Security Audit Report for Multiswap-ng

Date: Mar 21, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	stratum
Target	Multiswap-ng

Version History

Version	Date	Description
1.0	Mar 21, 2024	First Release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

Email: audit@blocksec.com

Date: 2024-03-28 03:00:47+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

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

The target of this audit is the code repository of Multiswap-ngof stratum.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Multiswap-ng	`Version 1`	`b519fd3a55d2680f7f4ae17feef1b70fdf773cac`[1]
	`Version 2`	`273990f92ad0c2621dbe3095299358077889a755`[2]

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 does 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.

[1] The original code in repository `https://github.com/stratum-exchange/LSTSwap`

[2] The final code in repository `https://github.com/stratum-exchange/multiswap-ng`

- **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
- Permission management
- 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]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

		Likelihood	
		High	*Low*
Impact	*High*	High	Medium
	Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

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

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

Chapter 2 Findings

In total, we find **two** potential issues. Besides, we also have **three** recommendations and **three** notes.

- Medium Risk: 2
- Recommendation: 3
- Note: 3

ID	Severity	Description	Category	Status
1	Medium	Lack of Check in Function _fetchPrice()	DeFi Security	Fixed
2	Medium	Withdrawal of Admin Fees by Various Privileges	DeFi Security	Fixed
3	-	Incorrect Error Message	Recommendation	Fixed
4	-	Lack of Check in Function updatePrice()	Recommendation	Fixed
5	-	Redundant Code	Recommendation	Fixed
6	-	Timely Updates of the Price Oracle	Note	-
7	-	Incompatible with Non-18 Decimal Price Response	Note	-
8	-	SwapFee Claimed by RebaseHandler	Note	-

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Lack of Check in Function _fetchPrice()

Status Fixed in Version 2.

Introduced by Version 1

Description In function _fetchPrice() of the contract Api3PriceFeed, if the obtained response fails to pass relevant checks, it will directly return the lastGoodPrice. However, this lastGoodPrice does not guarantee timely updates and lacks verification, potentially returning an outdated price.

```
function _fetchPrice() internal view returns (Status, uint256) {
// Get current and previous price data from Api3, and current price data from Band
OracleResponse memory response = _getCurrentResponse();


// --- CASE 1: System fetched last price from Api3 ---
if (status == Status.oracleWorking) {
  // If Api3 is broken or frozen
  if (_oracleIsBroken(response) || _oracleIsFrozen(response)) {
    // If Api3 is broken, switch to Band and return current Band price
    return (Status.oracleUntrusted, lastGoodPrice);
  }


  // If Api3 is working, return Api3 current price (no status change)
```

```
    return (Status.oracleWorking, response.answer);
  }


  // --- CASE 2: Api3 oracle is untrusted at the last price fetch ---
  if (status == Status.oracleUntrusted) {
    if (_oracleIsBroken(response) || _oracleIsFrozen(response)) {
      return (Status.oracleUntrusted, lastGoodPrice);
    }


    return (Status.oracleWorking, response.answer);
  }
}
```

Listing 2.1: Api3PriceFeed.sol

Impact An outdated lastGoodPrice in function _fetchPrice() may be used.

Suggestion Add a check to ensure that the generation time of the lastGoodPrice is within a valid range.

2.1.2 Withdrawal of Admin Fees by Various Privileges

Status Fixed in Version 2.

Introduced by Version 1

Description In function withdrawAdminFees(), the privileged role rebaseHandler is allowed to withdraw admin fees to rebaseHandler. However, in function skim(), both the owner or rebaseHandler can withdraw admin fees to any addresses, which means the admin fees can be withdrawn by roles in different privileges

```
  function skim(
  address _to
) external virtual nonReentrant whenNotPaused onlyOwnerOrRebaseHandler {
  return swapStorage.skim(_to);
}
```

Listing 2.2: Swap.sol

```
  function withdrawAdminFees() external virtual nonReentrant {
  require(msg.sender == rebaseHandler, "Not rebaseHandler");
  swapStorage.withdrawAdminFees(rebaseHandler);
}
```

Listing 2.3: Swap.sol

Impact Admin fees can be withdrawn to any addresses by the privileged function skim().

Suggestion Remove the function skim().

2.2 Additional Recommendation

2.2.1 Incorrect Error Message

Status Fixed in Version 2.

Introduced by Version 1

Description In function swap(), the error message "Token index out of range" is incorrect.

```
  function swap(
  Swap storage self,
  uint8 tokenIndexFrom,
  uint8 tokenIndexTo,
  uint256 dx,
  uint256 minDy
) external returns (uint256) {
  {
    IERC20 tokenFrom = self.pooledTokens[tokenIndexFrom];
    require(tokenIndexFrom != tokenIndexTo, "Token index out of range");
    require(dx > 0, "do not exchange 0 tokens");
    require(
      dx <= tokenFrom.balanceOf(msg.sender),
      "Cannot swap more than you own"
    );
    // Transfer tokens first to see if a fee was charged on transfer
    uint256 beforeBalance = tokenFrom.balanceOf(address(this));
    tokenFrom.safeTransferFrom(msg.sender, address(this), dx);


    // Use the actual transferred amount for AMM math
    dx = tokenFrom.balanceOf(address(this)).sub(beforeBalance);
  }
  uint256 dy;
  uint256 dyFee;
  uint256[] memory xp;
  uint256[] memory balances = self.balances;
  (dy, dyFee, xp) = _calculateDY(
    self,
    tokenIndexFrom,
    tokenIndexTo,
    dx,
    balances
  );
  require(dy >= minDy, "Swap didn't result in min tokens");


  uint256 amp = _getAPrecise(self);
  upkeepOracles(self, xp, amp, getD(xp, amp));


  self.balances[tokenIndexFrom] = balances[tokenIndexFrom].add(dx);
  self.balances[tokenIndexTo] = balances[tokenIndexTo].sub(dy).sub(
    dyFee
```

```
 );


 self.pooledTokens[tokenIndexTo].safeTransfer(msg.sender, dy);


 emit TokenSwap(msg.sender, dx, dy, tokenIndexFrom, tokenIndexTo);


 return dy;
}
```

Listing 2.4: SwapUtils.sol

Suggestion Revise the error message accordingly.

2.2.2 Lack of Check in Function updatePrice()

Status Fixed in Version 2.

Introduced by Version 1

Description In function updatePrice(), each invocation of the function _fetchPrice() updates lastGoodPrice with the newly obtained price. An additional check can be implemented to prevent updates when the obtained price equals the recorded lastGoodPrice.

Description

```
 function updatePrice() external override returns (uint256) {
 (Status newStatus, uint256 price) = _fetchPrice();
 lastGoodPrice = price;
 if (status != newStatus) {
  status = newStatus;
  emit PriceFeedStatusChanged(newStatus);
 }
 return price;
}
```

Listing 2.5: Api3PriceFeed.sol

Suggestion Add checks to prevent unnecessary lastGoodPrice updates.

2.2.3 Redundant Code

Status Fixed in Version 2.

Introduced by Version 1

Description The function _scalePriceByDigits() in contract Api3PriceFeed is redundant.

```
 function _scalePriceByDigits(
   uint256 _price,
   uint32 _digits
  ) internal pure returns (uint256) {
   /*
```

```
     * Convert the price returned by the Api3 oracle to an 18-digit decimal for use by Liquity.
     * At date of Liquity launch, Api3 uses an 8-digit price, but we also handle the possibility of
     * future changes.
     */
    uint256 price;
    if (_digits >= TARGET_DIGITS) {
      // Scale the returned price value down to Liquity's target precision
      price = _price.div(10 ** (_digits - TARGET_DIGITS));
    } else if (_digits < TARGET_DIGITS) {
      // Scale the returned price value up to Liquity's target precision
      price = _price.mul(10 ** (TARGET_DIGITS - _digits));
    }
    return price;
  }
```

Listing 2.6: Api3PriceFeed.sol

Suggestion Remove the redundant code.

2.3 Note

2.3.1 Timely Updates of the Price Oracle

Description In the protocol, tokens' corresponding xp values are calculated using an external price oracle. If the oracle fails to update prices promptly for an extended period, significant arbitrage opportunities may arise over time, leading to losses for liquidity providers.

2.3.2 Incompatible with Non-18 Decimal Price Response

Description The protocol only supports Oracle responses with 18-decimal precision. Otherwise, the rate is incorrect. The team has confirmed the oracle API3's price responses are in precision of 18-decimal, and its security is not covered in this audit.

2.3.3 SwapFee Claimed by RebaseHandler

Description According to the protocol design, liquidity providers will not receive swap fees. The fee will be collected by the rebasehandler and sent to the gauge for distribution to gauge voters. Liquidity providers will only receive farming rewards.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS