BLOCKSEC

Security Audit Report for Staking Contracts

Date: April 7, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**
1.1 About the Audit Target 1
1.2 Disclaimer 2
1.3 Procedure of Auditing 2
1.3.1 Security Issues 2
1.3.2 Additional Recommendation 3
1.4 Security Model 3

Chapter 2 Findings **5**
2.1 Recommendation 5
2.1.1 Remove redundant code 5
2.2 Note 5
2.2.1 Ensure proper integration of the attestation mechanism 5

Report Manifest

Item	Description
Client	MegaETH
Target	Staking Contracts

Version History

Version	Date	Description
1.0	April 7, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-04-20 09:16:26+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 the Audit Target

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

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of Staking Contracts of MegaETH.

The Staking Contracts of MegaETH is a KPI-based staking system for the MEGA token on the MegaETH chain. Users can stake MEGA tokens and earn MEGA tokens based on achieved KPI milestones (i.e., tranches). Each tranche is created with a reward token and maturity curves that scale rewards according to the staking duration. This update introduces three key features: (1) a pending reward mechanism, (2) integration with Predicate[2] attestations, and (3) a linear maturity curve.

Note this audit only focuses on the smart contracts in the following directories/files. Code prior to and including the baseline version 0, where applicable, is outside the scope of this audit and assumes to be reliable and secure.

- src/abstracts/MegaStakingCore.sol
- src/abstracts/MegaStakingKPI.sol
- src/abstracts/MegaStakingLocked.sol
- src/abstracts/MegaStakingStake.sol
- src/abstracts/MegaStakingView.sol
- src/interfaces/IError.sol
- src/lib/MaturityCurveLib.sol
- src/MegaStaking.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

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. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

[1] https://github.com/megaeth-labs/mega-staking-contract

[2] https://github.com/predicatelabs/predicate-contracts

Project	Version	Commit Hash
mega-staking-contract	Version 0	37871116b9090b888cd13ec52bc8b0ce4c729347
	Version 1	138ec77a1ebfa6208846755c2667080616a076cc
	Version 2	c8ace833af6b246f3cbbf13d1678392b1ff010a4

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 Target, 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 the Target.

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.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target 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 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)

- Untrusted external call and control flow
- Exception handling
- Data handling and flow
- Events operation
- Error-prone randomness
- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism
- Economic and incentive impact

1.3.2 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 [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**,

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

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

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 five 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.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we have **one** recommendation and **one** note.

- Recommendation: 1
- Note: 1

ID	Severity	Description	Category	Status
1	-	Remove redundant code	Recommendation	Fixed
2	-	Ensure proper integration of the attestation mechanism	Note	-

The details are provided in the following sections.

2.1 Recommendation

2.1.1 Remove redundant code

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There are several redundant variables and errors. It is recommended to remove them for better code readability and gas optimization. Specifically, the following code should be removed or revised.

1. Redundant value assignments. The existing variable `stakeInfo` can be used directly.

```
374       // If stake was fully unstaked (amount = 0), delete the stake record
375       UserStakeInfo storage stakeInfoForCleanup = $.userStakes[msg.sender][_stakeId];
```

Listing 2.1: src/abstracts/MegaStakingStake.sol

2. Redundant errors.

```
19    /// @notice Thrown when an index is out of bounds
20    error IndexOutOfBounds();
```

Listing 2.2: src/interfaces/IError.sol

Suggestion Remove the redundant code.

2.2 Note

2.2.1 Ensure proper integration of the attestation mechanism

Introduced by `Version 1`

Description In the project, an attestation mechanism is introduced for claim functions. The project must ensure that all related dependencies (e.g., `DEFAULT_PREDICATE_REGISTRY`) are secure and that all on-chain registrations (e.g., policy and registry) are correctly configured.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS