Security Audit Report for Stafi Withdraw Contract

Date: April 07, 2023

Version: 1.0

Contact: contact@blocksec.com

BlockSec
Audit Team

Digitally signed by
BlockSec Audit Team
Date: 2023.04.07 18:00:37
+08'00'

Contents

Report Manifest

Item	Description
Client	Stafi Protocol
Target	Stafi Withdraw Contract

Version History

Version	Date	Description
1.0	April 07, 2023	First Release

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 5 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 withdraw contract [1] of the Stafi Protocol. The Stafi Protocol proposes the rETH solution to solve the illiquidity of staked ETH for both stakers and validators. Specifically, the staker can stake any amount of ETH through Stafi Protocol and receive a certain amount of rETH tokens. Then, the staked ETH will be allocated to a group of well-performing original validators to make rewards (rETH tokens as well). The original validators would establish and maintain appropriate amounts of validator nodes to provide staking rewards to stakers net of fees [2].

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
Stafi Protocol	`Version 1`	`1696f1b6b407461bbacd1d7ac8b5e9641fb8d460`
	`Version 2`	`ccedf13403c34516639c9244f90e1d690933914d`

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] https://github.com/stafiprotocol/eth2-staking/blob/v3/contracts/withdraw/StafiWithdraw.sol

[2] https://docs.stafi.io/rtoken-app/reth-solution

1.3 Auditing Approaches

This section provides an overview of the auditing approaches we adopted and the corresponding checkpoints we focused on. Specifically, we conduct the audit by engaging the following approaches:

- **Static analysis**. We scan smart contracts with both open-source and in-house tools, and then manually verify (reject or confirm) the issues reported by them.
- **Fuzzing testing**. We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an in-house fuzzing tool (developed and customized by our security research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Manual code review**. We manually review the design and the corresponding implementation of the whole project in a comprehensive manner, and check the known attack surface accordingly.

Generally, the checkpoints fall into four categories, i.e., **software security**, **DeFi security**, **NFT security** and **additional recommendation**. The main concrete checkpoints are summarized 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 above 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 [3] and Common Weakness Enumeration [4]. 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

Impact \ Likelihood	*High*	*Low*
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.

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

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

Chapter 2 Findings

In total, we find **one** potential issue. Besides, we also have **one** recommendation.

- Medium Risk: 1
- Recommendation: 1

ID	Severity	Description	Category	Status
1	Medium	Unfair claiming sequence for usres' unstaked assets	DeFi Security	Fixed
2	-	Use new version of OpenZeppelin library	Recommendation	Acknowledged

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Unfair claiming sequence for users' unstaked assets

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Based on the code logic, the staker must invoke the `withdraw` function to withdraw his/her unstaked Ether when there is no sufficient fund. Invoking the `withdraw` function will check the staker's `withdrawIndex` based on `maxClimableWithdrawIndex` as shown in the line 168 of the following code snippets. Therefore, two actions are required to launch to enable users to withdraw their unstaked Ether:

- (**Action 1**) Reserving Ether for withdrawing (e.g., through the `reserveEthForWithdraw` function).
- (**Action 2**) Updating `maxClaimableWithdrawIndex` (i.e., through the `distributeWithdrawals` function).

```
160  function withdraw(
161    uint256[] calldata _withdrawIndexList
162  ) external override onlyLatestContract("stafiWithdraw", address(this)) {
163    require(_withdrawIndexList.length > 0, "index list empty");
164
165    uint256 totalAmount;
166    for (uint256 i = 0; i < _withdrawIndexList.length; i++) {
167        uint256 withdrawIndex = _withdrawIndexList[i];
168        require(withdrawIndex <= maxClaimableWithdrawIndex, "not claimable");
169        require(unclaimedWithdrawalsOfUser[msg.sender].remove(withdrawIndex), "already claimed");
170
171        totalAmount = totalAmount.add(withdrawalAtIndex[withdrawIndex]._amount);
172    }
173
174    if (totalAmount > 0) {
175        (bool result, ) = msg.sender.call{value: totalAmount}("");
176        require(result, "user failed to claim ETH");
177    }
178
```

```
179     emit Withdraw(msg.sender, _withdrawIndexList);
180   }
```

Listing 2.1: StafiWithdraw.sol

However, suppose **Action 1** is executed first and there is a time gap between **Action 1** and **Action 2**. The unstaked asset of the later stakers can be claimed instantly. The former stakers have to wait for the finalization of **Action 2**, which will update `maxClaimableWithdrawIndex` to withdraw their unstaked assets. Furthermore, within the time gap between **Action 1** and **Action 2**, new users can unstake and instantly claim their assets reserved for former users. Obviously, such a claiming sequence is unfair.

Impact Unfair claiming sequence for users' unstaked assets.

Suggestion Update `maxClaimableWithdrawIndex` accordingly when `totalMissingAmountForWithdraw` is updated to zero.

2.2 Additional Recommendation

2.2.1 Use new version of OpenZeppelin library

Status Acknowledged

Introduced by `Version 1`

Description The OpenZeppelin library used in this project is out-of-date.

Impact The old version library might suffer from potential vulnerabilities [1].

Suggestion Use (stable) new version of OpenZeppelin library.

[1] https://github.com/OpenZeppelin/openzeppelin-contracts/security/advisories/GHSA-9c22-pwxw-p6hx