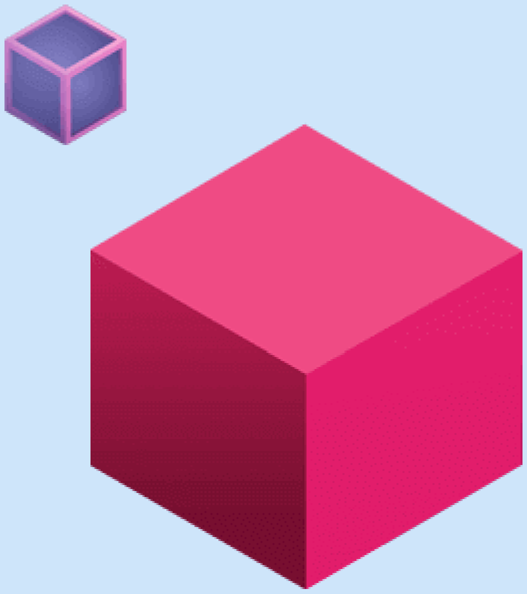
WP Smart Contracts

The Ultimate Smart Contracts Integration for WordPress



CROWDSALE, SECURITY VAULT & AIRDROP

# WP Smart Contracts

Smart Contract Audits

Sep 2023

# INTRODUCTION

At WPSmartContracts.com, our commitment to delivering solutions to our users and the demands of a dynamic market has been our driving force. We are proud of our journey, which has led us to develop a WordPress plugin that adapts to the evolving needs of our user community.

With the release of WPSmartContracts 2.0, we have marked a significant milestone in the evolution of our WP Smart Contracts project. In our pursuit of excellence, during September 2023, we embarked on a series of audits for the third batch of smart contracts within WPSmartContracts. These audits encompassed critical contracts, including:

- **Bubblegum Crowdsale**
- **Coconut Safe Vault**
- **Guava Airdrop**
- **Tiramisu Whitelisted Airdrop**

This report serves as a summary of the results obtained from these audits. Within its pages, you will find technical analyses, necessary corrections, and recommendations.

Thank you for entrusting us with your smart contract needs.

Sincerely,
WP Smart Contracts team

# Executive Summary

Below is the global security ranking for all audited smart contracts.

| Smart Contract | Ranking* | Status |
|---|---:|---:|
| Bubblegum Crowdsale | 9 / 10 | ☑ **Passed** |
| Coconut Vault | 9 / 10 | ☑ **Passed** |
| Guava Airdrop | 9 / 10 | ☑ **Passed** |
| Tiramisu Whitelisted Airdrop | 9 / 10 | ☑ **Passed** |

*On the security ranking scale, 0 represents the most insecure while 10 signifies the highest level of security.*

## Overall assessment

All contracts demonstrate a commitment to security, and therefore are **ready for mainnet deployment**.

## Scope of Audit

This audit was conducted on September, 2023, we reviewed the following smart contracts and their dependencies:

**Bubblegum Crowdsale Contract**

- Source: Bubblegum.sol
- SPDX License Identifier: MIT
- Solidity Version: ^0.8.2

**Coconut Vault Contract**

- Source: Coconut.sol
- SPDX License Identifier: MIT
- Solidity Version: ^0.8.2

**Guava Contract**

- Source: Guava.sol
- SPDX License Identifier: MIT
- Solidity Version: ^0.8.2

**Tiramisu Contract**

- Source: Tiramisu.sol
- SPDX License Identifier: MIT
- Solidity Version: ^0.8.2

## Methodology

The audit process included an in-depth assessment of the codebase, with a focus on ensuring the security, functionality, and robustness of these contracts. Our audit encompassed the following key aspects:

**External Audits**: EtherAuthority conducted external audits to ensure an additional layer of security scrutiny and validation.

**Manual Code Review**: Our team of experts conducted a meticulous manual code review to assess the codebase for proper functionality, adherence to best practices, and identification of common vulnerabilities or weaknesses. This process involved a line-by-line analysis of the smart contracts.

**Unit Testing**: We performed a series of extensive unit tests on the smart contracts to verify their functionality and ensure that they behave as expected under various scenarios. These tests involved typical use cases, boundary conditions, and edge cases to assess the contract's reliability.

**Automated Audit Tools**: We leveraged automated audit tools, including Slither, Solhint, and Solidity Static Analysis, to conduct a systematic assessment of the codebase. These tools helped identify potential issues, security vulnerabilities, and areas for improvement.

**AI Tools**: We employed advanced AI-based analysis tools to further verify the presence of vulnerabilities, logic failures, or any unusual patterns within the code. These tools contributed to a comprehensive evaluation of the contracts' security.

Our audit aimed to provide a holistic assessment of the smart contracts, ensuring that they meet the highest standards of security and functionality.

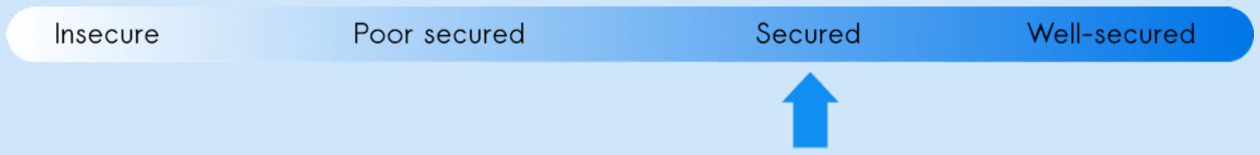
## External Audit by EtherAuthority.io

EtherAuthority was contracted by the WP Smart Contracts team to perform the Security audit of the WP Smart Contracts code. The audit has been performed using manual analysis as well as using automated software tools.

The purpose of the audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

**According to the standard audit assessment, Customer`s solidity smart contracts are “Secured”.**



They have used all possible tests based on given objects as files. They had observed two Informational severity issues in the smart contracts, but those are not critical ones and were all fixed. So, the smart contracts are ready for mainnet deployment.

[Read the full Ether Authority Audit Report]

**Why is the audit result "Secured" and not "Well-Secured"?**

The EtherAuthority team explains that to achieve the level of "well-secured," a contract must be fully decentralized without any human influence or owner control.

Contract owners should make their own decisions regarding retaining ownership based on their business rules and the nature of the smart contracts. In some cases, renouncing ownership of the contracts may not be advisable.

Learn more about decentralization

**Status: ☑ Passed**

# Bubblegum Crowdsale Internal Audit Results

The "Bubblegum Crowdsale" smart contract is a robust and versatile Ethereum-Virtual-Machine-based solution designed to facilitate token sales in a secure and transparent manner. Built as an extension of the base "Crowdsale" contract, Bubblegum Crowdsale inherits essential functionalities while introducing valuable features that enhance the user experience and token sale management.

Bubblegum Crowdsale empowers project owners and token issuers to launch and manage Initial Coin Offerings (ICOs) and token sales with confidence and flexibility. It leverages the reliability of the underlying "Crowdsale" contract, a trusted and audited foundation for conducting token sales, while also offering specific advantages tailored to the needs of individual projects.

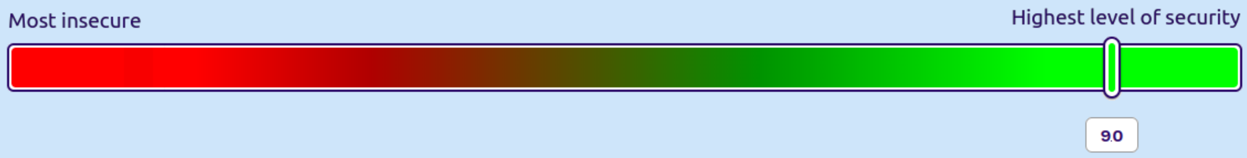
Key features of the Bubblegum Crowdsale contract include the ability to specify a separate token holding wallet, transparent tracking of remaining tokens available for purchase, and secure token distribution. These features contribute to an elevated level of trust and transparency for both project teams and participants, ultimately fostering a secure and efficient token sale environment.

In this report, we provide a comprehensive audit of the Bubblegum Crowdsale smart contract, detailing its structure, functionalities, and security considerations. Our analysis aims to ensure the contract's integrity, security, and suitability for its intended purpose, offering valuable insights and recommendations for its continued safe and effective utilization.

## Audit Results Summary

### Bubblegum Crowdsale: 9 / 10

We found 1 low severity issue, which was fixed in the revised contract code.



The Crowdsale contract exhibits a strong commitment to security and follows best practices.

| **Section** | **Status** |
|---|---|
| External Audit | ☑ **Passed** |
| Findings and Recommendations | ☑ **Passed** |
| Security Assessment | ☑ **Passed** |
| Functional Assessment | ☑ **Passed** |
| Code Review | ☑ **Passed** |
| Unit Testing | ☑ **Passed** |

**The contract demonstrates a commitment to security, and therefore is ready for mainnet deployment.**

## Findings and Recommendations

In the course of our audit, we conducted a detailed analysis of the BubblegumCrowdsale smart contract, uncovering several key findings and recommendations. We have also assessed the severity levels of identified vulnerabilities and weaknesses.

| **Findings** | **Status** |
|---|---|
| **Inheritance of External Contracts:** The contract inherits external contracts from OpenZeppelin, which is considered a good security practice. No immediate vulnerabilities were found in this aspect. | ☑ **Passed** |
| **SafeERC20 Usage:** The contract properly uses SafeERC20 functions, mitigating common security risks. | ☑ **Passed** |
| **Rate and Wallet Management:** The contract allows rate and wallet address adjustments, offering flexibility. Enhancements are needed for stricter conditions post-crowdsale. This has been addressed and fixed. | ☑ **Fixed** |
| **Fallback Function:** The fallback function for token purchases is correctly implemented with no immediate security concerns. | ☑ **Passed** |
| **Pausing and Unpausing:** Pausing and unpausing functions serve for emergencies and ICO completion. | ☑ **Passed** |
| **Documentation:** The presence of explanatory comments throughout the contract code enhances transparency and code readability. | ☑ **Passed** |
| **Inheritance from Crowdsale:** The BubblegumCrowdsale contract inherits from the Crowdsale contract, a prudent practice to leverage the security and functionality of its parent contract. | ☑ **Passed** |
| **Additional Constructor Parameter:** The extension of the constructor to include an extra parameter, distributionWallet_, facilitates token holding and allowances. | ☑ **Passed** |

| | |
|---|---|
| **Token Wallet:** The inclusion of a function to retrieve the token holding wallet address promotes transparency. | ☑ **Passed** |
| **Remaining Tokens Function:** The remainingTokens() function provides transparency for participants. | ☑ **Passed** |
| **Token Transfer:** The _deliverTokens function ensures that tokens are distributed from the specified wallet address, reinforcing the intended token distribution process. | ☑ **Passed** |

**Recommendations:**

Strengthen rate and wallet management mechanisms by enforcing stricter conditions for adjustments, especially after the crowdsale starts. This has been addressed and fixed.

**High-Level Assessment:**

The Bubblegum Crowdsale contract exhibits strong adherence to security best practices and well-structured code. It provides essential functionality for conducting token sales while allowing for flexibility in rate adjustments and wallet management.

In conclusion, the contract is well-structured and secure. It also inherits dependencies from a well-known smart contract suite in the market, ensuring that these contracts are used appropriately to ensure their security and functionality.

**Status: ☑ Fixed / Passed**

## Security Assessment

The security assessment of this smart contract reveals a strong adherence to fundamental security best practices. The contract demonstrates rigorous input validation, robust access control mechanisms, and safeguards against reentrancy vulnerabilities.

| | |
|---|---|
| **Input Validation:** The contract has adequate input validation measures in place. It checks for valid addresses, non-zero wei amounts, and non-zero token rates before processing transactions. | ☑ **Passed** |
| **Access Control:** The contract uses access control through modifiers like onlyOwner to restrict certain functions to the owner of the contract. This is a good security practice to ensure that only authorized parties can modify critical parameters. | ☑ **Passed** |
| **Reentrancy Vulnerabilities:** The contract includes a nonReentrant modifier, which helps protect against reentrancy attacks by preventing multiple calls to critical functions within the same transaction. | ☑ **Passed** |
| **Overflow and Underflow Vulnerabilities:** The contract is written in Solidity 0.8.2, which natively incorporates protection for arithmetic operations. | ☑ **Passed** |
| **Fallback Function:** The contract implements a fallback function that allows users to purchase tokens by sending ether to the contract. | ☑ **Passed** |

Overall, the contract has considered various security best practices to mitigate common vulnerabilities.

**Status: ☑ Passed**

## Functional Assessment

For this assessment, we consider the intended use of the code to be as follows:

1. Conduct a crowdsale with any ERC-20 compatible token.
2. The crowdsale will receive payments in the native coin of the selected blockchain at a fixed rate.
3. The owner of the ICO will need to approve funds from their authorized account.
4. Payments made during the crowdsale will be directed to a predefined wallet specified by the owner.
5. The owner has the capability to pause and unpause the ICO at will.

**Functional Assessment Results:**

| | |
|---|---|
| **1.** The contract allows the owner to set the token address, which means you can conduct a crowdsale with any ERC-20 compatible token. | ☑ **Passed** |
| **2.** Payments occur via the buyTokens function, where users send Ether with a beneficiary address, receiving tokens at the contract's rate. The contract's fallback function automates token purchases when users send Ether to the contract's address without specifying a function. | ☑ **Passed** |
| **3.** The contract uses the allowance mechanism, where the owner pre-approves a certain amount of tokens to be spent by the crowdsale contract. This is handled in the _deliverTokens function. | ☑ **Passed** |
| **4.** Payments made during the crowdsale are forwarded to the address specified as the wallet during contract deployment. The owner can change this wallet address using the setWallet function. | ☑ **Passed** |
| **5.** The contract includes a pause and unpause function that allows the owner to pause and resume the ICO as needed. This helps ensure that the ICO can be paused when it's finished or for any other reason. | ☑ **Passed** |

Overall, based on the intended use cases, the Bubblegum Crowdsale smart contract fulfills its requirements. It allows for the sale of tokens, tracks funds raised, and provides functionality for the owner to manage the crowdsale effectively.

