Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.07.08, the SlowMist security team received the Alpaca Finance team's security audit application for Alpaca Finance Oracle, 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 party 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.

Level	Description
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:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Aduit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit version:

OracleMedianizer: https://github.com/alpaca-finance/bsc-alpaca-contract/blob/main/contracts/6/protocol/OracleMedianizer.sol

commit: dff7932581f5a838a695905a8c5f3816fd1525ff

ChainLinkOracle: https://github.com/alpaca-finance/bsc-alpaca-contract/blob/main/contracts/6/protocol/price-oracle/ChainLinkPriceOracle.sol

commit: edcc0ac483200b6022bd0182cce0f403b31d322a

Fixed version:

OracleMedianizer: https://github.com/alpaca-finance/bsc-alpaca-contract/blob/main/contracts/6/protocol/OracleMedianizer.sol

commit: 958c812a958fd36c5206f8f59797d0a6e934ba27

ChainLinkOracle: https://github.com/alpaca-finance/bsc-alpaca-contract/blob/main/contracts/6/protocol/price-oracle/ChainLinkPriceOracle.sol

commit: edcc0ac483200b6022bd0182cce0f403b31d322a

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Function visibility issue	Others	Suggestion	Fixed
N2	The Token Pair Check	Design Logic Audit	Low	Discussed

4 Code Overview

4.1 Contracts Description

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:

OracleMedianizer			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
setPrimarySources	External	Can Modify State	onlyOwner

OracleMedianizer			
setMultiPrimarySources	External	Can Modify State	onlyOwner
_setPrimarySources	Internal	Can Modify State	-
_getPrice	internal	-	-
getPrice	External	-	-

ChainLinkPriceOracle			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
setPriceFeeds	External	Can Modify State	onlyOwner
_setPriceFeed	Internal	Can Modify State	-
getPrice	External	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Function visibility issue

Category: Others

Content

In the OracleMedianizer contract, the user can get the price of the pair token through the getPrice function. The getPrice function will call the `_getPrice` function to get the price, but the visibility of the `_getPrice` function is public.

Code location:

```
function _getPrice(address token0, address token1) public view returns (uint256) {
    uint256 candidateSourceCount = primarySourceCount[token0][token1];
```

```
    require(candidateSourceCount > 0, "OracleMedianizer::getPrice:: no primary
source");
    uint256[] memory prices = new uint256[](candidateSourceCount);
    // Get valid oracle sources
    uint256 validSourceCount = 0;
    for (uint256 idx = 0; idx < candidateSourceCount; idx++) {
      try primarySources[token0][token1][idx].getPrice(token0, token1) returns
(uint256 price, uint256 lastUpdate) {
        if (lastUpdate >= now - maxPriceStales[token0][token1]) {
          prices[validSourceCount++] = price
        }
      } catch {}
    }
    require(validSourceCount > 0, "OracleMedianizer::getPrice:: no valid source");
    // Sort prices (asc)
    for (uint256 i = 0; i < validSourceCount - 1; i++) {
      for (uint256 j = 0; j < validSourceCount - i - 1; j++) {
        if (prices[j] > prices[j + 1]) {
          (prices[j], prices[j + 1]) = (prices[j + 1], prices[j]);
        }
      }
    }
    uint256 maxPriceDeviation = maxPriceDeviations[token0][token1];
    // Algo:
    // - 1 valid source --> return price
    // - 2 valid sources
    //     --> if the prices within deviation threshold, return average
    //     --> else revert
    // - 3 valid sources --> check deviation threshold of each pair
    //     --> if all within threshold, return median
    //     --> if one pair within threshold, return average of the pair
    if (validSourceCount == 1) return prices[0]; // if 1 valid source, return
    if (validSourceCount == 2) {
      require(
        prices[1].mul(1e18) / prices[0] <= maxPriceDeviation,
        "OracleMedianizer::getPrice:: too much deviation 2 valid sources"
      );
      return prices[0].add(prices[1]) / 2; // if 2 valid sources, return average
    }
    bool midP0P1Ok = prices[1].mul(1e18) / prices[0] <= maxPriceDeviation;
    bool midP1P2Ok = prices[2].mul(1e18) / prices[1] <= maxPriceDeviation;
    if (midP0P1Ok && midP1P2Ok) return prices[1]; // if 3 valid sources, and each
pair is within thresh, return median
    if (midP0P1Ok) return prices[0].add(prices[1]) / 2; // return average of pair
within thresh
```

```
    if (midP1P2Ok) return prices[1].add(prices[2]) / 2; // return average of pair
within thresh
    revert("OracleMedianizer::getPrice:: too much deviation 3 valid sources");
  }

  function getPrice(address token0, address token1) external view override returns
(uint256, uint256) {
    return (_getPrice(token0, token1), block.timestamp);
  }
```

Solution

If this is not the expected design, it is recommended to set the visibility of the `_getPrice` function to internal and only keep the getPrice function for price acquisition.

Status

Fixed

[N2] [Low] The Token Pair Check

Category: Design Logic Audit

Content

There is a `_setPriceFeed` function in the ChainLinkPriceOracle contract, which is used to set the source of the token pair. In the function, check whether `priceFeeds[token1][token0]` already exists, but then set the source for `priceFeeds[token0][token1]`.

Code location:

```
  function _setPriceFeed(
    address token0,
    address token1,
    AggregatorV3Interface source
  ) internal {
    require(
      address(priceFeeds[token1][token0]) == address(0),
      "ChainLinkPriceOracle::setPriceFeed:: source on existed pair"
    );
    priceFeeds[token0][token1] = source;
```

```
    emit SetPriceFeed(token0, token1, source);
}
```

Solution

If this is not the expected design, it is recommended to check whether `priceFeeds[token0][token1]` exists.

Status

Discussed; After communicating with the project party, the project party said: This is intended by design. The reason we check `priceFeeds[token1][token0]` only is because we allow to add one combination per pair. Then we will reverse it if other contracts query for a reverse of the pair.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002107120002	SlowMist Security Team	2021.07.08 - 2021.07.12	Passed

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found one low-risk, one suggestion vulnerabilities. And one low-risk vulnerability was discussed; All other findings were fixed. 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