Security Audit Report for Ouro Stable Coin

Date: Dec 31, 2021

Version: 1.0

Contact: contact@blocksecteam.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2021.12.31 11:36:18 +08'00'

Contents

1 Introduction **1**
1.1 About Target Contracts 1
1.2 Disclaimer 1
1.3 Procedure of Auditing 2
1.3.1 Software Security 2
1.3.2 DeFi Security 2
1.3.3 NFT Security 2
1.3.4 Additional Recommendation 3
1.4 Security Model 3

2 Findings **4**
2.1 DeFi Security 4
2.1.1 Sandwich attack I (MEV) 4
2.1.2 Sandwich attack II 5
2.1.3 Sandwich attack III 6
2.1.4 Centralization risk 6
2.1.5 The inconsistent bookkeeping for deflation tokens 7
2.2 Additional Recommendation 7
2.2.1 Add view functions to facilitate users to check prices 7
2.2.2 Use SafeMath library 7

Report Manifest

Item	Description
Client	Ouro Finance
Target	Ouro Stable Coin

Version History

Version	Date	Description
1.0	Dec 31, 2021	First Release

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The files that are audited in this report include the following ones.

Repo Name	Github URL
ouro-official/contracts	https://github.com/ouro-official/contracts

The auditing process is iterative. Specifically, we will further audit the commits that fix the founding issues. Furthermore, we also audit the new commits during the audit. Thus, there are multiple commit SHA values referred in this report. The commit SHA values before and after the audit are shown in the following.

Before and during the audit

Project		Commit SHA
ouro-official/contracts	C1	f44daf653ba39f40366e762511639fa5f4e3e9fb
	C2	9419b3e648fb1e24651f38389168d3f7ebd725ab
	C3	781b8393688c1f851649d6e2261bc0e6d964a37b

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report do not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data Flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Access control
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [1] and Common Weakness Enumeration [2]. Accordingly, the severity measured in this report are classified into four categories: **High**, **Medium**, **Low** and **Undetermined**.

[1] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[2] https://cwe.mitre.org/

Chapter 2 Findings

In total, we find five potential issues in the smart contract. We also have two recommendations, as follows:

- High Risk: 0
- Medium Risk: 0
- Low Risk: 5
- Recommendations: 2

ID	Severity	Description	Category	Status
1	Low	*Sandwich attack I (MEV)*	DeFi Security	
2	Low	*Sandwich attack II*	DeFi Security	Acknowledged
3	Low	*Sandwich attack III*	DeFi Security	Acknowledged
4	Low	*Centralization risk*	DeFi Security	Acknowledged
5	Low	*The inconsistent bookkeeping for deflation tokens*	DeFi Security	Acknowledged
6	-	*Add view functions to facilitate users to check prices*	Recommendation	Confirmed and Fixed
7	-	*Use SafeMath library*	Recommendation	

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Sandwich attack I (MEV)

Status TBD

Description

There are two public functions in the contract `OUROReserve` that make trades in the Pancake (AMM). As shown in the following code snippet, if the value of the collateral asset rises to the threshold, `rebase` will sell the collateral asset for OGS. Otherwise, it will mint OGS to buy the collateral asset. The function `distributeRevenue` will sell the XVS rewards and the asset revenue for OGS.

In order to prevent the Pancake pool from being manipulated by flash loan before invoking the two functions, Ouro Finance allows only EOA to invoke the two functions. However, the MEV sandwich attack still works. This issue exists in commit C1, C2, C3.

```
580    function rebase() public {
581        // only from EOA or owner address
582        _require((msg.sender == owner()) || (!msg.sender.isContract() && msg.sender == tx.origin));
583        ......
584    }
```

```
1007    function distributeRevenue() external {
1008        // only from EOA
1009        _require(!msg.sender.isContract() && msg.sender == tx.origin);
1010
1011        _distributeXVS();
```

```
        _distributeAssetRevenue();

        // log
        emit RevenueDistributed();
    }
```

Impact If these two functions have not been called for too long or the price of the collateral asset raises too much, then the benefits of the attack may be high enough to attract an attacker to initiate a MEV sandwich attack through Flashbot, which may cause OURO lose money. In addition, if the Pancake does not exist the pool of the collateral asset/WBNB, the MEV sandwich attack is also profitable, because the attacker can create the pool with his expected ratio in advance.

Suggestion I Check slippage before interacting with the AMM pool.

Suggestion II Check whether the pool exists in the function `newCollateral`.

2.1.2 Sandwich attack II

Status Acknowledged.