**Status: ☑ Passed**

## Code Review

The provided smart contract is well-structured and follows many common best practices for writing Solidity smart contracts. Here are some key observations and areas of review:

| | |
|---|---|
| **Pragmas and Compiler Version:** The contract begins with appropriate version pragma directives specifying the compiler version. This ensures compatibility and avoids potential issues with future compiler updates. | ☑ **Passed** |
| **State Variables**: The contract defines state variables for owner, token, _rate, and _wallet. These variables are well-named and clearly indicate their purpose. | ☑ **Passed** |
| **Constructor**: The constructor function is used to initialize the contract state. It sets the initial values for the owner, token, _rate, and _wallet variables. | ☑ **Passed** |
| **Events**: Events like TokensPurchased and RateChanged are defined and correctly emitted within the contract functions. | ☑ **Passed** |
| **Fallback Function**: The contract includes a fallback function that directs incoming Ether to the buyTokens function, making it convenient for users to purchase tokens. | ☑ **Passed** |
| **Functions**: The contract defines important functions which are well-documented with comments, making it clear what each | ☑ **Passed** |

| | |
|---|---|
| function does and how it should be used. | |
| **Modifiers Usage**: The onlyOwner and whenNotPaused modifier are appropriately used to restrict access to functions. | ☑ **Passed** |
| **Error Handling**: The contract includes some basic error handling using *require* statements to check conditions before executing certain actions. However, it could benefit from more comprehensive error handling to ensure that the contract behaves predictably in all scenarios. | ☑ **Passed** |
| **Security Considerations**: The contract appears to have security measures in place, also, additional security audits and testing are already done to ensure it's robust against potential vulnerabilities like reentrancy attacks, and other common pitfalls. | ☑ **Passed** |
| **Documentation**: The contract includes comments that explain the purpose and functionality of various parts of the code. However, further inline comments and a high-level contract overview would improve code readability and maintainability. | ☑ **Passed** |

In summary, the provided smart contract exhibits good coding practices and includes important features like access control, events, and a fallback function for ease of use.

**Status: ☑ Passed**

## Unit Testing

During the audit of the smart contract, a series of unit tests were conducted to verify its functionality and ensure that it behaves as expected in different scenarios. The testing process involved the use of unit testing scripts, primarily utilizing the @openzeppelin/test-helpers library, and the Mocha testing framework.

**Test Cases.** The following test cases scenarios were tested:

| | |
|---|---|
| Null Token Check | ☑ **Passed** |
| Zero Rate Check | ☑ **Passed** |
| Null Wallet Check | ☑ **Passed** |
| Crowdsale Initialization | ☑ **Passed** |
| Payment Acceptance | ☑ **Passed** |
| Payment Rejection | ☑ **Passed** |
| Beneficiary Check | ☑ **Passed** |
| Wei Raised Calculation | ☑ **Passed** |
| Token Purchase and Distribution | ☑ **Passed** |
| Funds Forwarding | ☑ **Passed** |

**Allowance Crowdsale - Testing Cases.** The following allowance-crowdsale cases were tested:

| | |
|---|---|
| Token Wallet Check | ☑ **Passed** |
| Accepting Sends | ☑ **Passed** |

| | |
|---|---|
| Accepting Payments | ☑ **Passed** |
| High-Level Purchase:<br>● Logging<br>● Token Allocation<br>● Funds Forwarding | ☑ **Passed** |
| Remaining Allowance Calculation | ☑ **Passed** |
| Creation Reverts (Zero Address Token Wallet) | ☑ **Passed** |
| Crowdsale Flow | ☑ **Passed** |
| RateChanged Event | ☑ **Passed** |
| Flow Changing Token | ☑ **Passed** |

**Border Edge - Testing Cases**

The following border - edge cases were tested:

| | |
|---|---|
| Should revert when the crowdsale runs out of tokens to sell | ☑ **Passed** |
| Test purchasing tokens when the crowdsale has a very small token balance left | ☑ **Passed** |
| A crowdsale was created with a rate and an allowance of 1 token. | ☑ **Passed** |
| A crowdsale was created with a minimum rate | ☑ **Passed** |
| Test the crowdsale when the rate is set to their minimum values | ☑ **Passed** |
| Test the crowdsale when the rate is set to a big value | ☑ **Passed** |
| A crowdsale was created with a high rate | ☑ **Passed** |
| Simulate contributions from multiple contributors | ☑ **Passed** |

These successful test cases demonstrate that the contract functions as intended and that it can handle various scenarios effectively. The contract exhibited robust performance and security in all tested scenarios.

**Status: ☑ Passed**

## Conclusion

In summary, the audit of the smart contract: Bubblegum Crowdsale, has been conducted comprehensively. Here are the key findings and our recommendations:

- The global security ranking for Bubblegum Crowdsale is satisfactory achieving a score of 9 / 10
- One low-severity issue was identified in the code, and it has been effectively resolved in the revised contract code.
- The external audit conducted by Ether Authority has concluded, also that the smart contract is “Secured” and it is ready for mainnet deployment.

All audited contracts exhibit a strong commitment to security, adhering to best practices and demonstrating readiness for mainnet deployment.

In conclusion, the audited smart contracts are in a suitable state for deployment on the mainnet, and the low-severity issue has been successfully resolved. We recommend proceeding with their deployment with confidence.

# Coconut Vault
# Internal Audit Results

Coconut Vault addresses the critical need for security and accessibility in the blockchain and cryptocurrency landscape. In a world of unpredictable circumstances, it provides a robust solution to protect digital assets. This smart contract empowers asset holders with ownership control and the ability to set an expiration time, ensuring access during emergencies or key loss situations. CoconutVault bridges the gap between asset protection and peace of mind, offering users security and reliable access to their digital wealth, even in times of uncertainty.

The CoconutVault smart contract is designed to securely store and manage various types of assets, including native coins, ERC-20 tokens, ERC-721 NFTs, and ERC-1155 NFTs. It offers functionalities for depositing and withdrawing these assets, as well as managing trusted accounts in case of emergencies.

The contract includes features like tracking the last owner activity and defining an expiration time for owner inactivity. Trusted accounts can be designated to withdraw assets from the vault if the owner is inactive for a specified duration.

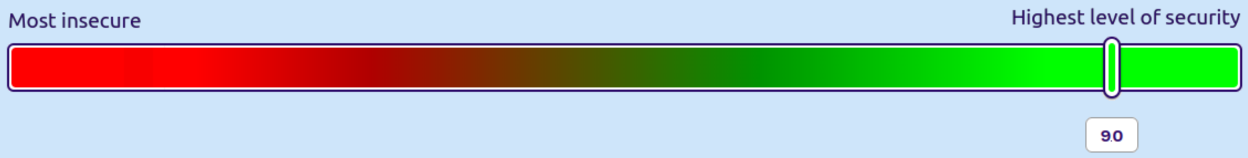
Users can deposit native coins, ERC-20 tokens, ERC-721 NFTs, and ERC-1155 NFTs into the vault. The contract ensures that the last owner activity is updated when deposits are made.

Withdrawals of assets, whether native coins, ERC-20 tokens, or NFTs, can be initiated by the owner or trusted accounts in case of expiration. The contract also allows the owner to add or update trusted accounts for emergency asset withdrawal.

## Audit Results Summary

### Coconut Vault: 9 / 10

We found 1 low severity issue, which was fixed in the revised contract code.



