Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.05.19, the SlowMist security team received the OURO Protocol team's security audit application for OURO Protocol, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

OURO Protocol is a DeFi ecosystem built around a dual-token model featuring OUSD (a stablecoin) and CBN (a reward token) that operates through an innovative epoch-based reward distribution system. Users can mint OUSD using USDT and participate in the protocol's economy where every OUSD transfer incurs a 10% tax that gets burned and converted into CBN tokens, which are then allocated across time-based epochs to generate compounding rewards over time. The protocol enables users to earn passive income through CBN token accumulation, with CBN holdings granting users the right to actively apply for progressive airdrops, benefit from a referral system that provides additional CBN returns, and access premium features through VIP memberships, while also offering specialized minting services where users can directly burn OUSD to acquire CBN tokens for enhanced reward potential. The entire system is designed to create a sustainable token economy where user activity generates value

through the tax mechanism, rewards long-term participation through compound interest-like returns, and provides multiple revenue streams including user-initiated airdrop claims, referral commissions, and staking-like rewards, all managed through an automated epoch progression system that balances token supply, demand, and reward distribution over time.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N2	Unbounded Loop Gas Exhaustion in Epoch History Traversal Functions	Denial of Service Vulnerability	Medium	Fixed
N3	Inconsistent Commission Calculation in OUROMinter Contract	Design Logic Audit	Low	Fixed
N4	Incorrect Growth Calculation for Multi-Stage Epochs	Design Logic Audit	Low	Fixed
N5	Missing Explicit Return Statement in powDecimal Function	Others	Low	Fixed
N6	Array Input Validation Issues in Minter Information Query Function	Design Logic Audit	Low	Fixed
N7	Redundant require statement followed by conditional block	Design Logic Audit	Suggestion	Fixed
N8	Overcomplicated multi-region calculation	Design Logic Audit	Suggestion	Fixed
N9	Function visibility optimization	Gas Optimization Audit	Suggestion	Fixed

NO	Title	Category	Level	Status
N10	Missing the event record	Others	Suggestion	Fixed
N11	Preemptive Initialization	Race Conditions Vulnerability	Suggestion	Acknowledged
N12	Redundant State Variable Assignment in Initialize Function	Scoping and Declarations Audit	Suggestion	Fixed
N13	Missing zero address check	Others	Suggestion	Fixed
N14	Integer Division Precision Loss in Multiple Functions	Arithmetic Accuracy Deviation Vulnerability	Information	Acknowledged
N15	Airdrop Level Hard Cap May Cause User Exclusion in Edge Cases	Design Logic Audit	Information	Acknowledged
N16	Epoch Advancement Requires User Activity Triggering	Design Logic Audit	Information	Acknowledged
N17	Token Compatibility Reminder	Others	Information	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/ouro-org/contract

commit: 9a0623d834133b2de2ec91d9ae75cf8aafcaa00b

Fixed Version:

https://github.com/ouro-org/contract

commit: 8b92f353ac538d49dba8ab56d931689740aefe4b

```
./contracts
├── CBN.sol
├── OURODataLib.sol
├── OUROGenesisVip.sol
├── OUROMintRouter.sol
├── OUROMinter.sol
├── OURONormalVip.sol
├── OUROProtocolProxy.sol
├── OUROProtocolUpgradeable.sol
├── OUROVipManager.sol
├── OUSD.sol
├── interface
│   ├── ICBN.sol
│   ├── IOUROMintRouter.sol
│   ├── IOUROMinter.sol
│   ├── IOUROProtocol.sol
│   ├── IOUROVipManager.sol
│   └── IOUSD.sol
```

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

OUSD			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 Ownable
decimals	Public	-	-
setManager	External	Can Modify State	onlyOwner
mint	External	Can Modify State	onlyManager
burn	Public	Can Modify State	onlyManager
_update	Internal	Can Modify State	-

CBN			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 Ownable
decimals	Public	-	-
setManager	External	Can Modify State	onlyOwner
mint	External	Can Modify State	onlyManager
_update	Internal	Can Modify State	-

OUROGenesisVip			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC721 Ownable
mint	Public	Can Modify State	onlyVipManager
setVipManager	External	Can Modify State	onlyOwner
_update	Internal	Can Modify State	-

OUROMinter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
setManager	External	Can Modify State	onlyOwner
addMinter	External	Can Modify State	onlyOwner nonReentrant
updateMinterName	External	Can Modify State	onlyOwner nonReentrant
removeMinter	External	Can Modify State	onlyOwner nonReentrant
getMintersInfoAndAllowance	External	-	-
mintByMinter	External	Can Modify State	onlyManager whenNotPaused

OUROMinter			
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner

OUROMintRouter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
setManager	External	Can Modify State	onlyOwner
sendCommissionToMinter	External	Can Modify State	onlyManager

OURONormalVip			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC721 Ownable
setVipManager	External	Can Modify State	onlyOwner
mint	Public	Can Modify State	onlyVipManager
_update	Internal	Can Modify State	-

OUROProtocolProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC1967Proxy

OUROVipManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
setManager	External	Can Modify State	onlyOwner

OUROVipManager			
joinNormalMemberShip	External	Can Modify State	onlyManager
joinGenesisMemberShip	External	Can Modify State	onlyManager
isNormalVip	Public	-	-
isGenesisVip	Public	-	-

OUROProtocolUpgradeable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
_authorizeUpgrade	Internal	Can Modify State	onlyOwner
deposit	Public	Can Modify State	nonReentrant whenNotPaused
_handleReferrer	Private	Can Modify State	-
redeem	External	Can Modify State	nonReentrant whenNotPaused
handleTransferTax	External	Can Modify State	nonReentrant
_calculateReferrerRebate	Private	Can Modify State	-
_recordUserTaxBurned	Private	Can Modify State	-
_appendCBNToPendingEpoch	Private	Can Modify State	-
_generateNewEpoch	Private	Can Modify State	-
_checkAndAdvanceCurrentEpoch	Private	Can Modify State	-
_calculateEpochDelayRate	Private	-	-

OUROProtocolUpgradeable			
getEpochInfo	Public	-	-
userCumulativeApplyableRewardsAmount	Public	-	-
getUserCBNAppliedInfo	Public	-	-
applyAirdrop	External	Can Modify State	nonReentrant whenNotPaused
currentEpochPendingAmount	Public	-	-
calculateClaimableAmount	Public	-	-
claimAirdrop	External	Can Modify State	nonReentrant whenNotPaused
userNextApplyAmount	Public	-	-
calculateApplyAmount	Public	-	-
userCanBeReferrer	Public	-	-
getUserValidCBN	External	-	-
setReferrerQualification	External	Can Modify State	onlyOwner
joinNormalVip	External	Can Modify State	whenNotPaused
joinGenesisVip	External	Can Modify State	whenNotPaused
mintCBNByMinter	External	Can Modify State	-
withdrawRedeemTax	External	Can Modify State	onlyOwner whenNotPaused
withdrawVipIncome	External	Can Modify State	onlyOwner whenNotPaused
addTransferBurnExemptSenderAddress	External	Can Modify State	onlyOwner
removeTransferBurnExemptSenderAddress	External	Can Modify State	onlyOwner
pause	External	Can Modify State	onlyOwner

OUROProtocolUpgradeable			
unpause	External	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the OUROProtocolUpgradeable contract, the owner role can set the referrer address and the transferBurnExemptSenders address arbitrarily.

Code location:

contracts/OUROProtocolUpgradeable.sol#L670-L675, L722-L728

```
    function setReferrerQualification(address referrer) external onlyOwner {
        require(referrer != address(0), "Referrer cannot be the zero address");
        // require(userInfos[referrer].referrer == address(0), "Referrer already
set");
        userInfos[referrer].isReferrer = true;
        emit ReferrerQualificationSet(referrer);
    }

    function addTransferBurnExemptSenderAddress(address addr) external onlyOwner {
        transferBurnExemptSenders[addr] = true;
    }

    function removeTransferBurnExemptSenderAddress(address addr) external onlyOwner {
        transferBurnExemptSenders[addr] = false;
    }
```

2.In the OUROMinter contract, the owner role can add or remove minters arbitrarily, and set any dailyAllowance daliy mint amount.

contracts/OUROMinter.sol#L73-84, L92-L97

```
    function addMinter(address minter, string memory name, uint256 dailyAllowance)
external onlyOwner nonReentrant {
        ...
    }
```

```
    function removeMinter(address minter) external onlyOwner nonReentrant {
        ...
     }
```

3.The UUPSUpgradeable owner relevant authority can upgrade the contract, leading to the risk of over-privileged in this role.

