Security Audit Report for Melos DAO Contract

Date: June 03, 2022

Version: 1.0

Contact: contact@blocksec.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.06.03 14:52:42 +08'00'

Contents

Report Manifest

Item	Description
Client	Melos Finance
Target	Melos DAO Contract

Version History

Version	Date	Description
1.0	June 03, 2022	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 smart contracts audited are in the following Github repository [1]. This report only covers the audit result of two contracts, including `vMelosStaking.sol` and `vMelosRewards.sol`. Other contracts are **NOT** covered in this audit report. Specifically, the basic functionality is to stake the Melos token into the `vMelosStaking` contract to get rewards according to the APY in different pools. The reward can only be withdrawn after the `rewardTime` configured in the `vMelosRewards` contract. The staked Melos token can be withdrawn when the pool is ended.

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 code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project		Commit SHA
Melos DAO Contract	`Version 1`	`fe3a8aafbea6cf1c22c4bc95fe27bffaf8cc6c8b`
	`Version 2`	`4180453aad6af248f9f2d200c0e2948efb75e365`

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 assets. Any entity should not rely on this report to decide to buy or sell any token, product, service, or asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any specific 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/Melos-Finance/melos-dao

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, 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 the 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 issue will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The issue has been received by the client, but not confirmed yet.
- **Confirmed** The issue has been recognized by the client, but not fixed yet.
- **Fixed** The issue 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 **one** potential issue. We have **two** recommendations.

- High Risk: 0
- Medium Risk: 0
- Low Risk: 1
- Recommendations: 2

ID	Severity	Description	Category	Status
1	Low	Some functions lack the check of paused status	Software Security	Fixed
2	-	Ensure the security of the private key of the owner	Recommendation	
3	-	Use memory instead of storage to save gas	Recommendation	Fixed

The details are provided in the following sections.

2.1 Software Security

2.1.1 Some functions lack the check of paused status

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract will be paused when something emergent is happening. In this case, functions that can move user's tokens/rewards should not function. However, we find `withdraw`, `redeposit` and `claimRewards` do not check the paused status.

Impact NA

Suggestion Add the check.

Feedback from the Project In our design, we do not limit the functionality of `withdraw` and `claimRewards` when the contract is paused. Users can still withdraw tokens from the contract. However, we do add a check in `redeposit` since users are not allowed to stake the tokens into the contract when it's paused.

2.2 Additional Recommendation

2.2.1 Ensure the security of the private key of the owner

Description Since the contract owner can set various kinds of important parameters (e.g., the rewardTime), it is critical to ensure the security of the owner's private key. For instance, the multisig wallet can be used for the owner, and the hardware-based private key protection schema (e.g., TEE based solution) can be leveraged.

Impact N/A

Suggestion Ensure the security of the private key of the contract owner.

2.2.2 Use memory instead of storage to save gas

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `getAverageLockTime` uses `storage` to store temporary variables. Memory can be used to save gas.

Impact N/A

Suggestion Use `memory` instead of `storage`.