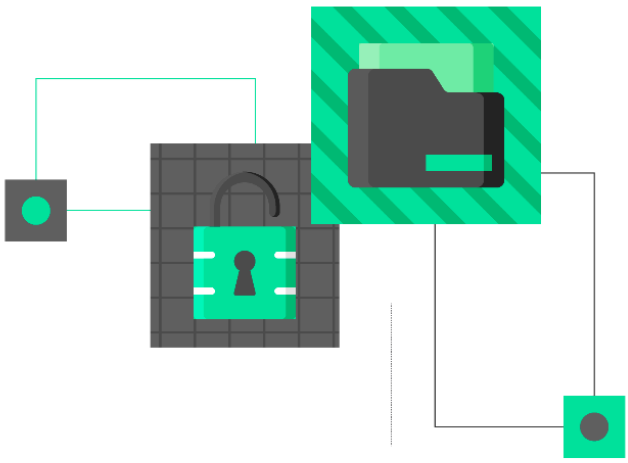
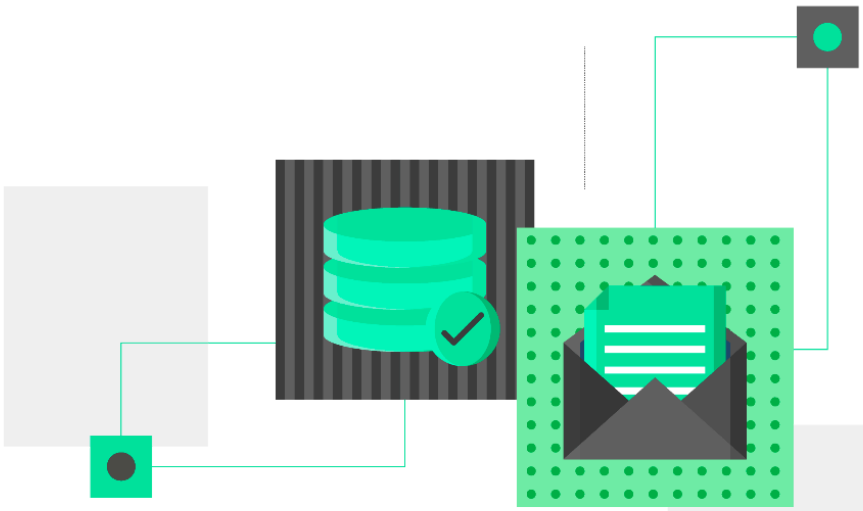
Mantisec Labs

Smart Contract Audit



DimAI



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
Concluding Remarks	12

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	DimAI
Contract links	https://github.com/X2074/DimAIContract/blob/main/contracts/DimAILogic.sol https://github.com/X2074/DimAIContract/blob/main/contracts/DimaiNFT.sol https://github.com/X2074/DimAIContract/blob/main/contracts/DimaiToken.sol
Language	Solidity

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:

- **DimAILogic.sol**
- **DimaiNFT.sol**
- **DimaiToken.sol**

Critical	High	Medium	Low	Informational
0	3	0	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

Contract: DimAILogic.sol

H01-Incorrect Fee Calculation in useDimaiByMeer Function

Description:

There is an issue with the useDimaiByMeer function,In the DimAILogic contract.Which allows users to use the Dimai service by providing a payment and other parameters.Although, the function does not take into consideration for the case when the times parameter is set to zero. This omission leads to an incorrect fee calculation and potentially results in unrequired behavior.

Impact:

The lack of handling for the times parameter being zero can have the following impacts:

Incorrect Fee Calculation: When times is set to zero, the totalFee calculation becomes zero. This means that no fee is charged to the user, regardless of the payment provided. Consequently, the fee calculation is inaccurate and does not reflect the intended charging mechanism.