Code location:

contracts/OUROProtocolUpgradeable.sol#L202

```
    function _authorizeUpgrade(address newImplementation) internal override onlyOwner
{}
```

Solution

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. And the authority involving user funds should be managed by the community, and the EOA address can manage the authority involving emergency contract suspension. This ensures both a quick response to threats and the safety of user funds. It's also recommended to set a more reasonable initial minRewardAmount to prevent dust transactions that waste gas and blockchain resources.

Status

Acknowledged

[N2] [Medium] Unbounded Loop Gas Exhaustion in Epoch History Traversal Functions

Category: Denial of Service Vulnerability

Content

The OURO protocol contains functions that iterate through users' complete epoch history without bounds, creating escalating gas consumption as participation history grows. These functions loop from ousdBurnedFirstEpoch to currentEpochNumber, causing legitimate long-term users to potentially face transaction failures or prohibitively expensive gas costs when accessing their rewards.

Malicious actors can exploit this by creating accounts with artificially long epoch histories through minimal transactions (cost: ~1 OUSD per account), then triggering these functions to cause denial-of-service conditions. The

attack preparation cost is minimal, but the impact escalates significantly as the system matures.

The primary concern is the applyAirdrop() function, which has significantly higher gas consumption per iteration due to complex calculations and storage operations, making it vulnerable within a reasonable timeframe as the protocol matures. This represents a genuine near-term risk to core protocol functionality that requires immediate attention.

The view function userCumulativeApplyableRewardsAmount() has similar structure but considerably lower gas consumption per iteration, requiring substantially longer protocol operation before becoming problematic, making it primarily a theoretical concern for long-term protocol sustainability.

Code location:

contracts/OUROProtocolUpgradeable.sol#L510-L567

```
    function applyAirdrop() external nonReentrant whenNotPaused {
        ...
        uint256 totalRewards = 0;
        for(uint32 epoch = userInfos[msg.sender].ousdBurnedFirstEpoch; epoch <
currentEpochNumber; epoch++) {
            ...
        }
        ...
    }
    function userCumulativeApplyableRewardsAmount() public view returns (uint256) {
        uint256 totalRewards = 0;
        for(uint32 epoch = userInfos[msg.sender].ousdBurnedFirstEpoch; epoch <
currentEpochNumber; epoch++) {
            ...
        }
        return totalRewards;
    }
```

Solution

It's recommended to add loop boundaries to prevent gas exhaustion.

Status

Fixed; The repair code may still have DOS situations in extreme cases. It is recommended that the project party strictly follow suggestApplyEpoch on the front end to take risk mitigation measures to help most ordinary users avoid triggering Gas problems.

[N3] [Low] Inconsistent Commission Calculation in OUROMinter Contract

Category: Design Logic Audit

Content

The mintByMinter function in the OUROMinter contract contains a critical inconsistency in commission calculation logic. The function calculates commission amounts using two different formulas, resulting in a 10x discrepancy between what users are charged and what the system expects to transfer. The function first calculates ousdCommissionAmount as a percentage of the CBN amount (cbnAmount * 1%), which is used to determine the total amount the user must pay. However, it later calculates commissionAmount as a percentage of the OUSD burn amount (ousdAmount * 1% = cbnAmount * 10 * 1%), which is returned to the main contract for the actual transfer. Since ousdAmount is 10 times larger than cbnAmount (due to OUSD_TO_CBN_RATE = 10), the returned commission is 10 times larger than what the user was initially charged.

