Smart Contract Audit Report

Project Name: Prospera
Smart Contract Name: PROSPERA
Contract Address: 0x18F764179b89B24b05A01052f9B6bf621208bE81
Auditor: 0xGoul
Date: August 5, 2024

Introduction

This audit report has been prepared for the Prospera project to evaluate the security and functionality of the `PROSPERA` smart contract. The contract implements an ERC20 token with various features including burning, pausing, vesting, and upgradeability using UUPS proxies. The source code has been reviewed to ensure it follows best practices and is free from vulnerabilities.

Overview

The `PROSPERA` contract has the following functionalities:

- ERC20 standard token implementation.
- Burning and pausing functionalities.
- Vesting mechanism for token allocation.
- Upgradeability using UUPS proxies.
- Custom error handling for improved gas efficiency.
- Detailed event logging for critical operations.

Inheritance and Libraries

The contract inherits from several OpenZeppelin libraries to ensure standard compliance and security:

- `ERC20Upgradeable`
- `ERC20BurnableUpgradeable`
- `ERC20PausableUpgradeable`
- `OwnableUpgradeable`
- `UUPSUpgradeable`
- `ReentrancyGuardUpgradeable`

Key Components and Functionality

Blacklist Functionality

The contract includes mechanisms to blacklist addresses to prevent them from participating in token transfers.

- **Functions**: `addToBlacklist`, `removeFromBlacklist`
- **Observations**: These functions are well-implemented with appropriate error handling.

Recommendation: Ensure critical addresses (e.g., vesting contract, ICO contract) are not blacklisted inadvertently.

Vesting and ICO Mechanisms

The contract integrates vesting and ICO functionalities to restrict token transfers during specific conditions.

- **Functions**: `recordIcoCompletion`, `releaseVestedTokensForAccount`, `addAccountToVesting`, `transferICOTokens`
- **Observations**: The contract appropriately checks and handles vested tokens and ICO state.

Recommendation: Add a check to prevent the ICO state from being set to active after completion.

Token Transfer Logic

The transfer logic includes checks for blacklisted addresses and vested tokens.

- **Functions**: `_update`
- **Observations**: The `_update` function is overridden to include necessary checks during transfers.

Recommendation: Ensure that essential addresses are not blacklisted.

Upgradeable Contract

The contract uses the UUPS proxy pattern for upgradeability, with access control to ensure only the owner can authorize upgrades.

- **Function**: `_authorizeUpgrade`
- **Observations**: The function correctly restricts upgrade authorization to the owner.

Recommendation: Regularly audit and review new implementations before upgrades.

Initialization

The contract initialization sets up wallets and mints the total token supply.

- **Functions**: `initialize`, `_initializeWallets`, `_initializeContracts`
- **Observations**: Initialization is handled carefully to set up the contract state correctly.

Recommendation: Thoroughly review and test the initialization process to avoid any mistakes in the initial setup.

Security and Best Practices

OwnableUpgradeable Contract

The `OwnableUpgradeable` contract is well-implemented with standard access control mechanisms.

Initializable Contract

The `Initializable` contract includes standard logic for initialization and reinitialization.

UUPSUpgradeable Contract

The `UUPSUpgradeable` contract correctly implements the upgradeability mechanism with security checks.

ERC20Upgradeable Contract

The `ERC20Upgradeable` contract includes standard ERC20 functionality along with custom burning and pausing mechanisms.

Specific Recommendations

1. **Vesting Contract Checks**: Ensure the `vestingContract` address adheres to the `IVestingContract` interface and add unit tests for vesting scenarios.
2. **ICO Contract Integration**: Verify that the `icoContract` implements the `IPROSPERAICO` interface and add integration tests for ICO phases.
3. **Fallback and Receive Functions**: Handle unexpected calls appropriately and log them for auditing.
4. **WithdrawETH Function**: Add additional access control checks to prevent unauthorized withdrawals.

Conclusion

The `PROSPERA` smart contract is well-structured and adheres to many best practices in smart contract development. The use of OpenZeppelin libraries ensures a robust and secure foundation. Continuous testing and regular audits are recommended, especially for upgradeable contracts and complex tokenomics like vesting and ICOs.
