

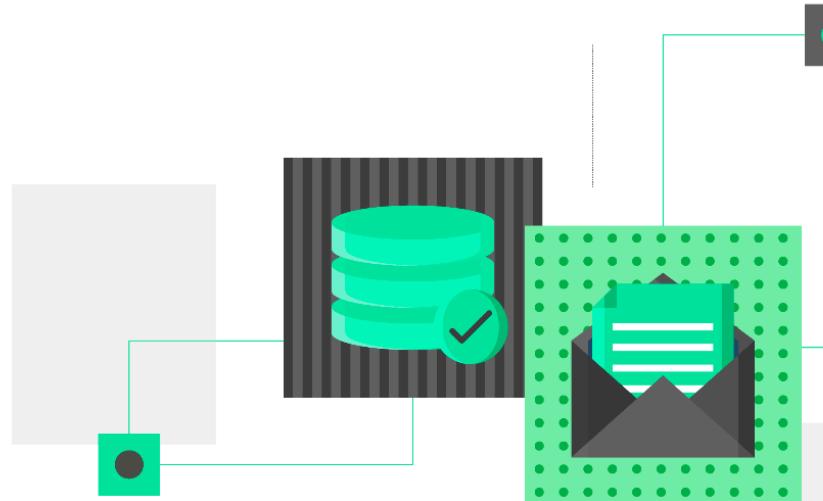
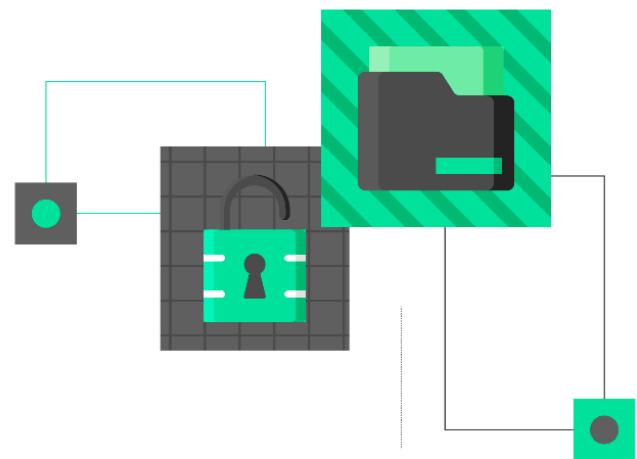


Mantis Labs

Smart Contract Audit

TokenSwap.sol

Dec 2024



Contents

Disclaimer	3
Audit Process & Methodology	4
Audit Purpose	5
Contract Details	5
Security Level Reference	6
Finding Overview	7
Identified Issues and Resolution	7
Detailed Findings	8
Additional Details	9
Concluding Remarks	10



Disclaimer

This disclaimer is to inform you that the report you are reading has been prepared by Mantisec Labs for informational purposes only and should not be considered as investment advice. It is important to conduct your own independent investigation before making any decisions based on the information contained in the report. The report is provided "as is" without any warranties or conditions of any kind and Mantisec Labs excludes all representations, warranties, conditions, and other terms. Additionally, Mantisec Labs assumes no liability or responsibility for any kind of loss or damage that may result from the use of this report.

It is important to note that the analysis in the report is limited to the security of the smart contracts only and no applications or operations were reviewed. The report contains proprietary information, and Mantisec Labs holds the copyright to the text, images, photographs, and other content. If you choose to share or use any part of the report, you must provide a direct link to the original document and credit Mantisec Labs as the author.

By reading this report, you are accepting the terms of this disclaimer. If you do not agree with these terms, it is advisable to discontinue reading the report and delete any copies in your possession.



Audit Process & Methodology

The Mantisec Labs team carried out a thorough audit for the project, starting with an in-depth analysis of code design patterns. This initial step ensured the smart contract's architecture was well-structured and securely integrated with third-party smart contracts and libraries. Also, our team conducted a thorough line-by-line inspection of the smart contract, seeking out potential issues such as Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, among others.

During the Unit testing phase, we assessed the functions authored by the developer to ascertain their precise functionality. Our Automated Testing procedures leveraged proprietary tools designed in-house to spot vulnerabilities and security flaws within the Smart Contract. The code was subjected to an in-depth audit administered by an independent team of auditors, encompassing the following critical aspects:

- Scrutiny of the smart contract's structural analysis to verify its integrity.
- Extensive automated testing of the contract
- A manual line-by-line Code review, undertaken with the aim of evaluating, analyzing, and identifying potential security risks.
- An evaluation of the contract's intended behavior, encompassing a review of provided documentation to ensure the contract conformed to expectations.
- Rigorous verification of storage layout in upgradeable contracts.
- An integral component of the audit procedure involved the identification and recommendation of enhanced gas optimization techniques for the contract



Audit Purpose

Mantisec Labs was hired by the Ethernity team to review their smart contract. This audit was conducted in **Dec 2024**.

The main reasons for this review were:

- To find any possible security issues in the smart contract.
- To carefully check the logic behind the given smart contract.

This report provides valuable information for assessing the level of risk associated with this smart contract and offers suggestions on how to improve its security by addressing any identified issues.