Code location:

contracts/OUROMinter.sol#L145-L165

```
    function mintByMinter(address user, address minter, uint256 cbnAmount) external
onlyManager whenNotPaused returns (uint256, uint256) {
        require(cbnAmount > 0, "Amount must be greater than 0");
        require(minterWhiteList[minter] == true, "Minter does not exist");
        require(block.timestamp < minterInfo[minter].activeTime +
MINTER_ACTIVE_PERIOD, "Minter is not active");
        uint256 currentDay = block.timestamp / 1 days;
        uint256 ousdAmount = cbnAmount * OUSD_TO_CBN_RATE;
        uint256 ousdCommissionAmount = cbnAmount * minterCommissionRate / RATE_BASE;
        uint256 totalAmount = ousdAmount + ousdCommissionAmount;
        ...
        minterDailyMinted[minter][currentDay] += ousdAmount;
        minterInfo[minter].totalMinted += totalAmount;

        uint256 commissionAmount = ousdAmount * minterCommissionRate / RATE_BASE;
        emit MintByMinter(user, minter, ousdAmount, commissionAmount);
        return (ousdAmount, commissionAmount);
    }
```

Solution

It's recommended that the commission calculation be made consistent by both calculations.

Status

Fixed

[N4] [Low] Incorrect Growth Calculation for Multi-Stage Epochs

Category: Design Logic Audit

Content

In the OURODataLib contract, the epochCBNVolume function incorrectly calculates growth rates when spanning multiple epochs. For n > 200, the code only applies 99 growth iterations for the second stage (periods 101-200) when it should apply 100 iterations. This results in a slightly lower volume calculation than intended.

Code location:

contracts/OURODataLib.sol#L159-L181

```
function epochCBNVolume(uint256 n) internal pure returns (uint256) {
     require(n > 0, "n must be greater than 0");
     if (n == 1) return 1000 * 1e6;

     uint256 base = 1000 * 1e6;
     if (n <= 100) {
         base = mulDecimal(base, powDecimal(103e4, n - 1)); // 1.03^(n-1)
     } else if (n <= 200) {
         base = mulDecimal(base, powDecimal(103e4, 99)); // 1.03^99
         base = mulDecimal(base, powDecimal(102e4, n - 100)); // 1.02^(n-100)
     } else if (n <= 300) {
         base = mulDecimal(base, powDecimal(103e4, 99)); // 1.03^99
         base = mulDecimal(base, powDecimal(102e4, 99)); // 1.02^99
         base = mulDecimal(base, powDecimal(101e4, n - 200)); // 1.01^(n-200)
     } else {
         base = mulDecimal(base, powDecimal(103e4, 99)); // 1.03^99
         base = mulDecimal(base, powDecimal(102e4, 99)); // 1.02^99
         base = mulDecimal(base, powDecimal(101e4, 99)); // 1.01^99
         base = mulDecimal(base, powDecimal(1005e3, n - 300)); // 1.005^(n-300)
     }

     return base / 1e6 * 1e6;
 }
```

Solution

It's recommended to modify the code to correctly apply 100 growth iterations for each complete stage. The corrected implementation should use 100 instead of 99 for completed stages.

Status

Fixed

[N5] [Low] Missing Explicit Return Statement in powDecimal Function

Category: Others

Content

In the OURODataLib contract, the powDecimal function uses named return variables but lacks an explicit return statement. While the function may work correctly with implicit returns in some Solidity versions, it creates potential issues with readability, maintainability, and compatibility with different compiler versions.

Code location:

contracts/OURODataLib.sol#L147-L156

```
    function powDecimal(uint256 base, uint256 exp) internal pure returns (uint256
result) {
        result = 1e6; // 相当于1.0
        while (exp > 0) {
            if (exp % 2 == 1) {
                result = mulDecimal(result, base);
            }
            base = mulDecimal(base, base);
            exp /= 2;
        }
    }
```

Solution

It’s recommended to add an explicit return statement at the end of the function.

Status

Fixed

[N6] [Low] Array Input Validation Issues in Minter Information Query Function

Category: Design Logic Audit

Content

The getMintersInfoAndAllowance function in OUROMinter contract lacks proper input validation, creating two potential issues: denial-of-service vulnerability through unbounded array processing and incorrect result calculation due to duplicate address handling.