Description

The public function `revenueArrival` in the contract `OURODist` sells its asset balance for OGS in Pancake and adds liquidity to Pancake OGS related pool. There is only one place in the entire project that transfers assets to this contract: the public function `distributeRevenue` in the contract `OUROReserve`. The function `distributeAssetRevenue` transfers revenue of the contract `OUROReserve` to the contract `OURODist`, and invokes the function `revenueArrival` to sell it immediately. Since the function `distributeAssetRevenue` is protected by the `onlyEOA`, the mechanism is relatively safe. This issue exists in commit `C1, C2, C3`.

```
    function revenueArrival(address token, uint256 revenueAmount) external override {
        // lazy approve
        if (token != WBNB) {
            if (IERC20(token).allowance(address(this), address(router)) == 0) {
                IERC20(token).safeIncreaseAllowance(address(router), MAX_UINT256);
            }
        }

        // 50% - OGS token buy back and burn.
        uint256 revenueToBuyBackOGS = revenueAmount
                                    .mul(50)
                                    .div(100);
        _revenueToBuyBackOGS(token, revenueToBuyBackOGS);

        // 50% - Split to form LP tokens for the platform.
        uint256 revenueToFormLP = revenueAmount.sub(revenueToBuyBackOGS);
        _revenueToFormLP(token, revenueToFormLP);

        // log
        emit RevenuArrival(token, revenueAmount);
    }
```

Impact Any asset balance of the contract `OURODist` can be withdrawn by anyone (by launching a sandwich attack via flash loan).

Suggestion Do not expose the function `revenueArrival` to everyone. Otherwise, make sure do not transfer money to the contract `OURODist` directly.

Feedback from the project The project team will use keep3r to invoke distributeRevenue() periodically to mitigate the MEV.

2.1.3 Sandwich attack III

Status Acknowledged.

Description

The `deposit`, `withdraw`, `claimOGSReward`, and `claimOUROReward` in the `AssetStaking` contract can be invoked by smart contracts. They will trigger the internal function `updateOuroReward`, which sells XVS for asset in Pancake. An attacker can leverage flash loan to make two reverse trades in the Pancake XVS/WBNB pool, which can perform the sandwich attack on the invocation of one of these functions. The attack can make Ouro lose the previous block's XVS rewards from Venus. If the attacker launches the attack in every block, Ouro will lose all the XVS rewards.

However, we also note that, even the flash loan sandwich attack works, the XVS rewards may be not enough to cover transaction fees. That's because the XVS rewards will not accumulate as long as someone interacts with the contracts. This issue exists in commit `C1, C2, C3`.

Impact If the contract `AssetStaking` is not invoked for too long, the accumulated XVS rewards may be stolen by the flashloan sandwich attack.

Suggestion Check slippage before interacting with an AMM pool.

Feedback from the project The Ouro Finance think it's fine for their usage scenario.

2.1.4 Centralization risk

Status Acknowledged.

Description

The function `emergencyWithdraw` in the contract `OUROReserve` can let the project owner withdraw all the collaterals. This issue exists in commit `C1, C2, C3`.

```
311    function emergencyWithdraw(address to) external onlyOwner {
312        uint n = collaterals.length;
313        for (uint i=0;i<n;i++) {
314            // withdraw all tokens
315            uint256 amount = IERC20(collaterals[i].vTokenAddress).balanceOf(address(this));
316            if (amount > 0) {
317                IERC20(collaterals[i].vTokenAddress).safeTransfer(to, amount);
318            }
319        }
320
321        // log
322        emit EmergencyWithdraw(msg.sender, to);
323    }
```

Impact The existence of the function `emergencyWithdraw` makes the project centralized.

Suggestion Remove this function and use the pause mechanism to protect the `OUROReserve` contract. Besides, deploying security mechanisms to protect the private key of the contract owner.

Feedback from the project In order to evacuate and protect assets at the time of attack, the team decided not to remove this function at the current stage. The team has planned to add the DAO governance operated with OGS in the future to mitigate this issue, when the contract has proven to be bug free after the launch.

2.1.5 The inconsistent bookkeeping for deflation tokens

Status Acknowledged.

Description

The project does not use the balance change to record the amount of the collateral deposited by users. Therefore, Ouro Stable Coin does not support deflation tokens. This issue exists in commit C1, C2, C3.

Impact Supporting deflation tokens as collateral causes the contract OUROStaking lose all the collateral assets.

Suggestion Check the balance before and after the deposit of the collateral. Besides, the project can have a whitelist to ensure that the collateral cannot be a deflation token.

2.2 Additional Recommendation

2.2.1 Add view functions to facilitate users to check prices

Status Confirmed and fixed.

Description

Since the deposit and withdraw of the contract OUROReserve use the chainlink price oracle and the current OuroPrice to calculate the amount of OURO to be minted and the amount of collateral assets to be redeemed, the user can not know the number of assets they will get before invoking the two functions. A view function to retrieve the value could be added to facilitate the user. This recommendation applies to commit C1 and has been fixed in commit C2.

Impact NA

Suggestion Add two view functions to check prices.

2.2.2 Use SafeMath library

Status TBD

Description

The code L89 and L90 in the contract LPStaking may cause an integer overflow. This recommendation applies to commit C1, C2, C3.

```
83  function deposit(uint256 amount) external nonReentrant whenNotPaused {
84          require(amount > 0, "zero deposit");
85          // settle previous rewards
86          settleStaker(msg.sender);
87  
88          // modifiy
89          _balances[msg.sender] += amount;
90          _totalStaked += amount;
91  
92          // transfer asset from AssetContract
93          IERC20(assetContract).safeTransferFrom(msg.sender, address(this), amount);
94  
95          // log
96          emit Deposit(msg.sender, amount);
97      }
```

Impact NA

Suggestion Use the SafeMath library or upgrade to the latest version of the solidity compiler that has the integer overflow checking.