Payment Validation Bypass: Since the require statement require(msg.value >= totalFee, "Incorrect fee amount"); always evaluates to true when totalFee is zero, users can submit payments lower than the intended fee or even no payment at all. This bypasses the expected payment validation, potentially allowing unauthorized or free usage of the Dimai service.

Recommendation:

Validate the times parameter: Add a validation check at the beginning of the function to ensure that times is greater than zero. If times is zero, revert the transaction with an appropriate error message indicating that the fee multiplier must be greater than zero.

H02- Potential Reentrancy Vulnerability

Description:

The buyDIM and swapDIM functions do not explicitly protect against reentrancy attacks, which leaves it potentially vulnerable to such attacks. Reentrancy attacks occur when an external contract is called during the execution of a function, allowing the attacker to reenter the function before it completes and potentially manipulate the contract's state.

Potential Exploitation Scenario:

An attacker deploys a malicious contract that implements a function with a fallback or receive function. The attacker initiates a transaction to the buyDIM or swapDIM functions from their malicious contract. During the execution of the buyDIM or swapDIM function, the malicious contract's fallback or receive function is invoked, allowing the attacker to reenter the buyDIM or swapDIM function while it is still in progress. The attacker can perform undesired actions, such as calling other functions in the contract, manipulating data, or even draining the contract's Ether balance.

Recommended Mitigation:

To mitigate the potential reentrancy vulnerability, Implement a reentrancy guard modifier to prevent reentrant calls. This can be achieved by setting a boolean variable to indicate whether the function is currently being executed and reverting if an attempt is made to reenter the function before completion.

L01- Ambiguity in Event Emission for BuyDIM Transactions

Description:

In the DimAILogic contract, there is an ambiguity in the event emission related to the buyDIM transactions. The contract includes two functions, buyDIM and swapDIM, which enable users to purchase DIM tokens. However, both functions emit the same event, BuyDIM, without providing clear distinctions between the types of transactions. This lack of clarity can lead to confusion, making it difficult to differentiate between direct purchases and those executed based on a specific ratio. Consequently, this ambiguity hampers the accurate tracking, analysis, and interpretation of DIM token purchases within the contract.

Impact:

The confusion arising from the identical event emission can have several negative consequences:

Ambiguity in Event Logs: Emitting the same event for different types of transactions can make it challenging to differentiate between transactions executed via the buyDIM function and those executed via the swapDIM function. This ambiguity can hinder the ability to accurately analyze and interpret event logs. User Experience: From a user's perspective, it may be confusing to receive event notifications that do not clearly indicate the type of transaction they initiated. This lack of clarity may lead to frustration and a suboptimal user experience.

Recommendation:

To address the confusion arising from the event emission, We recommend creating a new event specifically for the swapDIM function. This will provide a clear distinction between the two types of transactions and enhance the clarity and understanding of the emitted events.

Contract: DimaiNFT.sol

H01-Lack of Access Control in the freeMint Function

Issue Description:

In the DimaiNFT contract, the freeMint function lacks proper access control, allowing anyone to call the function and mint tokens without any associated cost. This absence of access control poses a security risk and can lead to unauthorized minting of tokens.

Impact:

The lack of access control in the freeMint function can have various negative impacts:

Unauthorized Token Minting: Any user, including potential attackers, can freely call the freeMint function and mint tokens without incurring any cost. This unauthorized minting can lead to the creation of an excessive number of tokens or tokens being minted by malicious actors, potentially devaluing the token economy or causing disruption to the intended system.

Misuse of Resources: The unrestricted access to mint tokens can lead to the misuse of computational resources, as attackers or unintended users can overload the system by repeatedly calling the freeMint function and minting a large number of tokens without incurring any associated cost.

Recommendation:

Apply an access control modifier, such as onlyOwner, to the freeMint function. This ensures that only the contract owner or a designated authority can call the function and mint tokens. The onlyOwner modifier restricts access to trusted addresses and provides an additional layer of security.

Contract: DimaiToken.sol

No issues were found

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.