The function accepts an arbitrary-length array of minter addresses without size limitations, allowing malicious actors

to submit extremely large arrays that could cause transaction failures due to gas exhaustion. Additionally, the function does not check for duplicate addresses in the input array, causing the whitelistedCount variable to be inflated when the same whitelisted address appears multiple times. This leads to oversized result arrays and potential inconsistencies in the returned data.

Code location:

contracts/OUROMinter.sol#L106-L137

```
    function getMintersInfoAndAllowance(address[] memory minterAddressList) external
view returns (MinterInfo[] memory, uint256[] memory) {
        require(minterAddressList.length > 0, "Empty minter addresses array");

        // Count whitelisted addresses first
        uint256 whitelistedCount = 0;
        for (uint256 i = 0; i < minterAddressList.length; i++) {
            address minter = minterAddressList[i];
            if (minterWhiteList[minter]) {
                whitelistedCount++;
            }
        }

        // Initialize return arrays with correct size
        MinterInfo[] memory mintersInfo = new MinterInfo[](whitelistedCount);
        uint256[] memory todayAllowance = new uint256[](whitelistedCount);

        // Calculate current day timestamp
        uint256 currentDay = block.timestamp / 1 days;

        // Populate arrays with minter information
        uint256 index = 0;
        for (uint256 i = 0; i < minterAddressList.length; i++) {
            address minter = minterAddressList[i];
            if (minterWhiteList[minter]) {
                mintersInfo[index] = minterInfo[minter];
                todayAllowance[index] = minterInfo[minter].dailyAllowance -
minterDailyMinted[minter][currentDay];
                index++;
            }
        }

        return (mintersInfo, todayAllowance);
    }
```

Solution

It's recommended to implement proper input validation and duplicate address handling and define a reasonable upper limit constant to prevent DoS attacks through excessive array sizes.

Status

Fixed

[N7] [Suggestion] Redundant require statement followed by conditional block

Category: Design Logic Audit

Content

In the OURODataLib contract, the getCummulativeRateByEpoch function includes both a require statement and an if statement checking the same condition. Since the require statement checks that the difference between endEpoch and startEpoch is 100 or less, this makes the subsequent if statement checking `endEpoch - startEpoch > 100` redundant. This redundancy increases gas costs and creates confusion about the intended behavior.

Note: There is a typo in getCummulativeRateByEpoch ->"cumulative".

Code location:

contracts/OURODataLib.sol#L47-L50

```
    function getCummulativeRateByEpoch(uint32 startEpoch, uint32 endEpoch) internal
pure returns (uint) {
        require(endEpoch >= startEpoch, "End epoch must be greater than or equal to
start epoch");
        require(endEpoch - startEpoch <= 100, "Epoch difference cannot exceed 100");
        if (endEpoch - startEpoch > 100) {
            endEpoch = startEpoch + 100;
        }
        ...
    }
```

Solution

It's recommended to choose one approach - either keep the strict require statement or use the conditional adjustment, but not both.

Status

Fixed

[N8] [Suggestion] Overcomplicated multi-region calculation

Category: Design Logic Audit

Content

The getCummulativeRateByEpoch function unnecessarily splits the rate calculation into three parts: start region, middle regions (via loop), and end region. However, due to the maximum epoch difference constraint of 100 and minimum region size of 100, any two epochs can only span at most two adjacent regions. This means that the middle region loop is redundant, and the calculation can be simplified to just two parts: the start region component and the end region component.

Code location:

contracts/OURODataLib.sol#L45-L103

```
    function getCummulativeRateByEpoch(uint32 startEpoch, uint32 endEpoch) internal
pure returns (uint) {
        require(endEpoch >= startEpoch, "End epoch must be greater than or equal to
start epoch");
        require(endEpoch - startEpoch <= 100, "Epoch difference cannot exceed 100");
        ...
        uint rate;

        // Calculate first region
        if (boundaries[startRegion] >= startEpoch) {
            rate = getCummulativeRate(boundaries[startRegion] - startEpoch) /
factors[startRegion];
        }

        // Add rates for complete regions in between
        for (uint32 i = startRegion + 1; i < endRegion; i++) {
            if (boundaries[i] >= boundaries[i-1]) {
                rate += getCummulativeRate(boundaries[i] - boundaries[i-1]) /
factors[i];
            }
        }

        // Add final partial region
        if (endEpoch >= boundaries[endRegion-1]) {
            // rate += ( getCummulativeRate(endEpoch - startEpoch) -
getCummulativeRate(endEpoch - boundaries[endRegion-1]) / factors[endRegion];
            rate += (getCummulativeRate(endEpoch - startEpoch) -
getCummulativeRate(boundaries[endRegion-1] - startEpoch)) / factors[endRegion];

        }
```

```
        return rate;
    }
```

Solution

It's recommended to simplify the calculation to just two parts: calculate the start region contribution (from startEpoch to region boundary) and end region contribution (from boundary to endEpoch).

Status

Fixed

[N9] [Suggestion] Function visibility optimization

Category: Gas Optimization Audit

Content

Multiple functions are declared as public but are not called internally within the OUROProtocolUpgradeable contract. Using the external visibility modifier would be more efficient when the function is called as part of an on-chain transaction. While view functions don't cost gas when called externally through eth_call (off-chain query), they do consume gas when called as part of an on-chain transaction by another non-view function. In those cases, external functions are more gas efficient than public functions because of how they handle function parameters.

Code location:

contracts/OUROProtocolUpgradeable.sol#L449, L476, L498, L574, L596

contracts/OUROVipManager.sol#L83, L87

```
    function getEpochInfo(uint256 epoch) public view returns
    function userCumulativeApplyableRewardsAmount() public view returns (uint256)
    function getUserCBNAppliedInfo(address user, uint32 epoch) public view returns
(uint32, uint256)
    function currentEpochPendingAmount() public view returns (uint256)
    function calculateClaimableAmount() public view returns (uint256)
    function isNormalVip(address user) public view returns (bool)
    function isGenesisVip(address user) public view returns (bool)
```

Solution

It's recommended to change the function visibility from public to external to optimize potential on-chain gas usage.

Status

Fixed

[N10] [Suggestion] Missing the event record

Category: Others

Content

The following functions are missing event records after calling the modification.

Code location:

contracts/OUROProtocolUpgradeable.sol#L722-L728

```
function addTransferBurnExemptSenderAddress(address addr) external onlyOwner {
    transferBurnExemptSenders[addr] = true;
}

function removeTransferBurnExemptSenderAddress(address addr) external onlyOwner {
    transferBurnExemptSenders[addr] = false;
}
```

Solution

It is recommended to record events when sensitive parameters are modified for self-inspection or community review.

Status

Fixed

[N11] [Suggestion] Preemptive Initialization

Category: Race Conditions Vulnerability

Content

By calling the initialize function to initialize the contract, there is a potential issue that malicious attackers can preemptively call the initialize function to initialize.

Code location:

contracts/OUROProtocolUpgradeable.sol#L155-L200

```
function initialize(
    ...
) public initializer {
```

```
        ...
    }
```

Solution

It is suggested that the initialization operation can be called in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

Status

Acknowledged

[N12] [Suggestion] Redundant State Variable Assignment in Initialize Function

Category: Scoping and Declarations Audit

Content

The initialize function in OUROProtocolUpgradeable contains redundant assignment of the currentEpochNumber variable, where the same value is assigned twice consecutively without any intervening logic that would modify the variable.

Code location:

contracts/OUROProtocolUpgradeable.sol#L185, L192

```
    function initialize(
        ...
    ) public initializer {
        ...
        pendingEpochNumber = 1;
        currentEpochNumber = 1;
        ...
        // init the first epoch
        currentEpochNumber = 1;  // Set initial epoch to 1
        ...
    }
```

Solution

It's recommended to remove the redundant assignment of currentEpochNumber = 1 in the initialize function.

Status

Fixed

[N13] [Suggestion] Missing zero address check

Category: Others

Content

In the following function call and construct, invoking the contract can modify or set the address, but there is no check for the zero address.

Code location:

contracts/OUROProtocolUpgradeable.sol#L772-L728

contracts/OUROVipManager.sol#L32-L40

```
function addTransferBurnExemptSenderAddress(address addr) external onlyOwner {
    transferBurnExemptSenders[addr] = true;
}

function removeTransferBurnExemptSenderAddress(address addr) external onlyOwner {
    transferBurnExemptSenders[addr] = false;
}

constructor(
    address _ousd,
    address _normalVip,
    address _genesisVip
) Ownable(msg.sender) {
    ousd = _ousd;
    normalVip = _normalVip;
    genesisVip = _genesisVip;
}
```

Solution

It is recommended to add the 0 address check when sensitive parameters are modified.

Status

Fixed

[N14] [Information] Integer Division Precision Loss in Multiple Functions

Category: Arithmetic Accuracy Deviation Vulnerability

Content

Multiple functions in the contract perform integer division followed by multiplication with the same value, causing

significant precision loss. This anti-pattern appears in both epochCBNVolume and epochDuration functions, leading to inconsistent results and potential value loss.

For epochCBNVolume:

When base = 999,999: Result is 0 (100% value loss)

When base = 1,999,999: Result is 1,000,000 (50% value loss)

Values just below multiples of 1e6 lose all fractional parts

For epochDuration:

Epoch 57: Exact value is 5.99 hours, but returns 5 hours (16.56% loss)

Epoch 95: Exact value is 8.00 hours, but returns 7 hours (12.47% loss)

Values just below whole hours are truncated, with relative impact higher at lower epochs

This pattern creates an inconsistent distribution of values and may lead to unfair token allocation or scheduling issues for time-dependent operations.

Code location:

contracts/OURODataLib.sol#L180, L201

```
    function epochCBNVolume(uint256 n) internal pure returns (uint256) {
        ...
        return base / 1e6 * 1e6;
     }
    function epochDuration(uint256 epoch) internal pure returns (uint256) {
        ...
        if (epoch <= 800) {
            uint256 e = epoch + 1;
            uint256 log2Part = log2Approximation(e);
            uint256 linearPart = epoch * 12 * 3600 / 1000 + 95 * epoch * epoch * 3600
/ 1e6;
            duration = (log2Part * 3600 + linearPart) / 3600 * 3600;
        } else {
            duration = 86 * 3600;
        }

        return duration;
    }
```

Solution

If rounding or truncation is not the intended behavior, reverse the operation order to avoid precision loss. If truncation

is an intentional design choice, add explanatory comments to clarify this behavior.

Status

Acknowledged

[N15] [Information] Airdrop Level Hard Cap May Cause User Exclusion in Edge Cases

Category: Design Logic Audit

Content

The airdrop system in the OUROProtocolUpgradeable contract contains a hard-coded limit at level 64 that could theoretically prevent users from applying for future airdrops. However, the practical likelihood of users reaching this limit is extremely low due to the exponential growth model and economic constraints built into the protocol.

The airdrop system uses an exponential growth model where each level requires 2^level OUSD to apply (1, 2, 4, 8, 16... OUSD). To reach level 64, a user would need to cumulatively apply for 2^64 - 1 ≈ 18.4 quintillion OUSD, which is mathematically impossible given the protocol's reward structure. Even reaching level 20 would require cumulative applications of over 1 million OUSD, which would necessitate burning approximately 104,000 OUSD initially to generate sufficient CBN rewards (assuming maximum 100x reward multiplier over 100 epochs).

The economic reality makes level 64 practically unreachable: users encounter VIP requirements at level 7 (cumulative 127 OUSD), and the cost-to-reward ratio becomes economically unfavorable around level 15-20. Most users will naturally be constrained by their available CBN rewards long before approaching the theoretical limit. The exponential growth ensures that each successive level requires exponentially more rewards, creating a natural economic barrier.

Code location:

contracts/OUROProtocolUpgradeable.sol#L639-L647

```
function userNextApplyAmount() public view returns (uint256) {
    return calculateApplyAmount(userInfos[msg.sender].airdropLevel);
}

function calculateApplyAmount(uint64 level) public view returns (uint256) {
    require(level < 64, "Level too high");
    uint256 claimable = 1 << level;
    return claimable * 10 ** IERC20MetadataUpgradeable(ousd).decimals();
}
```

Solution

It’s recommended to add a level reset mechanism that allows high-level users to reset their level at a cost. Or remove the hard-coded level 64 limit or implement a more sustainable growth model.

Status

Acknowledged

[N16] [Information] Epoch Advancement Requires User Activity Triggering

Category: Design Logic Audit

Content

The OUROProtocolUpgradeable contract implements a passive epoch advancement mechanism that relies on user transactions to trigger epoch progression checks. The _checkAndAdvanceCurrentEpoch function is only called during OUSD transfer operations via handleTransferTax, meaning that epoch advancement depends entirely on user activity rather than automatic time-based progression. While this design is generally functional, it creates a theoretical risk of epoch stagnation during periods of low user activity.

The epoch advancement mechanism requires users to perform OUSD transfers (which trigger the 10% transfer tax) to invoke the epoch progression check. Without such transactions, epochs may remain static even after their designated time periods have elapsed, potentially causing user rewards to stop accumulating despite the passage of real-world time. This could create a negative feedback loop where users lose interest due to stagnant rewards, leading to further reduced activity and prolonged epoch stagnation.

However, the practical risk of this issue is minimal due to several mitigating factors: any OUSD transfer operation will trigger the epoch check, the protocol's economic incentives naturally encourage ongoing user participation, users have strong motivations to trigger epoch advancement to unlock their accumulated rewards, and even brief periods of activity can quickly catch up multiple delayed epochs. The epoch advancement condition also includes economic balance requirements (totalCBN * OUSD_TO_CBN_RATE >= totalAirdropApplied), ensuring that epochs only advance when the protocol can sustainably support the progression.

During normal protocol operation, the constant flow of user transactions (deposits, withdrawals, transfers, airdrop applications, and minter operations) provides natural epoch advancement triggers. The economic design incentivizes users to remain active, particularly as their CBN rewards accumulate and become available for airdrop applications. Additionally, any user who wants to claim accumulated rewards has a direct incentive to trigger a transaction that

would advance stalled epochs.

Code location:

contracts/OUROProtocolUpgradeable.sol#L416-L428

```
    function handleTransferTax(address from, address to, uint256 amount) external
nonReentrant returns (uint256) {
        ...
    }

    function _checkAndAdvanceCurrentEpoch() private {
        ...
    }
```

Solution

While epoch advancement theoretically depends on user activity, the protocol's economic incentive structure and normal usage patterns provide sufficient natural triggering mechanisms. The risk of prolonged epoch stagnation is minimal in practice, as users have strong motivations to maintain activity levels that naturally trigger epoch progression. No immediate action is required, though future versions might consider adding automated epoch advancement mechanisms for enhanced robustness.

Status

Acknowledged

[N17] [Information] Token Compatibility Reminder

Category: Others

Content

The OUSD token implements non-standard ERC20 behavior that deviates from typical token transfer mechanics. During normal transfer operations, OUSD automatically applies a tax mechanism that burns a portion of the transferred amount and generates CBN tokens for users. This non-standard behavior may cause compatibility issues when integrating with external protocols, decentralized exchanges, or other smart contracts that expect standard ERC20 token behavior.

External protocols typically assume that when transferring X tokens, exactly X tokens will be received by the recipient. However, OUSD's tax mechanism means that the actual amount received will be less than the amount sent

(after tax deduction), and additional side effects occur (CBN token generation, referrer rewards, epoch allocations).

This can lead to unexpected behavior in:

Automated Market Makers (AMMs) and decentralized exchanges

Lending and borrowing protocols

Multi-token operations expecting precise balance calculations

Smart contracts that don't account for deflationary token mechanics

Integration with standard DeFi protocols expecting vanilla ERC20 behavior

Developers and users should be aware of these non-standard characteristics when integrating OUSD with external systems or protocols.

Solution

N/A

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002505230001	SlowMist Security Team	2025.05.19 - 2025.05.23	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and the SlowMist team's analysis tool to audit the project. During the audit work, we found 2 medium risks, 3 low risks, 7 suggestions, and 4 information. All the findings were fixed or acknowledged. The code was not deployed to the mainnet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist