Making Web3 Space Safer for Everyone

KALOS

Brick Towers Ethereum Validator Fee Splitter

Security Assessment

Published on : 24 Jan. 2024
Version v1.0

Security Report Published by KALOS

v1.0 24 Jan. 2024

Auditor: Jade

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	-	-	-	-
High	-	-	-	-
Medium	-	-	-	-
Low	-	-	-	-
Tips	-	-	-	-

TABLE OF CONTENTS

ABOUT US

Making Web3 Space Safer for Everyone

Pioneering a safer Web3 space since 2018, KALOS proudly won 2nd place in the Paradigm CTF 2023. As a leader in the global blockchain industry, we unite the finest in Web3 security expertise.

Our team consists of top security researchers with expertise in blockchain/smart contracts and experience in bounty hunting. Specializing in the audit of mainnets, DeFi protocols, bridges, and the zkEVM protocol, KALOS has successfully safeguarded billions in crypto assets.

Supported by grants from the Ethereum Foundation and the Community Fund, we are dedicated to innovating and enhancing Web3 security, ensuring that our clients' digital assets are securely protected in the highly volatile and ever-evolving Web3 landscape.

Inquiries: audit@kalos.xyz
Website: https://kalos.xyz

Executive Summary

Purpose of this report

This report was prepared to audit the security of the ValidatorFeeSplitter contracts developed by the Brick Towers team. KALOS conducted the audit focusing on whether the system created by the Brick Towers team is soundly implemented and designed as specified in the materials, in addition to the safety and security of the ValidatorFeeSplitter contract.
In detail, we have focused on the following

- Assets drainage in ValidatorFeeSplitter Contract
- Assets freezing in ValidatorFeeSplitter Contract
- Side Effects Due to Node Operation Characteristics

Codebase Submitted for the Audit

The code used in this Audit can be found on GitHub (https://github.com/bricktowers/ethereum-contracts).
The last commit of the code used for this Audit is “33bc4e3960f59b73c099d5dd1f68bba814c4f706”.

Audit Timeline

Date	Event
2024/01/23	Audit Initiation
2024/01/24	Delivery of v1.0 report.

Findings

No security issues have been discovered in the project.

OVERVIEW

Protocol overview

• **ValidatorFeeSpliter**

The `ValidatorFeeSplitter` contract is specifically crafted for the efficient distribution of accumulated fees between predetermined entities, such as the `withdrawalAddress` and the `operatorAddress`. These addresses are set as immutable and payable, providing a secure and unchangeable destination for the allocated funds. In this contract, the distribution of funds is automated, directing 30% of the contract's balance to the operator address and the remaining 70% to the withdrawal address. The `distributeBalance` function, integral to this process, is automatically triggered upon receiving Ether, which is particularly crucial for the distribution of fees related to MEV. However, in non-MEV scenarios, this function requires manual invocation to distribute the accumulated fees.

• **ValidatorFeeSpliterVAT**

Expanding on the `ValidatorFeeSplitter` Contract, the `ValidatorFeeSplitterVAT` contract integrates VAT distribution, which is a crucial element for managing specific tax obligations. In this enhanced version, an additional immutable payable address, known as the `vatAddress`, is incorporated for VAT purposes. During the fee distribution process, 8.1% of the total fees, not just the operator's share, are specifically allocated to this `vatAddress`. This mechanism is especially advantageous for Brick Towers clients in Switzerland, as it facilitates automated VAT payments on fees directly from the accumulated fees. The 8.1% rate is selected in accordance with Swiss VAT regulations. However, recognizing the needs of a diverse client base, the contract is designed to be adaptable. For investors from other countries, the Brick Towers Team can deploy an updated version of the contract with a revised VAT rate, tailoring it to different jurisdictions' tax requirements.

Scope

contracts
├── ValidatorFeeSplitter.sol
└── ValidatorFeeSplitterVAT.sol

Access Controls

There is no access control.

FINDINGS

No security issues have been discovered in the project.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and code that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure code, correction of discovered problems and sufficient testing thereof are required.

Appendix

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	Medium	High
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	• Integer under/overflow vulnerability • floating point and rounding accuracy
Access & Privilege Control	• Manager functions for emergency handle • Crucial function and data access • Count of calling important task, contract state change, intentional task delay
Denial of Service	• Unexpected revert handling • Gas limit excess due to unpredictable implementation
Miner Manipulation	• Dependency on the block number or timestamp. • Frontrunning
Reentrancy	•Proper use of Check-Effect-Interact pattern. •Prevention of state change after external call • Error handling and logging.
Low-level Call	• Code injection using delegatecall • Inappropriate use of assembly code
Off-standard	• Deviate from standards that can be an obstacle of interoperability.
Input Validation	• Lack of validation on inputs.
Logic Error/Bug	• Unintended execution leads to error.
Documentation	•Coherency between the documented spec and implementation
Visibility	• Variable and function visibility setting
Incorrect Interface	• Contract interface is properly implemented on code.

End of Document