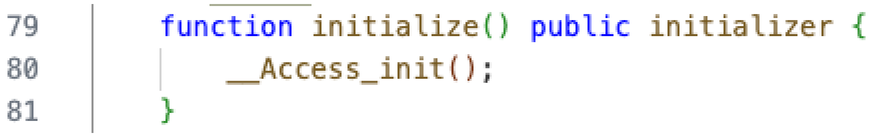
4.2.13 Preemptive Initialization

ID	Severity	Location	Status
13	Low	Farm.sol，SystemDao.sol	Ignore

Description

The initializer modifier is used to ensure that initialize is only executed during the initial call to prevent secondary initialization. If the initialize method is preemptively executed during deployment or calling, the related state data is initialized incorrectly. It is recommended to add an administrator permission or use the _disableInitializers method of the proxy contract.

Code location:



```
function initialize() public initializer {
    __Access_init();
}
```

Recommendation

It is recommended to add an administrator permission or use the _disableInitializers method of the proxy contract.

Status

Ignore.

As a built-in contract in the system, the node completes the initialize() operation in the first block, and this process is continuous.

4.2.14 There is no limit on the value of privileged role settings

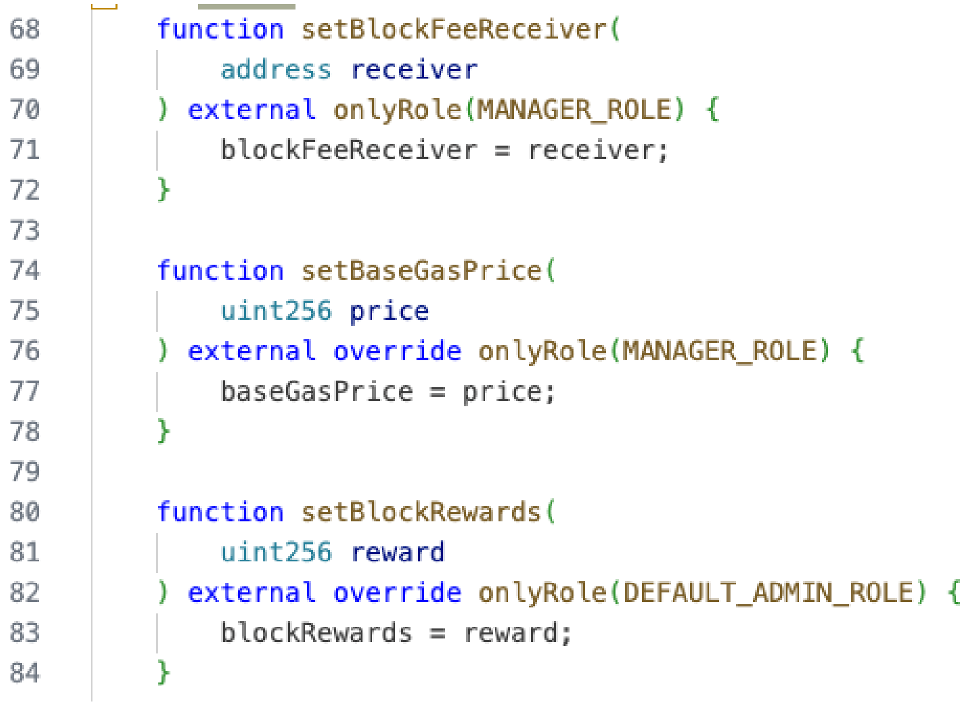
ID	Severity	Location	Status
14	Low	SystemDao.sol	Ignore

Description

The receiver address can be set to zero. There is no limit on the price and reward values, and any value can be set.

It is recommended to add additional checks to ensure that the content set each time is within the contract range. In addition, use multi-signatures to associate privileged roles.

Code location:



```
    function setBlockFeeReceiver(
        address receiver
    ) external onlyRole(MANAGER_ROLE) {
        blockFeeReceiver = receiver;
    }

    function setBaseGasPrice(
        uint256 price
    ) external override onlyRole(MANAGER_ROLE) {
        baseGasPrice = price;
    }

    function setBlockRewards(
        uint256 reward
    ) external override onlyRole(DEFAULT_ADMIN_ROLE) {
        blockRewards = reward;
    }
```

Recommendation

It is recommended to add additional checks to ensure that the content set each time is within the contract range. In addition, use multi-signatures to associate privileged roles.

Status

Ignore.

The customer explained that there may be a demand to set the 0x00 address as the receiving address, or to set the gasPrice to 0. There is no theoretical maximum value provided in the demand.

4.2.15 The same subscriber address is added multiple times

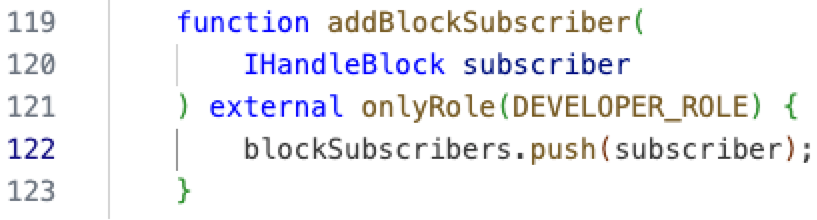
ID	Severity	Location	Status
15	Low	SystemDao.sol	Fixed

Description

The addBlockSubscriber method allows the same subscriber address to be added multiple times.

It is recommended to check whether the address already exists in blockSubscribers before adding a subscriber.

Code location:



```
function addBlockSubscriber(
    IHandleBlock subscriber
) external onlyRole(DEVELOPER_ROLE) {
    blockSubscribers.push(subscriber);
}
```

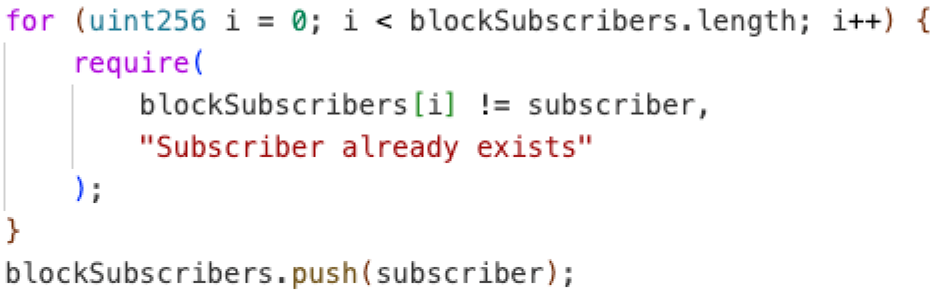
Recommendation

It is recommended to check whether the address already exists in blockSubscribers before adding a subscriber.

Status

Fixed.

The client has added a judgment when adding a subscriber to prevent duplication.



```
for (uint256 i = 0; i < blockSubscribers.length; i++) {
    require(
        blockSubscribers[i] != subscriber,
        "Subscriber already exists"
    );
}
blockSubscribers.push(subscriber);
```

4.2.16 The claimDelegatorRewards and other methods do not add modifiers to prevent reentry

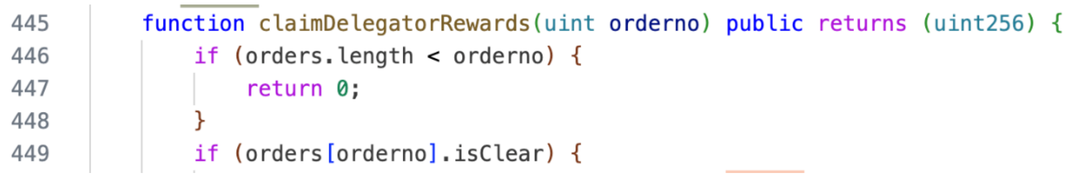
ID	Severity	Location	Status
16	Low	Validator.sol	Ignore

Description

claimDelegatorRewards involves calling external transfer operations, which poses a potential risk of reentrancy attacks.

Add the noReentrant modifier to prevent reentrancy attacks during execution.

Code location:



```
function claimDelegatorRewards(uint orderno) public returns (uint256) {
    if (orders.length < orderno) {
        return 0;
    }
    if (orders[orderno].isClear) {
```

Recommendation

Add the noReentrant modifier to prevent reentrancy attacks during execution.

Status

Ignore.

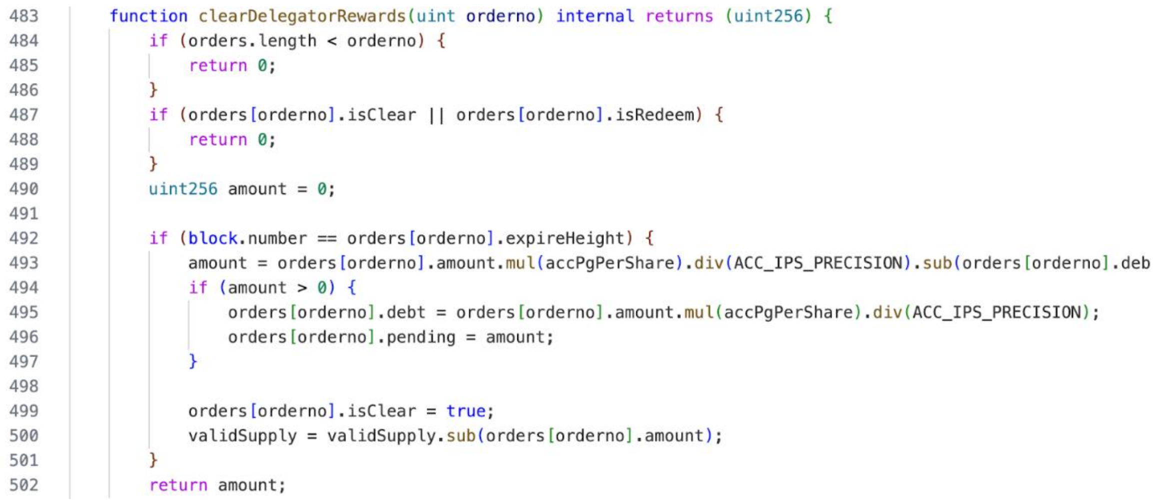
4.2.17 Strict maturity height matching problem

ID	Severity	Location	Status
17	Low	Validator.sol	Ignore

Description

The condition block.number == orders[orderno].expireHeight is too strict and may cause orders to not be cleared in time, especially when the block generation time is unstable.

Code location:



```
function clearDelegatorRewards(uint orderno) internal returns (uint256) {
    if (orders.length < orderno) {
        return 0;
    }
    if (orders[orderno].isClear || orders[orderno].isRedeem) {
        return 0;
    }
    uint256 amount = 0;

    if (block.number == orders[orderno].expireHeight) {
        amount = orders[orderno].amount.mul(accPgPerShare).div(ACC_IPS_PRECISION).sub(orders[orderno].deb
        if (amount > 0) {
            orders[orderno].debt = orders[orderno].amount.mul(accPgPerShare).div(ACC_IPS_PRECISION);
            orders[orderno].pending = amount;
        }

        orders[orderno].isClear = true;
        validSupply = validSupply.sub(orders[orderno].amount);
    }
    return amount;
```

Recommendation

You can consider clearing as long as the expiration block height is exceeded.

Status

Ignore.

The client explained that this method will only be called by mining nodes, and will be called once for each block, so there will be no problem with strict height matching.

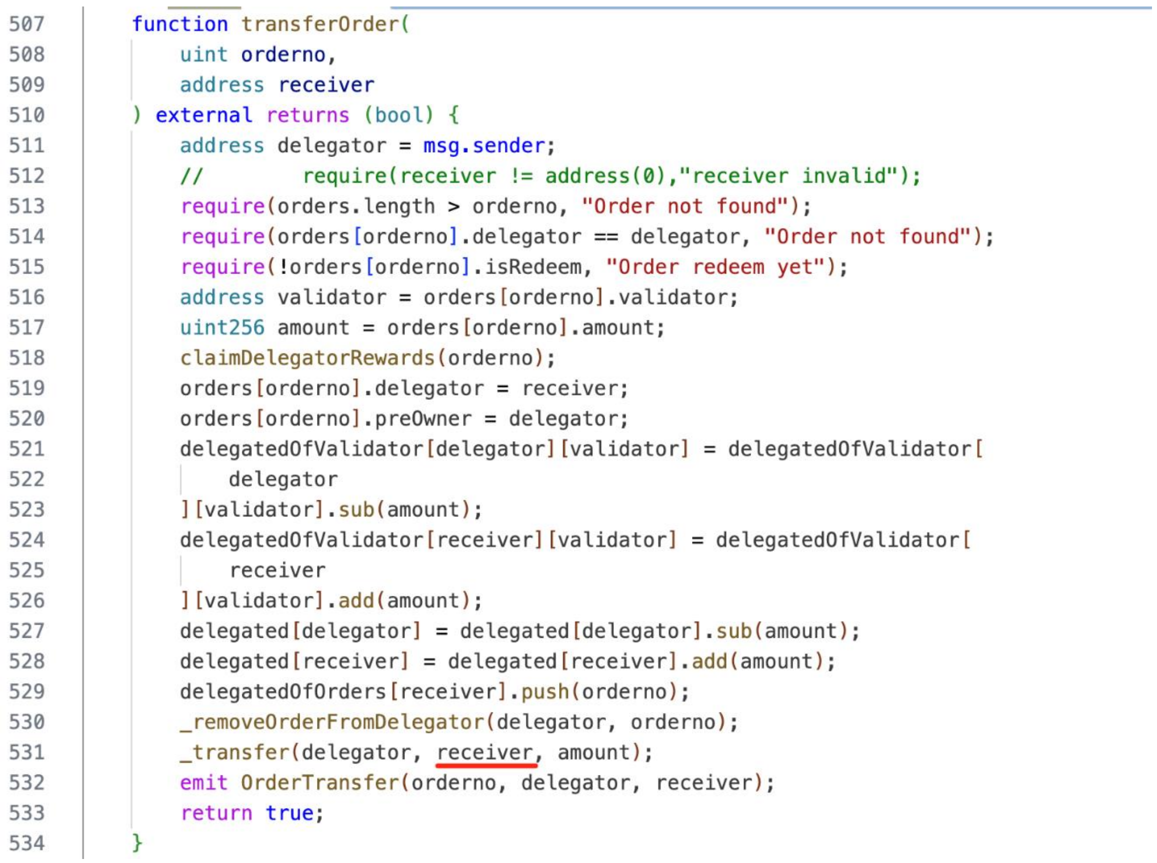
4.2.18 The transferOrder method does not check if the receiver address is the zero address

ID	Severity	Location	Status
18	Low	Validator.sol	Fixed

Description

The transferOrder method does not check whether the receiver address is a zero address. If the receiver address is a zero address, the funds may be locked.

Code location:



```
function transferOrder(
    uint orderno,
    address receiver
) external returns (bool) {
    address delegator = msg.sender;
    //        require(receiver != address(0),"receiver invalid");
    require(orders.length > orderno, "Order not found");
    require(orders[orderno].delegator == delegator, "Order not found");
    require(!orders[orderno].isRedeem, "Order redeem yet");
    address validator = orders[orderno].validator;
    uint256 amount = orders[orderno].amount;
    claimDelegatorRewards(orderno);
    orders[orderno].delegator = receiver;
    orders[orderno].preOwner = delegator;
    delegatedOfValidator[delegator][validator] = delegatedOfValidator[
        delegator
    ][validator].sub(amount);
    delegatedOfValidator[receiver][validator] = delegatedOfValidator[
        receiver
    ][validator].add(amount);
    delegated[delegator] = delegated[delegator].sub(amount);
    delegated[receiver] = delegated[receiver].add(amount);
    delegatedOfOrders[receiver].push(orderno);
    _removeOrderFromDelegator(delegator, orderno);
    _transfer(delegator, receiver, amount);
    emit OrderTransfer(orderno, delegator, receiver);
    return true;
}
```

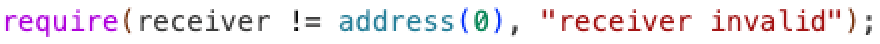
Recommendation

Add a check to ensure that the receiver is a valid address.

Status

Fixed.

The client has added a judgement.



```
require(receiver != address(0), "receiver invalid");
```

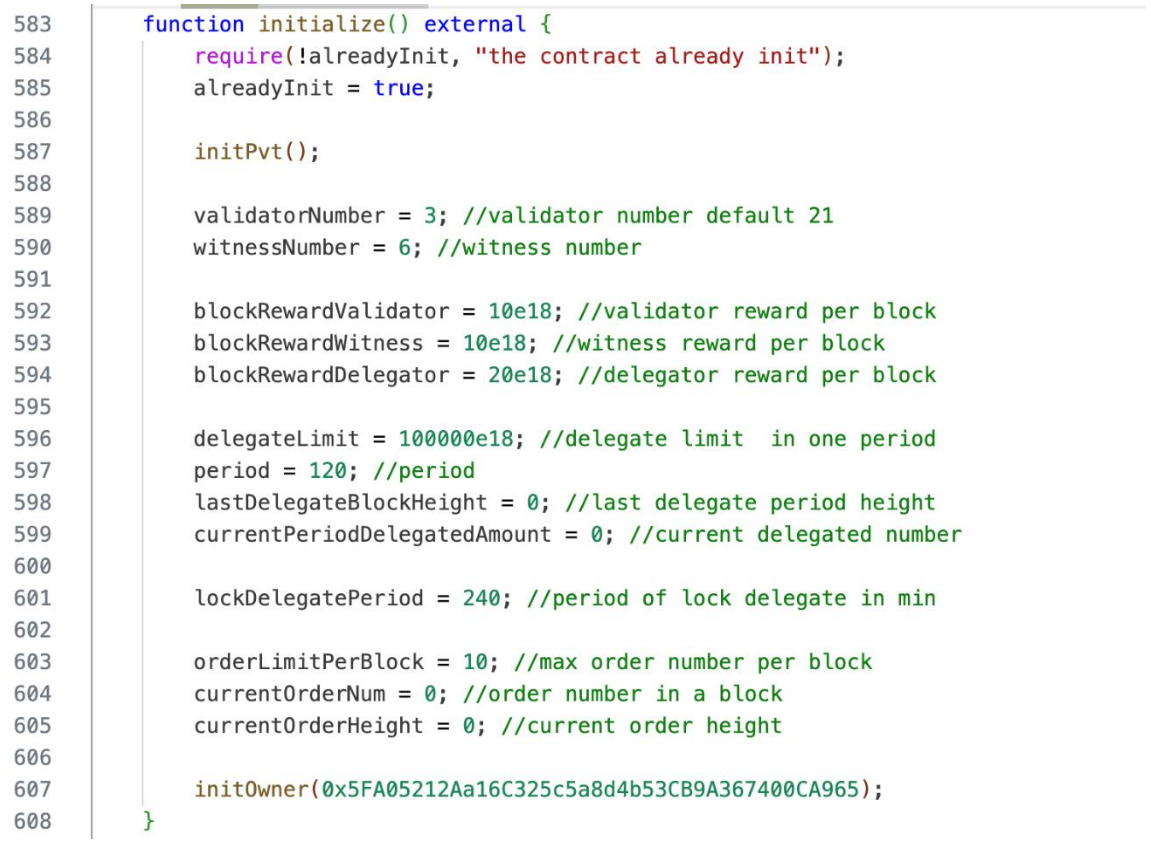
4.2.19 initialize has the risk of preemptive initialization

ID	Severity	Location	Status
19	Low	Validator.sol	Fixed

Description

The initialize method without protection mechanism may be called by any user, which creates the risk of preemptive initialization.

