Audit Report

Desme Staking Contract

Smart Contract final audit report

Mon 27 Nov, 2023

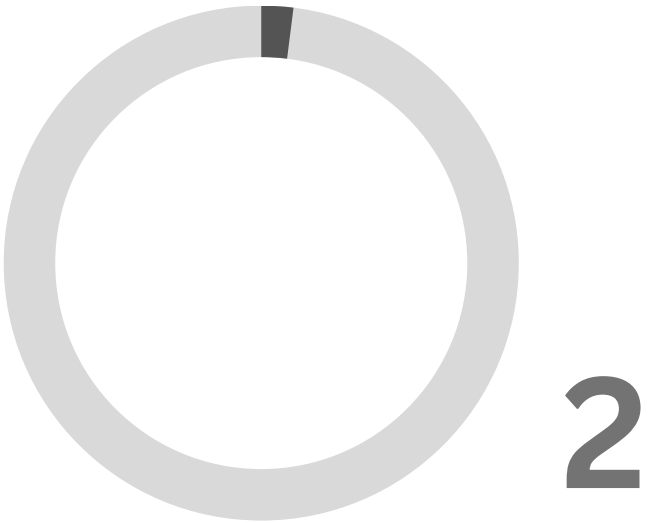
Smart Contract Initial Audit Report

Executive Summary

Audit Scope: The scope of this audit was to analyze the codebase for quality, security, and correctness.

Sourcecode: The code was given in the attached source file.

Findings Breakdown



Criticial	0
Medium	0
Minor / Informative	2

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception disorder
- Gasless send
- Byte array
- Transfer forwards all gas
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption

Tools and Platforms used for Audit Remix IDE, Hardhat, Solhint, Mythril, Slither, Solidity statistic analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Contract Structure

The contract is named Desme and it inherits from the Ownable contract. It is designed for staking tokens, allowing users to stake and unstake tokens while earning rewards. The contract keeps track of various details such as user staking information, staking counts, rewards distribution, and more.

Key Components and Functions:

State Variables:

- _users: An array containing addresses of users who have participated in staking.
- _totalStakingRewardsDistributed: Total staking rewards distributed by the owner.
- _stakingsCount: Counter for the total number of stakings.
- _calStakingReward: A variable to calculate the total staking rewards.
- _valueStaked: Total value of tokens staked across all users.
- _lastTimeRewardDistributed: Timestamp of the last time staking rewards were distributed.
- _carryForwardBalance: Balance carried forward for future reward distributions.
- _tokenAddress: Address of the ERC20 token used for staking.

Booleans:

- _noReentrancy: Used as a flag to prevent reentrancy attacks.
- _isPaused: A flag indicating whether the contract is currently paused.

Mappings:

- _mappingAccounts: Maps user addresses to their corresponding StructAccount data.
- _mappingStakings: Maps staking IDs to their corresponding StructStaking data.

Events:

- Various events (SelfAddressUpdated, Stake, UnStake, ClaimedStakingReward, DistributeStakingReward, ContractReset, ContractStatus) are emitted to log important contract actions.

Modifiers:

- noReentrancy: A modifier to prevent reentrancy attacks.
- whenNoPaused: A modifier to ensure that certain functions can only be executed when the contract is not paused.

Receive Function:

- receive() external payable {}: Allows the contract to receive Ether.

Constructor:

- constructor(address initialOwner): Initializes the contract with an initial owner and sets _lastTimeRewardDistributed to the current timestamp.

Internal Functions:

- _updateUserAddress: Updates the user's address in the contract.
- _updateCalStakingReward: Updates the calculated staking reward for a staking account.
- _stake: Handles the staking process.
- _getStakingRewardsById: Retrieves staking rewards for a specific staking ID.
- _getUserAllStakingRewards: Retrieves total staking rewards for a user.
- _claimUserStakingReward: Claims staking rewards for a user.
- _unStake: Handles the unstaking process.
- _toTokens, _toWei: Conversion functions between Wei and tokens.

External Functions:

- stake: Allows users to stake tokens.
- getStakingRewardsById: Retrieves staking rewards for a specific staking ID.
- getUserStakingRewards: Retrieves total staking rewards for a user.
- claimStakingReward: Claims staking rewards for a user.
- unStake: Allows users to unstake tokens.
- distributeStakingRewards: Allows the owner to distribute staking rewards.
- getUsersParticipatedList: Retrieves a list of user addresses.
- getUserShare: Retrieves the percentage share of a user's staked value.
- getContractDefault, setTokenAddress, getContractAnalytics, adjustCalRewards, getUserAccount, getStakingById, withdrawETH, isPaused, pause, unPause: Various utility and information retrieval functions.

High Severity Issues

No High Severity Issues detected.

MediumSeverity Issues

No Medium Severity Issues detected.

Low Severity / Informational Issues

1. Conformance to Solidity Naming Conventions

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with. The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).

2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).

3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE). 4. Use indentation to improve readability and structure.

5. Use spaces between operators and after commas.

6. Use comments to explain the purpose and behavior of the code.

7. Keep lines short (around 120 characters) to improve readability

Low Severity / Informational Issues

1. Conformance to Solidity Naming Conventions

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with. The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).

2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).

3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE). 4. Use indentation to improve readability and structure.

5. Use spaces between operators and after commas.

6. Use comments to explain the purpose and behavior of the code.

7. Keep lines short (around 120 characters) to improve readability

```
    uint256 private _totalStakingRewardsDistributed;

    uint256 private _lastTimeRewardDistributed;
    uint256 private _carryForwardBalance;
```

Low Severity / Informational Issues

2. Missing Events Arithmetic

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task. It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
function setTokenAddress(address tokenAddress_) external onlyOwner {
        _tokenAddress = tokenAddress_;
    }
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

Audit Summary Report

This summary report provides an overview of the audit conducted on the Desme smart contract. The audit focused on potential security vulnerabilities, code quality, and adherence to best practices. The audit covered the contract's functionality, including staking, unstaking, claiming rewards, and distributing staking rewards.

Findings:

1. Critical Findings:
   - No critical security issues were identified.
2. Major Findings:
   - No critical security issues were identified.
3. Moderate Findings:
   - The use of the block.timestamp for time calculations might be susceptible to manipulation. Consider using block.number for greater security.
   - The contract's logic for staking rewards distribution is complex and may benefit from additional comments to improve clarity.
4. Minor Findings:
   - The contract could include more comprehensive error messages to assist users in understanding transaction failures.
   - It's advisable to use the SafeMath library for arithmetic operations to prevent overflows and underflows.
   - Consider adding an event to log unsuccessful token transfers in the stake and unstake functions.

Recommendations:

1. Enhancements:
   - Consider adding more inline comments to improve code readability and maintainability.
   - Implement additional checks to ensure the safety of the time-related calculations.
2. Gas Optimization:
   - Evaluate the gas costs of various functions and optimize where possible.
   - Consider using view or pure modifiers for functions that don't modify the state.
3. Code Quality:
   - Utilize the SafeMath library for secure arithmetic operations.
   - Improve error messages to provide users with better information on transaction failures.
4. Security Best Practices:
   - Implement standard security best practices, such as using access control modifiers.

Overall Assessment:

The Desme contract shows a strong commitment to security, with no critical issues identified. Addressing the recommended enhancements and optimizations will further strengthen the contract's robustness and usability.

This audit is not an exhaustive review, and it is recommended to conduct further testing and peer reviews before deployment.

Note: This report is based on the analysis conducted at the time of the audit and may not reflect subsequent changes to the contract.

Disclaimer

The information contained in this report is not intended to serve as investment, financial, or trading advice and should not be construed as such. No part of this document should be transmitted, disclosed, referenced, or relied upon by any person other than the Company without prior written consent. The report does not endorse or disapprove of any specific project or team, and it does not provide an indication of the economic value of any "product" or "asset" created by a team or project that undergoes a security assessment.

There is no warranty or guarantee regarding the absolute bug-free nature of the analyzed technology, and the report does not offer insights into the business, business model, or legal compliance of the technology's proprietors. It should not be used as a basis for investment decisions or involvement with any particular project. The report represents a comprehensive assessment process designed to assist customers in enhancing code quality while mitigating risks associated with cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets inherently involve ongoing high-level risks. Each company and individual is responsible for their due diligence and continuous security practices. The goal is to contribute to the reduction of attack vectors and the variability associated with rapidly evolving technologies. However, there are no claims regarding the guarantee of security or functionality of the technology analyzed.

The assessment services are subject to dependencies and are continually evolving. Your access and/or use of services, reports, and materials are at your sole risk, on an "as-is where-is" and "as-available" basis. Cryptographic tokens are emerging technologies carrying inherent technical risks and uncertainties. Assessment reports may include false positives, false negatives, and other unpredictable results. The services may access and depend upon multiple layers of third-party involvement.