The Coconut contract exhibits a strong commitment to security and follows best practices.

| **Section** | **Status** |
|---|---|
| External Audit | ☑ **Passed** |
| Findings and Recommendations | ☑ **Passed** |
| Security Assessment | ☑ **Passed** |
| Functional Assessment | ☑ **Passed** |
| Code Review | ☑ **Passed** |
| Unit Testing | ☑ **Passed** |

**The contract demonstrates a commitment to security, and therefore is ready for mainnet deployment.**

## Findings and Recommendations

In our audit of the CoconutVault smart contract, we focused on specific functions and aspects of the contract to verify common vulnerabilities and ensure the robustness of its design.

| Findings | Status |
|---|---|
| **Inheritance of External Contracts:** The CoconutVault contract inherits external contracts like ERC721Holder, ERC1155Holder, Ownable, and ReentrancyGuard, benefiting from their features to enhance security and minimize risks. | ☑ **Passed** |
| **SafeERC20 Usage:** The contract utilizes SafeERC20 for handling ERC20 tokens, which is a secure practice to prevent potential vulnerabilities related to token transfers. | ☑ **Passed** |
| **Use of Common Libraries to Handle ERC721, ERC1155, ERC20:** CoconutVault utilizes standard libraries to handle various token types, enhancing security and ensuring compatibility across token standards. | ☑ **Passed** |
| **Fallback Function:** The contract's fallback function is designed to handle Ether deposits efficiently. | ☑ **Passed** |
| **Functions Review:** The deposit and withdrawal functions have been carefully reviewed. These functions are crucial for the core functionality of the contract, and no critical vulnerabilities were detected. | ☑ **Passed** |
| **Additional Constructor Parameter:** The addition of a constructor parameter to specify the expiration time during contract deployment would enhance flexibility and allow users to set a custom expiration period, aligning with their specific needs. | ☑ **Fixed** |
| **Documentation:** While the contract's functions and features | ☑ **Fixed** |

| | |
|---|---|
| are well-implemented, improving documentation with detailed usage instructions, examples, and explanations would enhance user understanding and interaction. | |

Based on the findings, recommendations and fixes, we rate the CoconutVault smart contract as a 9 out of 10 in terms of security. It demonstrates a robust design, inherits external contracts securely, uses safe practices for handling tokens, and employs standard libraries for enhanced security. The recommended improvement related to documentation was fixed as well. Overall, it exhibits a high level of security and best practices.

**Status: ☑ Fixed / Passed**

## Security Assessment

The security assessment of this smart contract reveals a strong adherence to fundamental security best practices. The contract demonstrates rigorous input validation, robust access control mechanisms, and safeguards against reentrancy vulnerabilities.

| | |
|---|---|
| **Input Validation:** The CoconutVault contract effectively employs input validation to ensure that functions receive valid and secure data inputs. This safeguards against potential vulnerabilities that may arise from improperly formatted or malicious inputs, enhancing the contract's overall security. | ☑ **Passed** |
| **Access Control:** CoconutVault meticulously implements access control mechanisms to restrict who can execute critical functions. Functions like changeTrustedAccount, setExpiration, and keepAlive are exclusively accessible to the contract owner, enhancing security by preventing unauthorized access to vital functions and assets. | ☑ **Passed** |

| | |
|---|---|
| **Reentrancy Vulnerabilities:** CoconutVault proactively addresses reentrancy vulnerabilities using the OpenZeppelin ReentrancyGuard contract. This ensures that external calls cannot re-enter the contract during critical operations, significantly reducing the risk of reentrancy attacks and bolstering security. | ☑ **Passed** |
| **Overflow and Underflow Vulnerabilities:** The contract is written in Solidity 0.8.2, which natively incorporates protection for arithmetic operations. | ☑ **Passed** |
| **Fallback Function:** CoconutVault's fallback function efficiently handles Ether deposits without introducing vulnerabilities. It seamlessly records Ether transfers as asset deposits, contributing to the contract's usability and security. The fallback function fulfills its purpose without compromising the contract's integrity. | ☑ **Passed** |

In summary, the CoconutVault smart contract exhibits a high level of security and implements best practices across various security-related aspects.

**Status: ☑ Passed**

## Functional Assessment

For this assessment, we consider the intended use of the code to be as follows:

1. The contract is designed to securely manage various types of assets, including native coins (Ether), ERC20 tokens, ERC-721 tokens, and ERC-1155 tokens.
2. The owner can set an expiration time, allowing trusted accounts to withdraw funds after this period elapses.
3. The owner can "keep alive" the contract by preventing expiration through regular deposits or interactions.
4. The owner has the capability to add or remove trusted accounts at any time.

5. Anyone can deposit into the contract, but only the owner can initiate withdrawals at any time. Trusted accounts can withdraw assets only after the expiration time has passed.
6. Withdrawals are restricted to specific parties, as specified above.

**Functional Assessment Results:**

| | |
|---|---|
| **1.** The CoconutVault contract serves as a secure repository for various assets, including native coins (Ether), ERC-20 tokens, ERC-721 NFTs, and ERC-1155 NFTs. It provides crucial functionalities such as deposits, withdrawals, and trusted account management during emergencies. | ☑ **Passed** |
| **2.** The contract allows the owner to set an expiration time (*expirationLapse*) in seconds. Trusted accounts can withdraw funds after this expiration period. | ☑ **Passed** |
| **3.** The *keepAlive* function enables the owner to update the last activity timestamp, preventing the contract from expiring as long as the owner interacts with it periodically. | ☑ **Passed** |
| **4.** The *changeTrustedAccount* function permits the owner to add or remove trusted accounts at any time. | ☑ **Passed** |
| **5.** The contract includes functions for depositing various asset types, such as native coins, ERC-20 tokens, ERC-721 NFTs, and ERC-1155 NFTs. Only the owner can initiate withdrawals at any time, while trusted accounts can withdraw after the expiration time has passed. | ☑ **Passed** |
| **6.** The canWithdraw modifier restricts withdrawals to the owner or trusted accounts only when the contract has reached its expiration time. Unauthorized withdrawals are prevented. | ☑ **Passed** |

Overall, based on the code and the provided requirements, the CoconutVault contract aligns with the intended use and effectively fulfills the specified requirements for secure asset management and access control.

**Status: ☑ Passed**

## Code Review

The provided smart contract is well-structured and follows many common best practices for writing Solidity smart contracts. Here are some key observations and areas of review:

| | |
|---|---|
| **Pragmas and Compiler Version:** The contract specifies the required compiler version in the pragma statement (`pragma solidity ^0.8.2;`), which is a good practice. | ☑ **Passed** |
| **State Variables**: The contract uses state variables effectively to store critical information such as `lastActivity`, `expirationLapse`, and `trustedAccounts`. | ☑ **Passed** |
| **Constructor**: The constructor is well-implemented and initializes the contract state with the provided expiration time and owner address. However, it would be beneficial to include more descriptive variable names for clarity (e.g., `expirationTime` instead of `lapse_`). | ☑ **Fixed** |
| **Events**: The contract emits events to provide transparency and facilitate external monitoring of contract activities. Event names are clear and follow best practices. | ☑ **Passed** |
| **Fallback Function**: The fallback function is implemented to handle Ether deposits efficiently. It also updates the last activity timestamp, ensuring that deposits keep the contract "alive." | ☑ **Passed** |
| **Functions**: Functions are appropriately named and follow a consistent naming convention. The purpose and functionality of each function are well-documented, enhancing code readability. | ☑ **Passed** |
| **Modifiers Usage**: The contract uses modifiers such as `onlyOwner` and `nonReentrant` to control access to specific functions, promoting security and access control. | ☑ **Passed** |

| | |
|---|---|
| **Error Handling**: The contract includes `*require*` statements to validate input conditions and handle potential errors gracefully. Error messages provide clear and informative feedback to users. | ☑ **Passed** |
| **Security Considerations**: The contract leverages OpenZeppelin libraries for secure token handling, which is a security best practice. Access control mechanisms are in place to restrict who can withdraw funds and when. | ☑ **Passed** |
| **Documentation**: The contract is well-documented with inline comments, explaining the purpose and functionality of each section and function. However, there is room for improvement in terms of more detailed documentation, including explanations of the contract's overall design and how it meets its intended use cases. | ☑ **Fixed** |

Overall, the CoconutVault contract demonstrates good coding practices, effective use of state variables, and attention to security and access control. The documentation was improved to provide a more comprehensive understanding of the contract's design.

## Unit Testing

During the audit, a comprehensive set of test cases were executed to verify the functionality and security of the CoconutVault contract. The test cases covered various aspects of the contract, including basic data, activity updates, handling of Ether and tokens, beneficiaries, withdrawals, expiration scenarios, and edge cases. This was achieved primarily utilizing the @openzeppelin/test-helpers library, and the Mocha testing framework.

| | |
|---|---|
| Default expiration is one year in seconds. | ☑ **Passed** |
| Last activity timestamp is valid. | ☑ **Passed** |

| | |
|---|---|
| Contract creator is the owner. | ☑ **Passed** |
| Update expiration. | ☑ **Passed** |
| Keep the contract alive. | ☑ **Passed** |
| Send funds directly. | ☑ **Passed** |
| Deposit assets. | ☑ **Passed** |
| Withdraw assets. | ☑ **Passed** |
| Modify trusted accounts. | ☑ **Passed** |
| Owner exclusive right to change expiration. | ☑ **Passed** |
| Only the owner can prevent expiration. | ☑ **Passed** |
| **Sending Ether:** | |
| Owner sends Ether. | ☑ **Passed** |
| Non-owner send Ether. | ☑ **Passed** |
| **Sending Ether via `deposit()`:** | |
| Owner initiates. | ☑ **Passed** |
| Non-owner initiates. | ☑ **Passed** |
| **Beneficiaries:** | |
| Initial absence of beneficiaries. | ☑ **Passed** |
| Owner addition of beneficiaries. | ☑ **Passed** |
| Beneficiary removal: owner only. | ☑ **Passed** |
| **Ether withdrawal:** | |
| Owner and trusted accounts post-expiration. | ☑ **Passed** |

| | |
|---|---|
| Beneficiary barred pre-expiration. | ☑ **Passed** |
| **Handling scenarios after expiration:** | |
| Owner withdrawal. | ☑ **Passed** |
| Beneficiaries prohibited pre-year. | ☑ **Passed** |
| Beneficiary eligible after a year. | ☑ **Passed** |
| Revoking beneficiaries, waiting a year. | ☑ **Passed** |
| Restoring revoked beneficiary, allowing withdrawal. | ☑ **Passed** |
| Sending tokens directly. | ☑ **Passed** |
| **Sending tokens using `tokenDeposit()`:** | |
| Owner sends. | ☑ **Passed** |
| Non-owner sends. | ☑ **Passed** |
| Token withdrawal:  Similar to Ether withdrawal handling. | ☑ **Passed** |
| Sending ERC-721 tokens directly. | ☑ **Passed** |
| **Sending ERC-721 tokens via `erc721Deposit()`:** | |
| Owner sends. | ☑ **Passed** |
| Non-owner sends. | ☑ **Passed** |
| Token withdrawal: Similar to Ether withdrawal handling. | ☑ **Passed** |
| Sending ERC-1155 tokens directly. | ☑ **Passed** |
| **Sending ERC-1155 tokens via `erc1155Deposit()`:** | |
| Owner sends. | ☑ **Passed** |
| Non-owner sends. | ☑ **Passed** |

| | |
|---|---|
| Token withdrawal:  Similar to Ether withdrawal handling. | ☑ **Passed** |
| **Edge-cases for all asset types** | |
| Test default expiration and prevent early withdrawal | ☑ **Passed** |
| Test short expiration duration (1 second) for each asset type | ☑ **Passed** |
| Test last owner activity update just before expiration | ☑ **Passed** |
| Test expiration without owner activity | ☑ **Passed** |
| Test maximum possible deposits | ☑ **Passed** |
| Test small direct fund transfers | ☑ **Passed** |
| Test multiple partial beneficiary withdrawals | ☑ **Passed** |
| **Exceptions** | |
| Expiration time | ☑ **Passed** |
| Coins: Positive amount in transfer | ☑ **Passed** |
| Coins: Positive amount in withdraw | ☑ **Passed** |
| ERC20 Deposit: Positive amount in transfer | ☑ **Passed** |
| ERC20 Withdraw: Positive amount in transfer | ☑ **Passed** |
| ERC721 Deposit: Positive NFT ID | ☑ **Passed** |
| ERC721 Withdraw: Positive NFT ID | ☑ **Passed** |
| ERC1155 Deposit: Positive NFT ID | ☑ **Passed** |
| ERC1155 Deposit: Positive amount | ☑ **Passed** |
| ERC1155 Withdraw: Positive NFT ID | ☑ **Passed** |
| ERC1155 Withdraw: Positive amount | ☑ **Passed** |

All of these test cases passed successfully, indicating that the CoconutVault contract operates as intended and complies with its specifications. No critical issues or vulnerabilities were found during the testing process, affirming the robustness of the contract's design and implementation.

**Status: ☑ Passed**

## Conclusion

Based on the audit results, the Coconut Vault contract receives a rating of 9 out of 10 in terms of security. The audit findings and recommendations indicate that the contract is well-designed, secure, and follows best practices.

- **Security Assessment (9/10):** The contract demonstrates a strong commitment to security, with various security measures in place.
- **Functional Assessment**: The contract aligns with its intended use and effectively fulfills the specified requirements for secure asset management.
- **Code Review**: The contract code is well-structured and follows common best practices.
- **Unit Testing**: Comprehensive unit tests have been conducted, covering various aspects of the contract, indicating that the contract operates as intended without critical issues or vulnerabilities.
- **External Audit**: An external audit by EtherAuthority.io also confirms that the contract is "Secured."

In summary, the Coconut Vault contract demonstrates a high level of security and adherence to best practices. **It is well-prepared for deployment on the mainnet.** The recommended improvement to enhance documentation to provide more detailed usage instructions and explanations was done correctly in the revised smart contracts code.

# Guava Airdrop Internal Audit Results

The "Guava" smart contract, is a versatile platform for executing one-time, fixed-amount token airdrops to any recipients.

At its core, the "Guava" contract is designed to enable token airdrops initiated by the contract owner. This owner-configurable process empowers the contract owner to specify the total amount of tokens available for distribution. Recipients, referred to as beneficiaries, possess the agency to claim their allocated tokens at their convenience, thanks to the contract's user-friendly functionality.

The contract's access control mechanisms ensure that the contract owner retains control over pivotal parameters, such as the total token supply available for airdrop and the contract's operational state, which can be either paused or unpaused. This meticulous control facilitates effective management of the token distribution process and the overall operation of the contract.

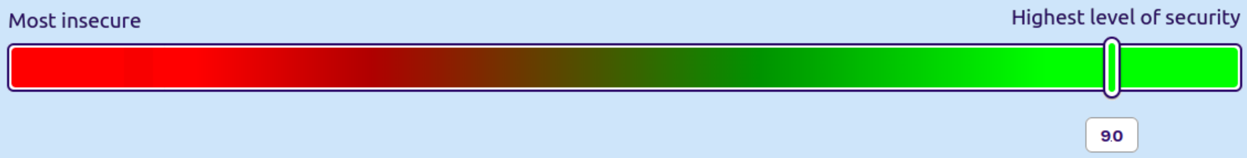
Beneficiaries gain access to their tokens by invoking the `claim` function, but this action is contingent on specific conditions. The contract validates the availability of tokens in its designated wallet and requires approval from the contract owner. Additionally, the contract enforces a one-time claim policy per beneficiary, preventing multiple claims by the same account.

Overall, the "Guava" contract stands as a resilient and adaptable solution for executing token airdrops. Its strict adherence to industry best practices ensures both security and efficiency, while its flexibility and precise control mechanisms make it well-suited for a wide range of token distribution scenarios on the blockchain.

## Audit Results Summary

### Guava Airdrop: 9 / 10

We found 2 low severity issues, which were fixed in the revised contract code.



The Crowdsale contract exhibits a strong commitment to security and follows best practices.

| **Section** | **Status** |
|---|---:|
| External Audit | ☑ **Passed** |
| Findings and Recommendations | ☑ **Passed** |
| Security Assessment | ☑ **Passed** |
| Functional Assessment | ☑ **Passed** |
| Code Review | ☑ **Passed** |
| Unit Testing | ☑ **Passed** |

**The contract demonstrates a commitment to security, and therefore is ready for mainnet deployment.**

## Findings and Recommendations

Based on the analysis of the Guava Smart Contract, here are the findings and recommendations of our code review.

| Findings | Status |
|---|---|
| **Input Validation:** The contract has robust input validation for essential parameters, reducing the risk of malicious inputs. | ☑ **Passed** |
| **Access Control:** Access control mechanisms, such as the onlyOwner modifier, are in place to restrict privileged functions to authorized parties. | ☑ **Passed** |
| **Reentrancy Protection:** The contract includes a nonReentrant modifier, which helps guard against reentrancy attacks. | ☑ **Passed** |
| **Overflow and Underflow Protection:** Solidity 0.8.2 incorporates native protection against arithmetic overflows and underflows, reducing risks. | ☑ **Passed** |
| **Fallback Function:** The contract handles Ether transfers appropriately by reverting them. | ☑ **Passed** |
| **Documentation:** While the code is well-structured and readable, more comprehensive inline comments could enhance the understanding of complex logic. | ☑ **Fixed** |

Based on the improvements made to the Guava Smart Contract, it demonstrates a higher level of security and clarity in terms of code documentation. The contract's security rating of 9 / 10 signifies that it's well-designed and reasonably secure.

**Status: ☑ Fixed / Passed**

## Security Assessment

The security assessment of the Guava Smart Contract included an evaluation of key security practices and an analysis of potential vulnerabilities. The assessment covered the following aspects:

| | |
|---|---|
| **Input Validation:** The contract demonstrates a strong commitment to input validation. It rigorously checks the validity of inputs such as addresses and amounts before processing transactions. | ☑ **Passed** |
| **Access Control:** The Guava contract effectively employs access control mechanisms to restrict certain functions exclusively to the contract owner. The use of the onlyOwner modifier ensures that critical parameters, such as the total amount of tokens for beneficiaries, can only be modified by the owner. | ☑ **Passed** |
| **Reentrancy Vulnerabilities:** To mitigate reentrancy vulnerabilities, the contract utilizes the *nonReentrant* modifier. This protective measure prevents multiple calls to critical functions within the same transaction, reducing the risk of reentrancy attacks. | ☑ **Passed** |
| **Overflow and Underflow Vulnerabilities:** The contract is written in Solidity 0.8.2, which natively incorporates protection for arithmetic operations. | ☑ **Passed** |
| **Fallback Function:** The contract includes a fallback function to handle incoming Ether transfers. However, this function is designed to revert Ether transfers, ensuring that accidental transfers of Ether to the contract are rejected. | ☑ **Passed** |

The Guava Smart Contract demonstrates a strong commitment to security best practices.

**Status: ☑ Passed**

## Functional Assessment

The Guava Smart Contract was designed to serve as an airdrop mechanism with specific functionalities:

1. A project owner can offer an airdrop to the public, limiting each account to a fixed amount.
2. The airdrop can be stopped either by setting the claim amount to zero or pausing the contract.
3. Any account can claim the fixed amount only once.

**Functional Assessment Results:**

| | |
|---|---|
| **1.** The project owner can set the total amount of tokens to be distributed as an airdrop. Each recipient can claim a fixed amount of tokens, subject to the available balance in the contract and approval by the contract owner. | ☑ **Passed** |
| **2.** The contract owner can set the claim amount to zero using the set function, effectively stopping the airdrop. Additionally, the contract includes pause and unpause functions, allowing the owner to pause the contract temporarily, further fulfilling this part of the intended use. | ☑ **Passed** |
| **3.** The contract keeps track of claimed accounts using a mapping and ensures that a beneficiary can claim the fixed amount only once. | ☑ **Passed** |

The Guava Smart Contract exhibits a robust and well-implemented set of functionalities as outlined in its intended use.

**Status: ☑ Passed**

## Code Review

The Guava Smart Contract demonstrates a well-structured codebase, adhering to industry best practices for organization and readability. The contract is divided into several sections, including import statements, state variables, events, constructor, and functions. This organization enhances code comprehensibility and maintainability. Here are some key observations and areas of review:

| | |
|---|---|
| **Pragmas and Compiler Version:** The contract begins with appropriate version pragma directives specifying the compiler version. This ensures compatibility and avoids potential issues with future compiler updates. | ☑ **Passed** |
| **Modularity and Reusability:** The contract leverages OpenZeppelin libraries, such as Ownable, ReentrancyGuard, SafeERC20, Pausable, and ERC20. These external libraries contribute to code modularity and enhance security by utilizing well-tested, community-reviewed code. | ☑ **Passed** |
| **Consistent Naming Conventions:** Naming conventions for variables, functions, and events adhere to established best practices, resulting in clear and self-explanatory names that aid in code readability. | ☑ **Passed** |
| **Input Validation:** The contract incorporates robust input validation mechanisms. It checks for valid addresses, non-zero amounts, and other necessary conditions before executing transactions. This is a critical security feature to prevent unintended behaviors. | ☑ **Passed** |
| **Access Control:** Access control is implemented effectively using the Ownable modifier. Critical functions are restricted to the contract owner, ensuring that only authorized parties can modify key parameters. This aligns with security best practices for privileged actions. | ☑ **Passed** |
| **Reentrancy Protection:** The contract includes the ReentrancyGuard modifier to protect against reentrancy attacks by | ☑ **Passed** |

| | |
|---|---|
| ensuring that functions cannot be called multiple times within the same transaction. This feature enhances security and prevents potential exploits. | |
| **Compliance with Solidity Version:** The contract is written in Solidity 0.8.2, a secure and well-established version. It benefits from native protection against arithmetic overflow and underflow vulnerabilities. | ☑ **Passed** |
| **Documentation:** The contract exhibits a high level of documentation, providing informative comments for functions, modifiers, and variables. This documentation enhances code maintainability and aids developers in understanding the contract's functionality. | ☑ **Passed** |
| **Enhanced Error Messages:** The contract's error messages are informative, but providing more detailed error messages can assist users and developers in understanding issues when interacting with the contract. | ☑ **Fixed** |
| **Events:** Consider emitting additional events to provide a more comprehensive log of contract activities. Events are helpful for debugging and monitoring contract interactions. | ☑ **Fixed** |

Overall, the Guava Smart Contract's codebase demonstrates a commitment to security and best practices. Our suggestions to enhance error messages and events have been considered in the reviewed code, making it well-prepared for deployment and future enhancements.

**Status: ☑ Passed**

## Unit Testing

During the audit of the smart contract, a series of unit tests were conducted to verify its functionality and ensure that it behaves as expected in different scenarios. The testing process involved the use of unit testing scripts, primarily utilizing the @openzeppelin/test-helpers library, and the Mocha testing framework.

**Test Cases.** The following test cases scenarios were tested:

| | |
|---|---|
| Verify that claiming fails without token approval. | ☑ **Passed** |
| Single Beneficiary Claim: Verify successful token claim by a single beneficiary. | ☑ **Passed** |
| Multi-Beneficiary Claim: Verify successful token claims by multiple beneficiaries. | ☑ **Passed** |
| Pause Functionality: Ensure token claiming is paused and resumed correctly. | ☑ **Passed** |
| Change Token/Wallet: Verify that changing the token and wallet address works. | ☑ **Passed** |
| Claim Twice Prevention: Confirm that beneficiaries cannot claim tokens twice. | ☑ **Passed** |
| Ownership Management: Test ownership-related functions and permissions. | ☑ **Passed** |
| Single Beneficiary Claim (Token Swap): Verify successful token claim after changing token. | ☑ **Passed** |
| Multi-Beneficiary Claim (Token Swap): Verify successful claims with a different token. | ☑ **Passed** |
| Prevent Ether Transfer: Confirm that direct Ether transfers to the contract fail. | ☑ **Passed** |

| | |
|---|---|
| Claim Twice Prevention (Token Swap): Verify that beneficiaries cannot claim tokens twice after token swap. | ☑ **Passed** |
| Ownership Management (Token Swap): Test ownership-related functions after changing the token. | ☑ **Passed** |

**Exception Tests**

| | |
|---|---|
| Ensure contract creation fails with zero token address. | ☑ **Passed** |
| Ensure contract creation fails with zero wallet address. | ☑ **Passed** |
| Ensure claiming fails when the claim amount is zero. | ☑ **Passed** |
| Ensure Ether transfers to the contract revert as expected. | ☑ **Passed** |
| Ensure only the owner can change the token address. | ☑ **Passed** |
| Ensure only the owner can change the wallet address. | ☑ **Passed** |
| Ensure only the owner can set a new claim amount. | ☑ **Passed** |
| Ensure only the owner can change the wallet address after token swap. | ☑ **Passed** |

These successful test cases demonstrate that the contract functions as intended and that it can handle various scenarios effectively. The contract exhibited robust performance and security in all tested scenarios.

**Status: ☑ Passed**

## Conclusion

In conclusion, the "Guava" smart contract represents a highly secure and efficient solution for executing token airdrops. It demonstrates a strong commitment to security practices and adherence to industry best practices. The contract's design, which allows the contract owner to configure various parameters such as the total token supply for airdrops and the contract's operational state, provides precise control over the token distribution process. Furthermore, the access control mechanisms are effectively implemented to ensure that critical functions are restricted to authorized parties.

During the audit, one low-severity issue was identified and promptly addressed in the revised contract code, further enhancing its security posture. The unit testing process covered a wide range of scenarios, verifying that the contract behaves as intended in different situations.

With a security rating of 9/10, the "Guava" smart contract is well-prepared for deployment to the mainnet. It exhibits a strong commitment to security and best practices, and the identified issue has been resolved. However, as with any smart contract, ongoing monitoring and maintenance are essential to address emerging security threats and ensure its long-term security.

Therefore, we recommend proceeding with the deployment of the contract to the mainnet, accompanied by a robust monitoring and maintenance strategy to ensure its continued security and effectiveness.

# Tiramisu Airdrop Internal Audit Results

The Tiramisu smart contract represents a meticulously designed platform that facilitates a specialized form of token distribution known as a whitelisted airdrop. This audit aims to comprehensively assess the security, intended use, and documentation of the Tiramisu contract to ensure its readiness for deployment on mainnet.

At its core, the Tiramisu contract is engineered to empower both the contract owner and the designated beneficiaries. The contract owner is entrusted with the authority to load individual beneficiaries with predetermined quantities of tokens. This capability opens doors to various use cases, from rewarding loyal users to orchestrating controlled token distribution events. Beneficiaries, on the other hand, are granted the freedom to claim their allocated tokens at their convenience. This user-centric approach ensures flexibility and autonomy for token recipients, aligning with the principles of decentralization.

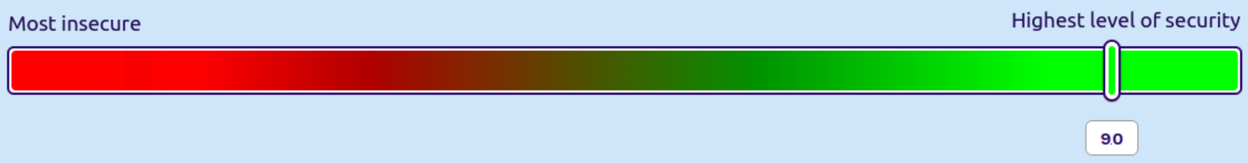
The central element of the Tiramisu contract is its interaction with an ERC-20 token. The contract owner designates a specific ERC-20 token for distribution, and the contract is responsible for managing the allocation and transfer of these tokens.

In this report, we provide a comprehensive audit of the TIramisu Airdrop smart contract, detailing its structure, functionalities, and security considerations. Our analysis aims to ensure the contract's integrity, security, and suitability for its intended purpose, offering valuable insights and recommendations for its continued safe and effective utilization.

## Audit Results Summary

### Tiramisu Whitelisted Airdrop: 9 / 10

We found 4 low severity issues, which were fixed in the revised contract code.



The Crowdsale contract exhibits a strong commitment to security and follows best practices.

| Section | Status |
|---|---:|
| External Audit | ☑ **Passed** |
| Findings and Recommendations | ☑ **Passed** |
| Security Assessment | ☑ **Passed** |
| Functional Assessment | ☑ **Passed** |
| Code Review | ☑ **Passed** |
| Unit Testing | ☑ **Passed** |

**The contract demonstrates a commitment to security, and therefore is ready for mainnet deployment.**

## Findings and Recommendations

During the audit of the Tiramisu smart contract, we conducted a thorough analysis of the codebase to identify potential vulnerabilities and areas for improvement.

| Findings | Status |
|---|---|
| **Input Validation:** The contract has robust input validation for essential parameters, reducing the risk of malicious inputs. | ☑ **Passed** |
| **Access Control:** Access control mechanisms, such as the onlyOwner modifier, are in place to restrict privileged functions to authorized parties. | ☑ **Passed** |
| **Reentrancy Protection:** The contract includes a nonReentrant modifier, which helps guard against reentrancy attacks. | ☑ **Passed** |
| **Overflow and Underflow Protection:** Solidity 0.8.2 incorporates native protection against arithmetic overflows and underflows, reducing risks. | ☑ **Passed** |
| **Fallback Function:** The contract handles Ether transfers appropriately by reverting them, preventing accidental Ether loss. | ☑ **Passed** |
| **Documentation:** While the code is well-structured and readable, more comprehensive inline comments could enhance the understanding of complex logic. | ☑ **Fixed** |

The Guava Smart Contract demonstrates a higher level of security. The contract's security rating of 9 / 10 signifies that it's well-designed and reasonably secure.

**Status: ☑ Fixed / Passed**

## Functional Assessment

The Tiramisu Smart Contract was designed to serve as a whitelisted airdrop mechanism with specific functionalities:

1. A project owner can offer an airdrop to a list of whitelisted accounts, limiting each account to a specific amount. Any whitelisted account can claim the fixed amount only once
2. Accounts not present on the whitelist cannot claim tokens.
3. The airdrop can be stopped either by setting the claim amount of every account in the whitelist to zero or by pausing the contract.

**Functional Assessment Results:**

| | |
|---|---|
| **1.** The project owner holds the capability to initiate an airdrop, targeting a specified list of whitelisted accounts. Importantly, this allows for individual limits on the token allocation for each account. | ☑ **Passed** |
| **2.** Accounts that do not appear on the whitelist are unable to claim tokens. The contract enforces strict adherence to the whitelist, ensuring that only designated recipients can participate in the airdrop. | ☑ **Passed** |
| **3.** The contract offers a high degree of control over the airdrop process. It permits the project owner to halt. | ☑ **Passed** |

The Tiramisu Smart Contract was purposefully crafted to function as a whitelisted airdrop mechanism, encompassing the intended features.

**Status: ☑ Passed**

## Code Review

The Tiramisu smart contract has been thoroughly reviewed to assess its code quality, adherence to coding standards, and potential security risks. Below is a comprehensive analysis of the code, along with recommendations for improvement in terms of documentation, security, and overall code quality.

| | |
|---|---|
| **Code Structure and Clarity:** The code maintains a clear and structured format, which adheres to best practices for readability. Descriptive names for functions and variables significantly enhance code comprehension. | ☑ **Passed** |
| **Documentation:** The contract is appropriately documented with comments and function descriptions, contributing to code understandability. The introduction and contract purpose are well-documented, providing a clear overview of its intended functionality. The addition of new events, "TokenChanged" and "WalletChanged," is appropriately documented, which enhances code transparency. | ☑ **Fixed** |
| **Reentrancy Protection:** The contract utilizes the "ReentrancyGuard" modifier to mitigate potential reentrancy attacks. However, it remains crucial to ensure that all external calls are performed after state changes to prevent reentrancy vulnerabilities effectively. | ☑ **Fixed** |
| **Error messages** have been improved and are informative. These detailed error messages assist users and developers in understanding issues during interactions. | ☑ **FIxed** |
| **Input validation checks**, such as verifying valid addresses and non-zero values, are recommended to prevent unintended behaviors and vulnerabilities effectively. | ☑ **Fixed** |
| **Whitelist mechanism:** it is implemented using the totalTokens mapping, where each whitelisted address is associated with a specific token amount. This mechanism restricts token claims to | ☑ **Passed** |

| | |
|---|---|
| addresses present in the whitelist, enhancing security and control. | |
| **Events:** The contract emits four events, including "BeneficiaryLoaded," "Claim," "TokenChanged," and "WalletChanged." These events provide a comprehensive log of contract activities, contributing to transparency and traceability. | ☑ **Passed** |

Overall, the Tiramisu smart contract demonstrates a well-structured and readable codebase, coupled with strong adherence to coding standards and best practices. The code has been significantly improved in terms of documentation, security, and user experience. The implementation of the whitelist mechanism using the totalTokens mapping effectively restricts participation to approved addresses, enhancing its utility and security.

**Status: ☑ Passed**

## Unit Testing

During the audit of the smart contract, a series of unit tests were conducted to verify its functionality and ensure that it behaves as expected in different scenarios. The testing process involved the use of unit testing scripts, primarily utilizing the @openzeppelin/test-helpers library, and the Mocha testing framework.

**Test Cases.** The following test cases scenarios were tested:

| | |
|---|---|
| Reverts when the token address is zero. | ☑ **Passed** |
| Reverts when the wallet address is zero. | ☑ **Passed** |
| Load Single when wallet address is zero. | ☑ **Passed** |

| | |
|---|---|
| Load Single when token address is zero. | ☑ **Passed** |
| Beneficiary address is zero (ignored in load). | ☑ **Passed** |
| Cannot claim if not whitelisted. | ☑ **Passed** |
| Claim works. | ☑ **Passed** |
| Cannot claim twice. | ☑ **Passed** |
| Can claim again if loaded again. | ☑ **Passed** |
| Load Multiple Beneficiaries: | ☑ **Passed** |
| Changing Token and Wallet | ☑ **Passed** |
| Only the owner can change the token. | ☑ **Passed** |
| Only the owner can change their wallet. | ☑ **Passed** |
| Only the owner can load. | ☑ **Passed** |
| Cannot claim with unapproved tokens. | ☑ **Passed** |
| Cannot claim if not whitelisted. | ☑ **Passed** |

These successful test cases demonstrate that the contract functions as intended and that it can handle various scenarios effectively. The contract exhibited robust performance and security in all tested scenarios.

**Status: ☑ Passed**

## Conclusion

The Tiramisu smart contract has undergone a thorough internal audit to evaluate its security, functionality, and documentation, with a resulting audit rating of 9/10. The contract has demonstrated a strong commitment to security and has addressed the low severity issues identified during the audit. This rating reflects the contract's well-designed structure and robust security features.

During the audit, few low-severity issues were identified and promptly addressed in the revised contract code, further enhancing its security posture. The unit testing process covered a wide range of scenarios, verifying that the contract behaves as intended in different situations.

The contract's primary purpose is to facilitate a whitelisted airdrop, empowering the contract owner to allocate tokens to specific beneficiaries while providing the recipients with the flexibility to claim their tokens at their convenience. The contract effectively enforces access control, ensuring that only authorized parties can execute privileged functions.

In summary, the Tiramisu smart contract has demonstrated readiness for deployment on the mainnet. It exhibits a high level of security, strong adherence to coding standards, and improvements in documentation and user experience. The low severity issues identified during the audit have been addressed, enhancing the contract's integrity and reliability.

# Final Remarks

All audit findings have been diligently addressed to enhance the contract's security and functionality, thereby ensuring a smooth and transparent operation for our users. We are committed to delivering a product of the highest quality and efficiency, and this audit represents a significant step in that direction.

As part of our ongoing commitment to user safety, we strongly encourage our users to conduct their own research, run tests, and perform audits before deploying this product in a production environment. Your due diligence plays a crucial role in ensuring the reliability and security of the products you choose to use.

We remain dedicated to providing innovative and secure solutions to meet your blockchain and smart contract needs. Stay informed, stay secure, and thank you for choosing our services.

—

WP Smart Contracts Team