Code location:



```
function initialize() external {
    require(!alreadyInit, "the contract already init");
    alreadyInit = true;

    initPvt();

    validatorNumber = 3; //validator number default 21
    witnessNumber = 6; //witness number

    blockRewardValidator = 10e18; //validator reward per block
    blockRewardWitness = 10e18; //witness reward per block
    blockRewardDelegator = 20e18; //delegator reward per block

    delegateLimit = 100000e18; //delegate limit  in one period
    period = 120; //period
    lastDelegateBlockHeight = 0; //last delegate period height
    currentPeriodDelegatedAmount = 0; //current delegated number

    lockDelegatePeriod = 240; //period of lock delegate in min

    orderLimitPerBlock = 10; //max order number per block
    currentOrderNum = 0; //order number in a block
    currentOrderHeight = 0; //current order height

    initOwner(0x5FA05212Aa16C325c5a8d4b53CB9A367400CA965);
}
```

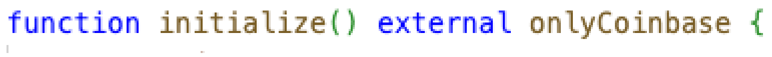
Recommendation

Add calling permission.

Status

Fixed.

The client has added the onlyCoinbase modifier.



```
function initialize() external onlyCoinbase {
```

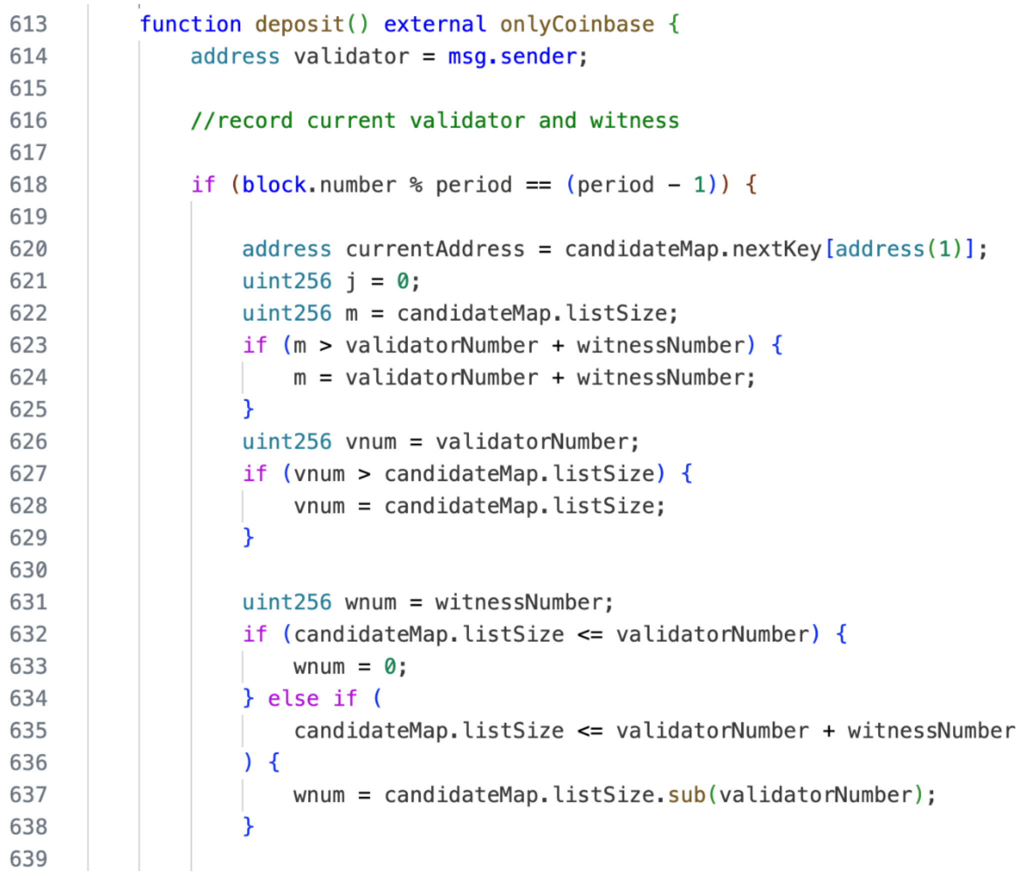
4.2.20 Inflexible time or block height control mechanism

ID	Severity	Location	Status
20	Low	Validator.sol	Ignore

Description

The conditional judgment of if (block.number % period == (period - 1)) depends on the period judgment of the block generation frequency. The current period judgment depends on the fixed block generation frequency. If the frequency is unstable, it may affect the accurate allocation of validators and witnesses.

Code location:



```
function deposit() external onlyCoinbase {
    address validator = msg.sender;

    //record current validator and witness

    if (block.number % period == (period - 1)) {

        address currentAddress = candidateMap.nextKey[address(1)];
        uint256 j = 0;
        uint256 m = candidateMap.listSize;
        if (m > validatorNumber + witnessNumber) {
            m = validatorNumber + witnessNumber;
        }
        uint256 vnum = validatorNumber;
        if (vnum > candidateMap.listSize) {
            vnum = candidateMap.listSize;
        }

        uint256 wnum = witnessNumber;
        if (candidateMap.listSize <= validatorNumber) {
            wnum = 0;
        } else if (
            candidateMap.listSize <= validatorNumber + witnessNumber
        ) {
            wnum = candidateMap.listSize.sub(validatorNumber);
        }

```

Recommendation

It is recommended to use a more flexible time or block height control mechanism to reduce the impact of the block generation interval.

Status

Ignore.

The customer stated this is the requirement.

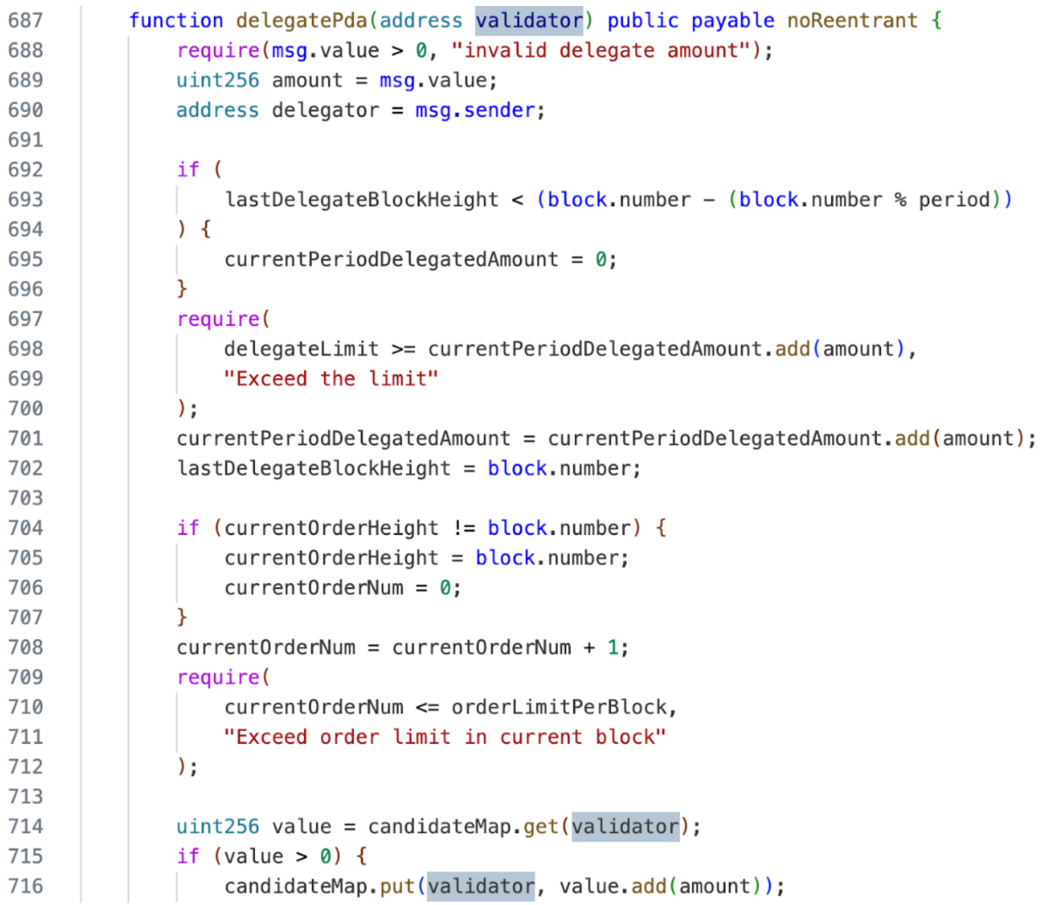
4.2.21 The delegatePda method lacks validity check for the validator

ID	Severity	Location	Status
21	Low	Validator.sol	Fixed

Description

The delegatePda method lacks a validity check on the validator, and does not verify whether the passed-in validator address is a valid validator.

Code location:



```
function delegatePda(address validator) public payable noReentrant {
    require(msg.value > 0, "invalid delegate amount");
    uint256 amount = msg.value;
    address delegator = msg.sender;

    if (
        lastDelegateBlockHeight < (block.number - (block.number % period))
    ) {
        currentPeriodDelegatedAmount = 0;
    }
    require(
        delegateLimit >= currentPeriodDelegatedAmount.add(amount),
        "Exceed the limit"
    );
    currentPeriodDelegatedAmount = currentPeriodDelegatedAmount.add(amount);
    lastDelegateBlockHeight = block.number;

    if (currentOrderHeight != block.number) {
        currentOrderHeight = block.number;
        currentOrderNum = 0;
    }
    currentOrderNum = currentOrderNum + 1;
    require(
        currentOrderNum <= orderLimitPerBlock,
        "Exceed order limit in current block"
    );

    uint256 value = candidateMap.get(validator);
    if (value > 0) {
        candidateMap.put(validator, value.add(amount));
```

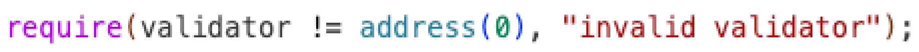
Recommendation

Add a require statement to ensure that the validator is a valid validator address.

Status

Fixed.

Added non-zero address check.



```
require(validator != address(0), "invalid validator");
```

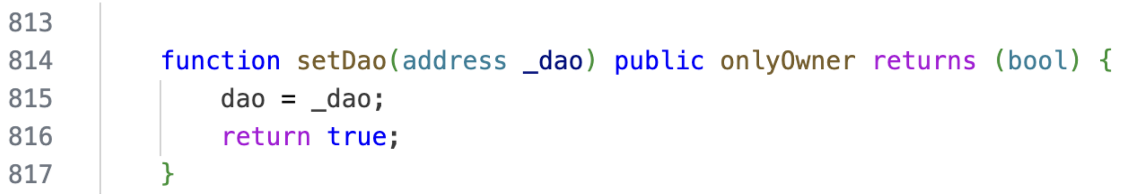
4.2.22 No check for zero address in setDao method

ID	Severity	Location	Status
22	Low	Validator.sol	Fixed

Description

There is no check for zero address in the setDao method, and setting dao to zero address may cause loss of functionality.

Code location:



```

    function setDao(address _dao) public onlyOwner returns (bool) {
        dao = _dao;
        return true;
    }
```

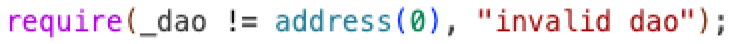
Recommendation

It is recommended to add a check in setDao.

Status

Fixed.

Added non-zero address check.



```
require(_dao != address(0), "invalid dao");
```

4.2.23 Calculation accuracy issues

ID	Severity	Location	Status
23	Low	tokens/ZEUSLock.sol	Ignore

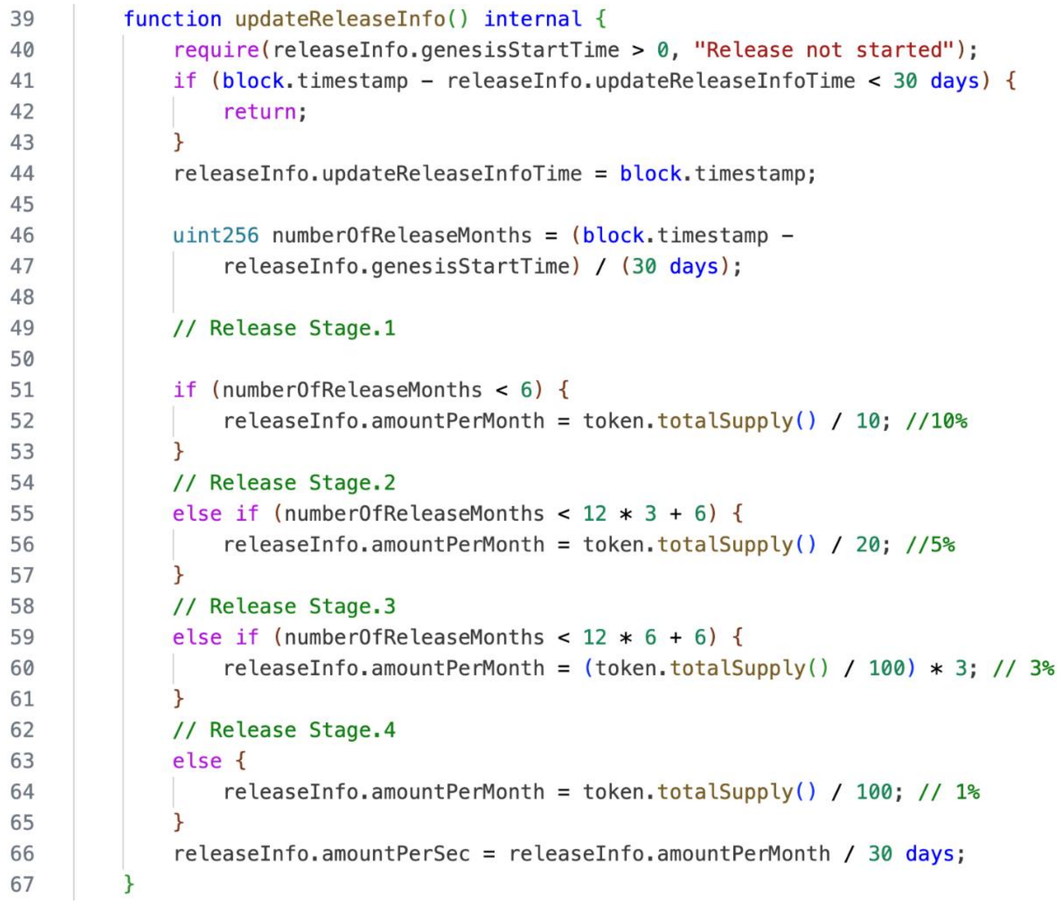
Description

When processing the calculation of amountPerSec, there is a precision problem, as follows:

releaseInfo.amountPerSec = releaseInfo.amountPerMonth / 30 days;

The monthly release amount is divided by 30 days to calculate the release amount per second. If the amountPerMonth value is small, the calculation result may be rounded to zero. If you want to maintain higher precision, you can use additional decimal precision (such as 10**18 units) to save and calculate to reduce the loss of precision.

Code location:



```
function updateReleaseInfo() internal {
    require(releaseInfo.genesisStartTime > 0, "Release not started");
    if (block.timestamp - releaseInfo.updateReleaseInfoTime < 30 days) {
        return;
    }
    releaseInfo.updateReleaseInfoTime = block.timestamp;

    uint256 numberOfReleaseMonths = (block.timestamp -
        releaseInfo.genesisStartTime) / (30 days);

    // Release Stage.1

    if (numberOfReleaseMonths < 6) {
        releaseInfo.amountPerMonth = token.totalSupply() / 10; //10%
    }
    // Release Stage.2
    else if (numberOfReleaseMonths < 12 * 3 + 6) {
        releaseInfo.amountPerMonth = token.totalSupply() / 20; //5%
    }
    // Release Stage.3
    else if (numberOfReleaseMonths < 12 * 6 + 6) {
        releaseInfo.amountPerMonth = (token.totalSupply() / 100) * 3; // 3%
    }
    // Release Stage.4
    else {
        releaseInfo.amountPerMonth = token.totalSupply() / 100; // 1%
    }
    releaseInfo.amountPerSec = releaseInfo.amountPerMonth / 30 days;
}
```

Recommendation

Additional decimal precision (e.g. units of 10**18) can be used for storage and calculations to reduce the loss of precision.

Status

Ignore.

The customer replied that the releaseInfo.amountPerMonth value will be guaranteed to be accurate enough when it is first set, and the value will only increase, not decrease. We will pay attention to the value set for the first time being greater than the number of seconds corresponding to 30days.

4.2.24 Unrecognized chainID causes accountPowerLimitSetter to always be empty

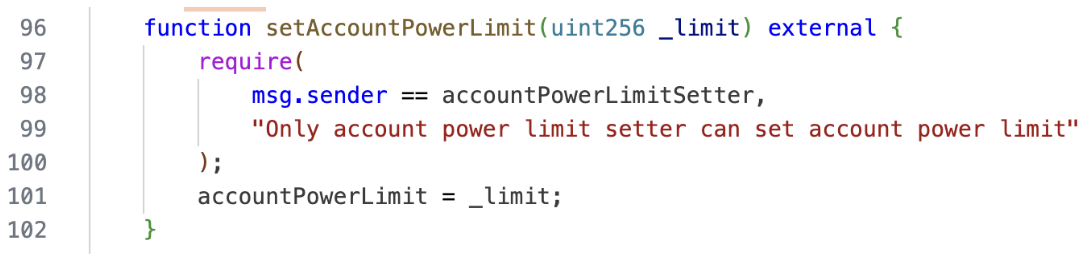
ID	Severity	Location	Status
24	Low	tokens/ZEUSToken.sol	Ignore

Description

If deployed on an unidentified chain (i.e. not in the chainID specified in the code), accountPowerLimitSetter remains an empty address and accountPowerLimit cannot be changed.

It is recommended to add a fallback method that allows the contract owner to set accountPowerLimitSetter to ensure that accountPowerLimitSetter can be initialized when deployed on different chains.

Code location:



```
function setAccountPowerLimit(uint256 _limit) external {
    require(
        msg.sender == accountPowerLimitSetter,
        "Only account power limit setter can set account power limit"
    );
    accountPowerLimit = _limit;
}
```

Recommendation

It is recommended to add a fallback method that allows the contract owner to set accountPowerLimitSetter to ensure that accountPowerLimitSetter can be initialized when deployed on different chains.

Status

Ignore.

4.2.25 The calculation of totalPower - originPower may be wrong

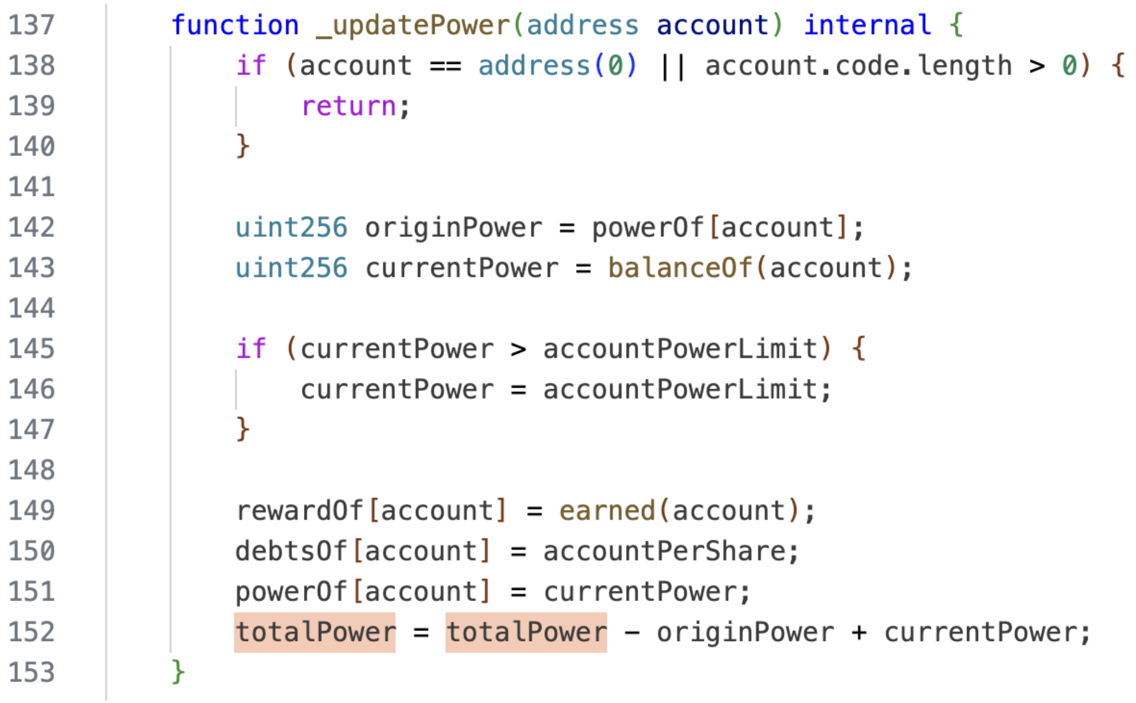
ID	Severity	Location	Status
25	Low	tokens/ZEUSToken.sol	Ignore

Description

totalPower is initialized to zero, and if originPower is positive and not updated, totalPower calculation will cause an error.

When used, check the current originPower and totalPower values to ensure that no unexpected subtraction underflow is caused.

Code location:



```
    function _updatePower(address account) internal {
        if (account == address(0) || account.code.length > 0) {
            return;
        }

        uint256 originPower = powerOf[account];
        uint256 currentPower = balanceOf(account);

        if (currentPower > accountPowerLimit) {
            currentPower = accountPowerLimit;
        }

        rewardOf[account] = earned(account);
        debtsOf[account] = accountPerShare;
        powerOf[account] = currentPower;
        totalPower = totalPower - originPower + currentPower;
    }
```

Recommendation

When used, check the current originPower and totalPower values to ensure that no unexpected subtraction underflow is caused.

Status

Ignore.

4.2.26 Privileged role management

ID	Severity	Location	Status
26	Low	tokens/ZEUSToken.sol	Ignore

Description

Currently, multiple variable settings in the project are controlled by privileged roles. It is recommended that privileged roles use multi-signatures for management.

Recommendation

It is recommended that privileged roles be managed using multi-signatures.

Status

Ignore.

Regarding the issue of centralization, after the next fork, it will be transferred to the DAO governance platform to achieve true decentralization.

4.2.27 There is no suspension mechanism in the project contract

ID	Severity	Location	Status
27	Low	tokens/ZEUSToken.sol	Ignore

Description

Currently, there are multiple deposit and withdrawal methods in the project. If unexpected situations occur, such as hacker attacks, node defects, etc., it may lead to capital loss.It is recommended to add a suspension mechanism to suspend the project in time after unexpected situations occur to avoid capital loss.

Recommendation

It is recommended to add a suspension mechanism to suspend the project in time after unexpected situations occur to avoid capital loss.

Status

Ignore.

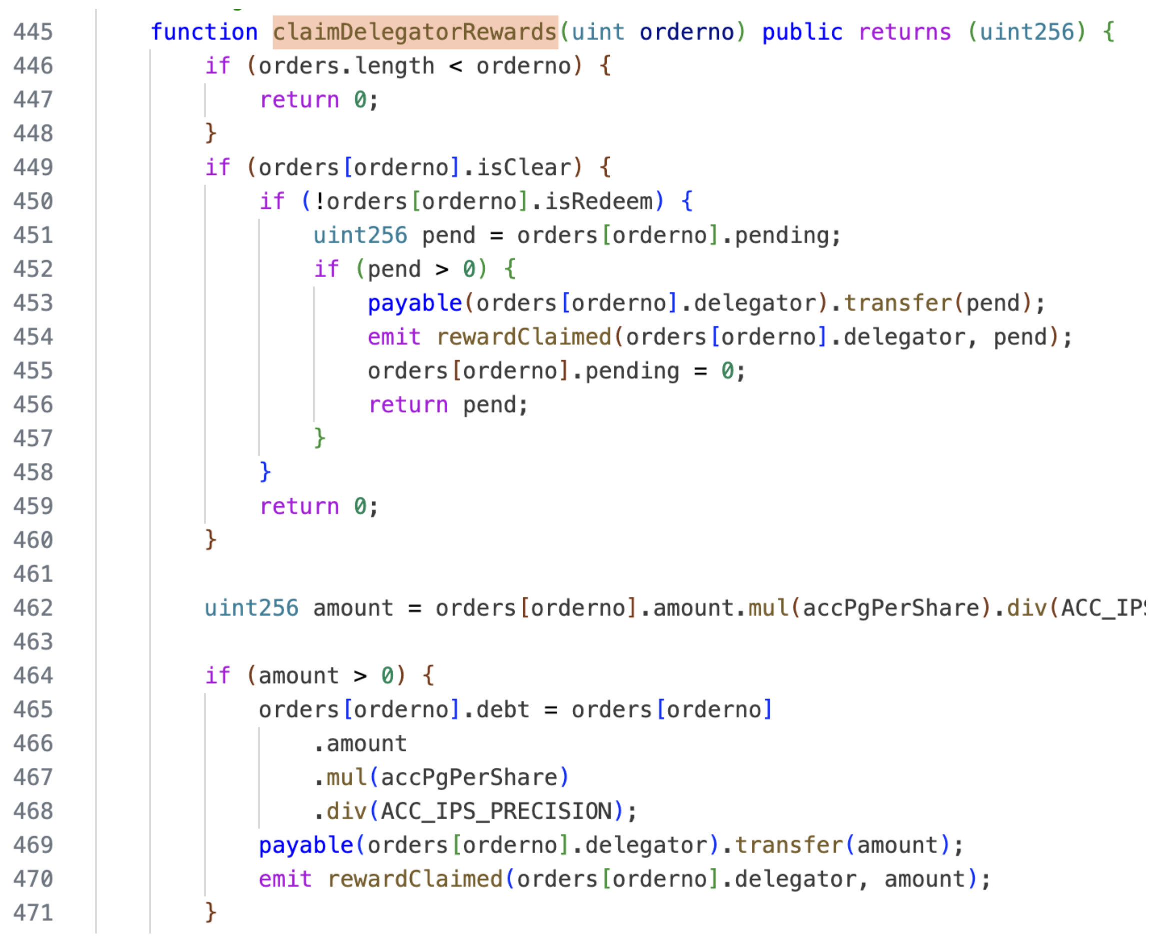
4.2.28 The claimDelegatorRewards method can be called by any user

ID	Severity	Location	Status
28	Low	Validator.sol	Fixed

Description

claimDelegatorRewards does not verify the identity of the caller, and any user can call claimDelegatorRewards and claim the reward for the specified order. This means that malicious users can help others claim rewards by calling this function and passing in other people's order numbers, causing the reward balance to be cleared..

Code location:



```
    function claimDelegatorRewards(uint orderno) public returns (uint256) {
        if (orders.length < orderno) {
            return 0;
        }
        if (orders[orderno].isClear) {
            if (!orders[orderno].isRedeem) {
                uint256 pend = orders[orderno].pending;
                if (pend > 0) {
                    payable(orders[orderno].delegator).transfer(pend);
                    emit rewardClaimed(orders[orderno].delegator, pend);
                    orders[orderno].pending = 0;
                    return pend;
                }
            }
            return 0;
        }

        uint256 amount = orders[orderno].amount.mul(accPgPerShare).div(ACC_IP

        if (amount > 0) {
            orders[orderno].debt = orders[orderno]
                .amount
                .mul(accPgPerShare)
                .div(ACC_IPS_PRECISION);
            payable(orders[orderno].delegator).transfer(amount);
            emit rewardClaimed(orders[orderno].delegator, amount);
        }
```

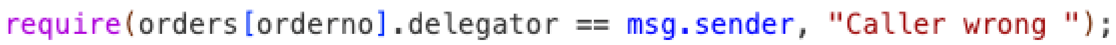
Recommendation

It is recommended to add a permission verification mechanism to ensure that only orders[orderno].delegator == msg.sender can claim rewards.

Status

Fixed.

Added restrictions on personal operations.



```
require(orders[orderno].delegator == msg.sender, "Caller wrong ");
```

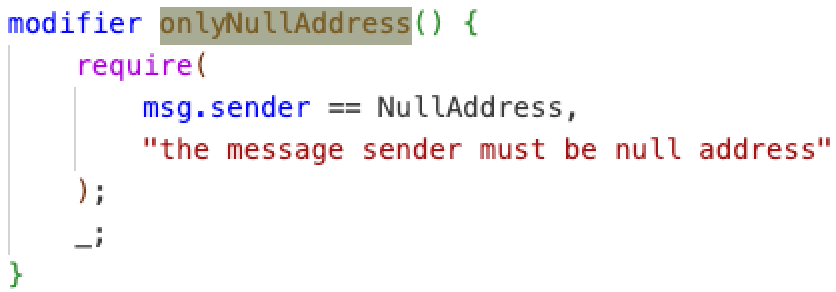
4.2.29 Unused onlyNullAddress modifier

ID	Severity	Location	Status
29	Informational	Access.sol	Fixed

Description

The onlyNullAddress modifier is not used. If you need to use it, you need to specify the location. If you don't use it, it is recommended to delete it.

Code location:



```
modifier onlyNullAddress() {
    require(
        msg.sender == NullAddress,
        "the message sender must be null address"
    );
    _;
}
```

Recommendation

It is recommended to delete unused codes.

Status

Fixed.

The client has removed the unused onlyNullAddress modifier.

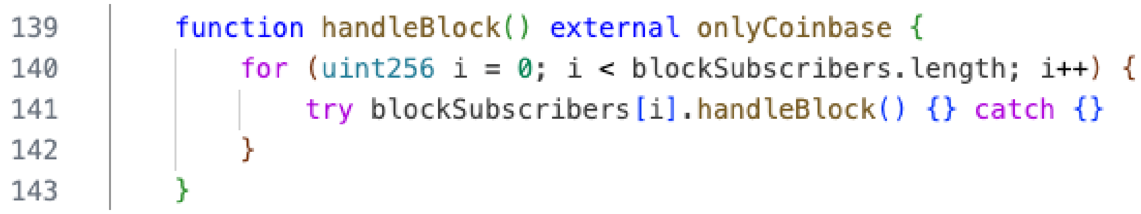
4.2.30 Lack of logging and error handling in handleBlock method

ID	Severity	Location	Status
30	Informational	SystemDao.sol	Ignore

Description

The handleBlock method currently only uses try-catch to catch exceptions. There is no log to record failed calls, which may make debugging and troubleshooting more difficult.

Code location:



```
function handleBlock() external onlyCoinbase {
    for (uint256 i = 0; i < blockSubscribers.length; i++) {
        try blockSubscribers[i].handleBlock() {} catch {}
    }
}
```

Recommendation

Add logging to the catch block.

Status

Ignore.

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.