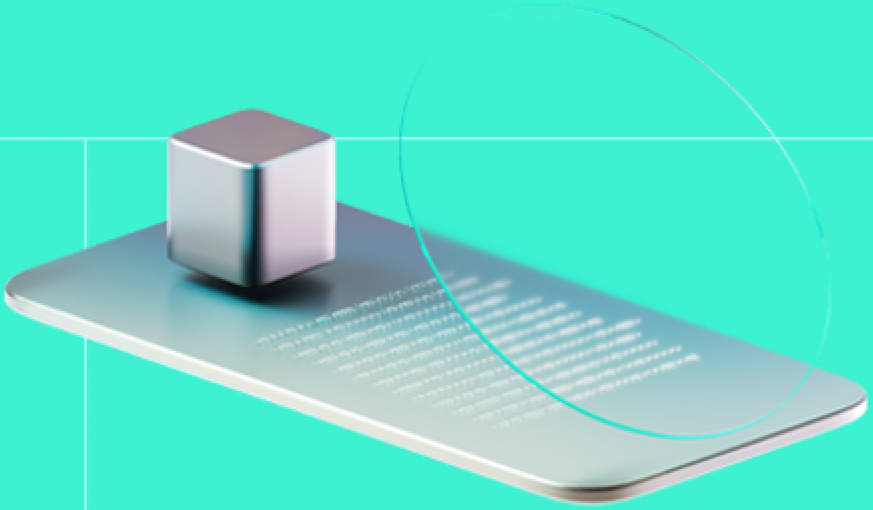
Smart Contract Code Review And Security Analysis Report



Customer: Multipool

Date: 10/05/2024

We express our gratitude to the Multipool team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

The Multipool is a basic ERC20 token contract with mintable and burnable features.

Platform: EVM

Language: Solidity

Tags: ERC20, ERC20Burnable

Timeline: 02/05/2024 - 03/05/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/Multipool-Finance/mul-token
Commit	f44dbbe

Audit Summary

10/10	10/10	100%	10/10
Security score	Code quality score	Test coverage	Documentation quality score

Total 10/10

The system users should acknowledge all the risks summed up in the risks section of the report

0	0	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	0
Low	0

Document

Name	Smart Contract Code Review and Security Analysis Report for Multipool
Audited By	Eren Gonen
Approved By	Ataberk Yavuzer
Website	https://whitepaper.multipool.finance/
Changelog	06/05/2024 - Preliminary Report
	09/05/2024 - Final Report

Table of Contents

System Overview

Multipool is a ERC20 token with the following contract:

Multipool — simple ERC-20 token that mints all initial supply to a deployer with the burning feature. Additional minting is allowed.

It has the following attributes:

- Name: Multipool
- Symbol: MUL
- Decimals: 18
- Total supply: 100m tokens.

Privileged roles

- The admin of the contract can arbitrarily transfer the admin role to another address and mint 2 million tokens to their address annually.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation quality score is **10** out of **10**.

- Functional requirements are provided
- Technical description is not provided.

Code quality

The total Code quality score is **10** out of **10**.

- The development environment is configured.

Test coverage

Code coverage of the project is **100%**

- Everything covered with tests

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **0** medium, and **0** low severity issues. Out of these, **0** issues have been addressed and resolved, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **10**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- **Centralized Minting to a Single Address**: The project concentrates minting tokens in a single address, raising the risk of fund mismanagement or theft, especially if key storage security is compromised.
- **Centralized Control of Minting Process**: The token contract’s design allows for centralized control over the minting process, posing a risk of unauthorized token issuance, potentially diluting the token value and undermining trust in the project's economic governance.
- **Administrative Key Control Risks**: The digital contract architecture relies on administrative keys for critical operations. Centralized control over these keys presents a significant security risk, as compromise or misuse can lead to unauthorized actions or loss of funds.
- **Solidity Version Compatibility and Cross-Chain Deployment:** The project utilizes Solidity version 0.8.24, which includes the introduction of the PUSH0 (0×5f) opcode. This opcode is currently supported on the Ethereum mainnet but may not be universally supported across other blockchain networks. Consequently, deploying the contract on chains other than the Ethereum mainnet, such as certain Layer 2 (L2) chains or alternative networks, might lead to compatibility issues or execution errors due to the lack of support for the PUSH0 opcode. In scenarios where deployment on various chains is anticipated, selecting an appropriate Ethereum Virtual Machine (EVM) version that is widely supported across these networks is crucial to avoid potential operational disruptions or deployment failures.

Findings

Observation Details

F-2024-2048 - Single-Step Ownership Transfer - Info

Description:

The current implementation in contract includes a method **transferAdminRole()** that allows transferring the administrative rights of the contract to another address in a single step. In Ethereum and similar blockchain environments, the ownership or administrative privileges of a contract are crucial for managing its lifecycle and critical functionalities. A typo or mistake in the address during transfer could potentially lock out the original administrators permanently if the new address is incorrect or does not correspond to an actual user or contract capable of interaction.

```
function transferAdminRole(address newAdmin) public onlyOwner {
require(newAdmin != address(0), "Invalid address");
owner = newAdmin;
}
```

The current method does not ensure that the new owner address is prepared or intended to take over the role, which could lead to administrative rights being transferred to an unintended or incapable handler.

Assets:

- MUL.sol [https://github.com/Multipool-Finance/mul-token]

Status: Fixed

Recommendations

Remediation:

Consider using Ownable2Step or `Ownable2StepUpgradeable` from OpenZeppelin Contracts to enhance the security of your contract ownership management. These contracts prevent the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call. This two-step ownership transfer process adds an additional layer of security to your contract's ownership management.

Resolution:

The team fixed the issue in commit **d99c164** by implementing `Ownable2Step` contract.

F-2024-2049 - Missing Events - Info

Description:

Events for critical state changes should be emitted for tracking actions off-chain.

It was observed that events are missing in the following functions:

```
function transferAdminRole(address newAdmin) external onlyAdmin {
require(newAdmin != address(0), "New admin address cannot be the zer
o address");
admin = newAdmin;
}
```

Events are crucial for tracking changes on the blockchain, especially for actions that alter significant contract states or permissions. The absence of events in these functions means that external entities, such as user interfaces or off-chain monitoring systems, cannot effectively track these important changes.

Assets:

- MUL.sol [https://github.com/Multipool-Finance/mul-token]

Status: Fixed

Recommendations

Remediation: Consider implementing and emitting events for the necessary functions.

Resolution: The team fixed the issue in commit **d99c164** by removing the function.

F-2024-2050 - Redundant Initialization Of ´yearsMinted´ Variable - Info

Description: In the **Multipool**, specifically within the `constructor()`, `uint256 yearsMinted` variable are explicitly initialized to zero, which is redundant since `uint256` variables in Solidity are zero by default.

```
constructor() ERC20("Multipool", "MUL") {
admin = msg.sender; // Assign the admin role to the contract deploye
r
_mint(msg.sender, INITIAL_SUPPLY);
deploymentTime = block.timestamp;
yearsMinted = 0;
}
```

While this issue does not pose a direct security risk, it leads to unnecessary opcode generation during contract compilation, resulting in minimal but avoidable gas costs during contract deployment. More importantly, it may detract from the code's readability by introducing unnecessary elements that do not contribute to functionality.

Assets:

- MUL.sol [https://github.com/Multipool-Finance/mul-token]

Status: Fixed

Recommendations

Remediation: The best practice in this case is to remove the explicit initializations of the `uint256` variable. The code should leverage Solidity's default initialization to reduce verbosity and potential confusion.

Resolution: The team fixed the issue in commit **d99c164** by removing redundant initialization.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/Multipool-Finance/mul-token
Commit	f44dbbe56e30efdeb83423417b9dea5562d868b9
Whitepaper	https://whitepaper.multipool.finance/
Requirements	N/A
Technical Requirements	N/A

Contracts in Scope

./contracts/MUL.sol