Security Audit Report for TakoKeysV1

Date: Jan 2, 2023

Version: 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	TakoProtocol
Target	TakoKeysV1

Version History

Version	Date	Description
1.0	Jan 2, 2023	First Version

About BlockSec The BlockSec Team focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team 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 released detailed analysis reports of high-impact security incidents. 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 [1] for the TakoKeysV1 contracts of TakoProtocol. The TakoKeysV1 contracts serve as a shares issuance market, enabling creators to issue shares represented by `FarcasterKey` NFTs. Creators can specify a piecewise pricing function, which transitions from a constant initial price to a curve-based price. Users are able to buy or sell shares at prices calculated from this piecewise function.

The auditing process is iterative. Specifically, we will 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. Our audit report is responsible for the only initial version (i.e., `Version 1`), as well as new codes (in the following versions) to fix issues in the audit report.

Project		Commit SHA
TakoKeysV1	`Version 1`	`f95ee9083bffbb5d1c06cbf35ea4c1ce07afb1ea`
	`Version 2`	`3c914c2f1acac51e93931675a2ea4ab0db34da37`

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/takoprotocol/TakoKeysV1/tree/Piecewise

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 [2] and Common Weakness Enumeration [3]. 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.

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

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

Chapter 2 Findings

In total, we find **two** potential issues. Besides, we also have **four** recommendations.

- Medium Risk: 1
- Low Risk: 1
- Recommendation: 4

ID	Severity	Description	Category	Status
1	Medium	Potential DoS to native token transfers due to insufficient gas	Software Security	Fixed
2	Low	Potential precision loss in the `_calculateFeesForPiecewise` function	Software Security	Fixed
3	-	Add a zero address check on `protocolFeeDestination`	Recommendation	Fixed
4	-	Add a sanity check on `sharesAmount`	Recommendation	Fixed
5	-	Remove unused contract	Recommendation	Confirmed
6	-	Fix typo	Recommendation	Fixed

The details are provided in the following sections.

2.1 Software Security

2.1.1 Potential DoS to native token transfers due to insufficient gas

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `_buySharesImp` function in the `ProfileMarketV1` contract uses `transfer` to refund overpaid native tokens. However, this `transfer` can fail if the recipient is a proxy contract with a fallback function that consumes a significant amount of gas, potentially resulting in a denial of service (DoS).

```
178    function _buySharesImp(uint256 creatorId, uint256 sharesAmount) internal {
179        address creator = _getCreatorById(creatorId);
180        uint256 supply = sharesSupply[creatorId];
181        fees memory fee = _calculateFeesForPiecewise(creatorId, sharesAmount, true);
182        uint256 totalFee = fee.price + fee.creatorFee + fee.protocolFee;
183        require(msg.value >= totalFee, "Insufficient payment");
184        // Refund if overpaid
185        if (msg.value > totalFee) {
186            payable(msg.sender).transfer(msg.value - totalFee);
187        }
188        sharesSupply[creatorId] += sharesAmount;
189        userClaimable[creator] += fee.creatorFee;
190        uint256[] memory tokenIds = farcasterKey.mint(
191            msg.sender,
192            sharesAmount,
193            creatorId
194        );
```

```
195         (bool success, ) = protocolFeeDestination.call{value: fee.protocolFee}("");
196         require(success, "Unable to send funds");
197         emit TradeEvent(
198             msg.sender,
199             creatorId,
200             true,
201             sharesAmount,
202             tokenIds,
203             fee,
204             supply + sharesAmount
205         );
206     }
```

Listing 2.1: ProfileMarketV1.sol

Impact Contract users using a proxy cannot buy shares due to the revert in the `_buySharesImp` function.

Suggestion Revise the code logic accordingly.

2.1.2 Potential precision loss in the `_calculateFeesForPiecewise` function

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `ProfileMarketV1` contract, the `_calculateFeesForPiecewise` function determines the protocol and creator fees based on the share price and fee percentage. However, there is a potential risk of precision loss when both the price and percentage are low, which could result in the calculated fee being rounded down to zero. In such cases, users would not incur any additional fees when buying or selling shares.

```
259     function _calculateFeesForPiecewise(uint256 creatorId, uint256 amount, bool isBuy) internal
            view returns (fees memory) {
260         require(amount > 0, "Amount not correct");
261         if(isBuy){
262             uint256 price = _getBuyPriceByPiecewise(creatorId, amount);
263             return fees(price, (price * protocolBuyFeePercent) / 1 ether, (price *
                    creatorBuyFeePercent) / 1 ether);
264         }else{
265             uint256 price = _getSellPriceByPiecewise(creatorId, amount);
266             return fees(price, (price * protocolSellFeePercent) / 1 ether, (price *
                    creatorSellFeePercent) / 1 ether);
267         }
268     }
```

Listing 2.2: ProfileMarketV1.sol

Impact The protocol and creators cannot collect fees from users when the fees are rounded down to zero.

Suggestion Revise the code logic accordingly.

2.2 Additional Recommendation

2.2.1 Add a zero address check on `protocolFeeDestination`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `setFeeDestination` function in the `ProfileMarketV1` contract does not verify that the new `protocolFeeDestination` address is non-zero. If `protocolFeeDestination` is set to a zero address, the protocol fees will be erroneously sent to the zero address and become irretrievable.

```
65    function setFeeDestination(address _feeDestination) external onlyOwner {
66        protocolFeeDestination = _feeDestination;
67        emit SetFeeTo(_feeDestination);
68    }
```

Listing 2.3: ProfileMarketV1.sol

Impact The protocol fee will be erroneously sent to the zero address and become irretrievable.

Suggestion Add the zero address check accordingly.

2.2.2 Add a sanity check on `sharesAmount`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `_createParamsVerification` function in the `ProfileMarketV1` contract only checks that `sharesAmount` is larger than 0 on line 153. However, this check is insufficient as `sharesAmount` represents the total supply of shares. The total supply should be set equal to or greater than the initial supply, specified by `idoAmount`. If the shares creator mistakenly sets `sharesAmount` to a value smaller than `idoAmount`, the curve price function will not be utilized when buying and selling shares.

```
151   function _createParamsVerification(uint256 creatorId, uint256 idoPrice, uint256 idoAmount,
          uint256 sharesAmount, uint256 a, uint256 b, bool signOfb, uint256 k, bool signOfk)
          internal view {
152       _isCreatedVerification(creatorId);
153       require(sharesAmount > 0, "incorrect sharesAmount");
154       require(a > 0, "incorrect curve params");
155       if(!signOfb){
156           require(b / (2 * a) < idoAmount, "incorrect curve params");
157       }
158       uint256 result = calculate(
159           calculate(a * idoAmount * idoAmount, b * idoAmount, signOfb),
160           k,
161           signOfk
162       );
163       require(result >= idoPrice, "incorrect curve params");
164   }
```

Listing 2.4: ProfileMarketV1.sol

Impact N/A

Suggestion Add the sanity check on `sharesAmount` accordingly.

2.2.3 Remove unused contract

Status Confirmed

Introduced by `Version 1`

Description The `Rescuable` contract in the access folder is unused in the current project implementation and can be removed from the code repository.

Impact N/A

Suggestion Remove the `Rescuable` contract.

2.2.4 Fix typo

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The “newBasURI” on Line 122 and 123 should be “newBaseURI”.

```
122    function setBaseURI(string memory newBasURI) external onlyOwner {
123        baseURI = newBasURI;
124    }
```

Listing 2.5: FarcasterKey.sol

Impact N/A

Suggestion Fix the typo.