QuillAudits

AUDIT REPORT

December 2025

For

mob

Table of Content

Executive Summary

Project Name	Mopay
Protocol Type	ERC20 Token
Overview	MobToken is a standard ERC20 token implementation built on OpenZeppelin's audited contracts. The contract allows for minting a fixed total supply of tokens to the deployer's address during contract deployment. The token follows the ERC20 includes basic functionality for transfers, approvals, and balance tracking.
Audit Scope	The scope of this Audit was to analyze the Mopay Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/Mobee-Exchange/mob-token/blob/main/src/MobToken.sol
Contracts in Scope	MobToken.sol
Commit Hash	e0b0c0462ee54bd9853178ca9b810540ba6135bd
Branch	Main
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing 28th
Review 1	October 2025
Updated Code Received	15th December 2025
Review 2	15th December 2025
Fixed In	4d17b9622b1b7c4cbf14e2dc5a649f051a52c9bb

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity

02

Total Issues

🟥	Critical	0(0.0%)
🟥	High	0(0.0%)
🟧	Medium	0(0.0%)
🟨	Low	0(0.0%)
🟪	Informational	2 (100%)

Severity

Issues

	🟥 Critical	🟥 High	🟧 Medium	🟨 Low	🟪 Informational
Open	0	0	0	0	0
Acknowledged	0	0	0	0	0
Partially Resolved	0	0	0	0	0
Resolved	0	0	0	0	**2**

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Missing Input Validation for Total Supply	Informational	Resolved
2	Floating Pragma Version	Informational	Resolved

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls

- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw
- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ 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
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC standards
- 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, Foundry, Solhint, Mythril, Slither, Solidity Static Analysis.

Types of Severity

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

Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

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.

Severity Matrix

Impact

Likelihood	🟥 High	🟧 Medium	🟨 Low
🟥 High	Critical	High	Medium
🟧 Medium	High	Medium	Low
🟨 Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Informational Issues

Missing Input Validation for Total Supply

Resolved

Path

MobToken.sol

Function

`constructor`

Description

The constructor accepts a totalSupply parameter without validating that it's greater than zero. This allows deployment of a token contract with 0 total supply, which would create a useless token that cannot be used for any meaningful purpose.

Impact

Wastes gas for deployment

Likelihood

Low

Recommendation

```
constructor(
    string memory name,
    string memory symbol,
    uint256 totalSupply
) ERC20(name, symbol) {
    require(totalSupply > 0, "Total supply must be greater than zero");
    _mint(msg.sender, totalSupply * 10 ** decimals());
}
```

Floating Pragma Version

Resolved

Path

MobToken.sol

Description

The contract uses a floating pragma ^0.8.20, which allows compilation with any version from 0.8.20 up to (but not including) 0.9.0. Different compiler versions may produce different bytecode and could introduce:

- Unexpected behavior from compiler bugs
- Inconsistent deployments across different environments
- Potential security vulnerabilities discovered in newer compiler versions
- Difficulty in reproducing exact deployment conditions

Impact

- Different teams may compile with different versions, leading to inconsistencies
- Future compiler versions might introduce breaking changes or bugs
- Harder to audit and verify deployed bytecode

Likelihood

Low

Recommendation

Lock the pragma to a specific, well-tested compiler version

Functional Tests

Some of the tests performed are mentioned below:

- ✓ Token Deployment: Successfully deploys with correct name, symbol, and total supply
- ✓ Minting: Correctly mints total supply to deployer address during construction
- ✓ Transfer: Standard ERC20 transfer functionality works as expected
- ✓ Approval & TransferFrom: Allowance mechanism functions correctly
- ✓ Balance Queries: balanceOf returns accurate token balances
- ✓ Decimal Precision: Uses standard decimals for calculations
- ✓ Overflow Protection: Solidity 0.8.20 arithmetic overflow checks function properly
- ✓ Basic Transfer: Standard ERC20 transfer functionality works correctly between addresses
- ✓ Transfer Event Emission: Transfer events are properly emitted with correct from, to, and amount parameters
- ✓ Zero Amount Transfers: Transfers of 0 tokens execute without errors
- ✓ Maximum Supply Minting: Constructor handles maximum possible supply values with overflow protection

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Threat Model

Contract	Function	Threats
MobToken	constructor	Deployer could accidentally pass totalSupply = 0, creating a token contract with no tokens minted. This results in a useless contract that wastes deployment gas and cannot serve its intended purpose.
MobToken	constructor	Malicious or accidental deployment with extremely large totalSupply values could cause arithmetic overflow during totalSupply * 10 ** decimals() calculation.
MobToken	transfer	If token contract made external calls during transfer, malicious contracts could re-enter and manipulate balances. Classic attack vector in poorly designed tokens.
MobToken	approve	Attacker monitoring mempool sees approve transaction changing allowance from X to Y. They front-run with transferFrom(X), then call transferFrom(Y) after, spending X+Y total.
MobToken	N/A	100% of token supply is minted to msg.sender (deployer) at deployment, creating extreme centralization where one address controls all tokens initially.
MobToken	N/A	Tokens cannot be removed from circulation permanently. Once minted, the total supply remains constant forever with no way to reduce it.

Closing Summary

In this report, we have considered the security of Mopay. We performed our audit according to the procedure described above.

No critical issues in Mopay token, just 2 issues of Informational severity were found. Mopay team resolved both the issues mentioned above

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With seven years of expertise, we've secured over 1400 projects globally, averting over $3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

QuillAudits

7+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

December 2025

For

mob

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com