Contract Details

Project Name	Ethernity
Contract links	https://github.com/ethernitychain/epictoken/blob/governor/contracts/TokenSwap.sol
Language	Solidity
Type	ERC20

Security Level Reference

Each problem identified in this report has been categorized into one of the following severity levels:

- **Critical:** Vulnerabilities that present an immediate and serious threat to system or data integrity, demanding urgent action.
- **High:** Significant risks that have the potential to cause major security breaches or loss of functionality.
- **Medium:** Issues that moderately affect system performance or security and require timely resolution.
- **Low:** Low-risk concerns primarily related to optimization and code quality, with minimal direct impact on system security.
- **Informational:** Observations or recommendations that do not pose any direct risk but provide insights for potential improvements or best practices.

Severity	Score
Critical	4-5
High	3-4
Medium	2-3
Low	1-2
Informational	0-1



Findings Overview

Contract Name:

- **TokenSwap.sol**

Critical	High	Medium	Low	Informational
0	0	1	1	0

Identified Issues and Resolutions

Issue	Severity	Fix Date
M01 -Over Complicated Initialization of incomingToken and outgoingToken Variables	Medium (2)	21-Dec-2024
L01 -Use Ownable2Step rather than Ownable	Low (1.2)	21-Dec-2024

Detailed Findings

M01-Over Complicated Initialization of incomingToken and outgoingToken Variables

The current design of the contract introduces unnecessary complexity in initializing the `incomingToken` and `outgoingToken` variables. These variables are only set once and by the owner, making them ideal candidates for constructor-based initialization and immutability. By deferring their assignment to separate functions, the implementation increases the likelihood of errors, delays, and potential misuse during the deployment phase.

To reduce complexity and streamline deployment:

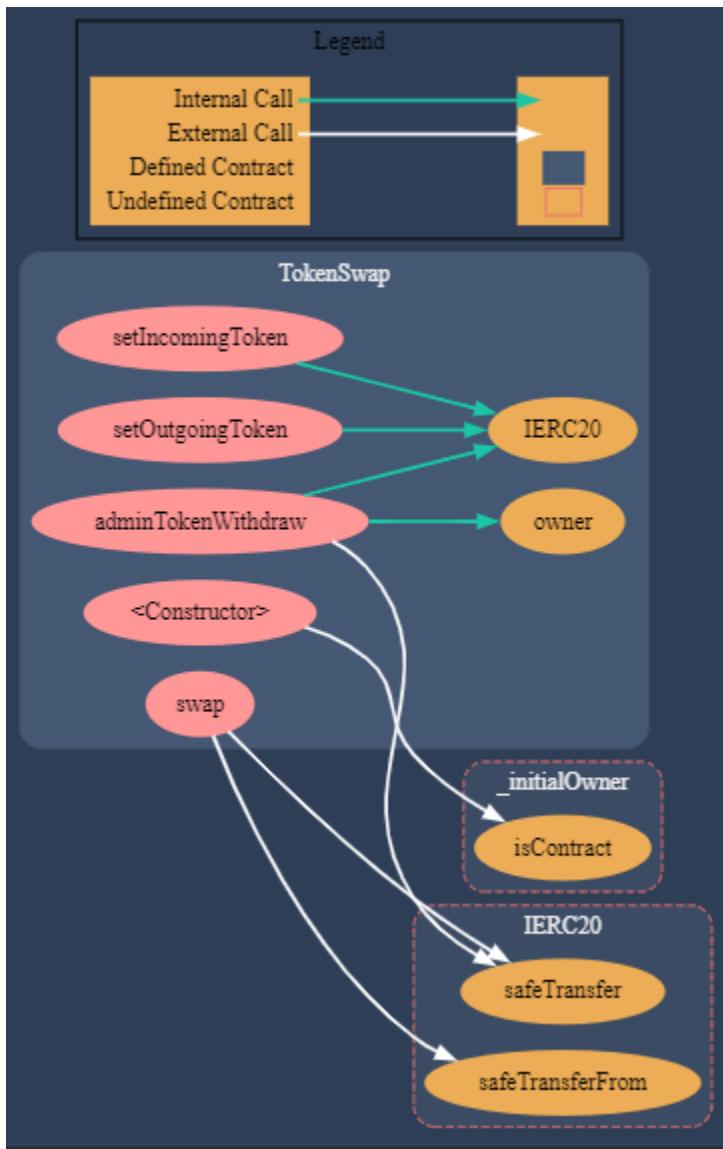
1. Include the `incomingToken` and `outgoingToken` addresses as constructor parameters.
2. Mark them as `immutable` to eliminate any risk of post-deployment modification and optimize gas usage.

The contract will be fully configured and operational immediately upon deployment. This approach minimizes errors, ensures completeness, and simplifies deployment scripts.

L02 - Use Ownable2Step rather than Ownable

`Ownable2Step` and `Ownable2StepUpgradeable` prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner's permissions actively accept via a contract call of its own.

Additional Details:





Mantisec Labs

Concluding Remarks

To wrap it up, this audit has given us a good look at the contract's security and functionality.

Our auditors confirmed that all the issues are now resolved by the developers.