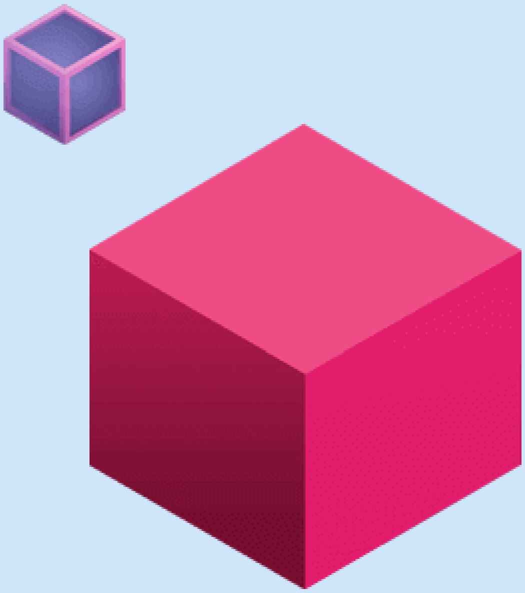
WP Smart Contracts

The Ultimate Smart Contracts Integration for WordPress



ADVANCED CROWDSALE

WP Smart Contracts

Smart Contract Audits

Nov 2024

Disclaimer

WP Smart Contracts Audits are not, nor should be considered, an “endorsement” or “disapproval” of any specific project or team. These audits do not, and should not, indicate the economic value or viability of any “product” or “asset” created by a team or project that contracts WP Smart Contracts for a security review.

WP Smart Contracts Audits do not offer any warranty or guarantee regarding the absolute bug-free nature of the analyzed technology, nor do they provide any insight into the technology's proprietors, business model, or legal compliance. These audits should not be used to make decisions regarding investment or engagement with any particular project. They do not constitute investment advice and should not be relied upon as such.

WP Smart Contracts Audits represent a rigorous review process aimed at assisting clients in enhancing their code quality while mitigating the substantial risks inherent in cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets carry a high level of ongoing risk. WP Smart Contracts emphasizes that each company and individual is responsible for their own due diligence and continuous security. Our objective is to help reduce attack vectors and address the risks associated with evolving technology. We do not guarantee the security or functionality of the technology we analyze.

A WP Smart Contracts Audit is a detailed document that analyzes a specific set of source code provided to WP Smart Contracts by a client. It includes a structured collection of testing results, analysis, and insights regarding the code’s architecture, implementation, and adherence to best practices. It serves as confirmation that a client has undergone a round of auditing to improve the quality of their IT infrastructure and/or source code.

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.

In our pursuit of excellence, during November 2024, we embarked on a series of audits for the four batch of smart contracts within WPSmartContracts. These audits encompassed critical contracts, including:

- **Caramel Advanced Crowdsale**

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
Caramel Advanced Crowdsale	9 / 10	☑ **Passed**

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

Scope of Audit

This audit was conducted on November, 2024, we reviewed the following smart contracts and their dependencies:

Caramel Advanced Crowdsale Contract

- Source: CaramelCrowdsale.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:

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.

Caramel Crowdsale

Internal Audit Results

The CaramelCrowdsale contract is a robust, flexible solution for managing token sales with advanced security features. Designed to allow participants to purchase tokens with Ether or any ERC-20 token, the contract offers both immediate and post-sale delivery options. Tokens can be held until the sale ends, ensuring secure distribution to contributors. A whitelist feature restricts participation to authorized addresses, protecting the sale process, while role-based access allows administrators to update whitelist settings and sale parameters. Dynamic rate and contribution settings provide customizable control, allowing the owner to set unique rates and limits for each token accepted as payment, alongside adjustable Ether contribution thresholds.

Features:

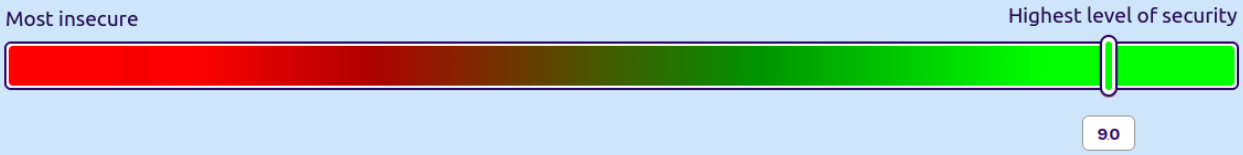
- Receive contributions in Ether or the native coin of the blockchain
- Receive contributions in any ERC-20 token
- Option to restrict participation to whitelisted contributors only
- Supporters can send contributions using a GUI
- Distribute tokens to supporters automatically during the crowdsale, or in batches at its finalization
- Tokens can optionally be distributed directly to a UBE staking contract
- Receive the funds directly in your ETH wallet
- ERC-20 Token Distribution
- Allowance Crowdsale
- Opening and closing dates
- Ability to finalize the crowdsale

In this report, we provide a comprehensive audit of the CaramelCrowdsale smart contract, evaluating its structure, functionalities, and security to ensure its integrity and suitability for intended use. This audit includes recommendations to support its safe and effective application in the long term.

Audit Results Summary

Caramel Crowdsale: 9 / 10

We found 1 critical and 1 high severity issue, which were fixed in the revised contract code.



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

Section	**Status**
Findings and Recommendations	☑ **Passed**
Security Assessment	☑ **Passed**
Functional Assessment	☑ **Passed**
Code Review	☑ **Passed**
Unit Testing	☑ **Passed**

Findings and Recommendations

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

Findings	Status
SafeERC20 Usage: The contract properly uses SafeERC20 functions, mitigating common security risks.	☑ **Passed**
Reentrancy Protection: The nonReentrant modifier from ReentrancyGuard mitigates reentrancy attacks, crucial for protecting functions that handle payments and transfers.	☑ **Passed**
Role-Based Access Control: The WhitelistedRole and WhitelistAdminRole roles enforce secure access, limiting contributions to whitelisted users and enhancing control over participation.	☑ **Passed**
Modular Structure with Inheritance: The modular design with separate contracts for specific functions improves readability, testing, and maintainability, making the code easy to audit.	☑ **Passed**
Event Logging for Transparency: The contract emits events for key state changes like token purchases and whitelist updates, aiding in transparency and easier monitoring.	☑ **Passed**
Flexible Token Rate Management: Dynamic token rates per ERC-20 token allow the owner to set varying rates, minimums, and maximums, supporting multiple payment assets.	☑ **Passed**
Controlled Contribution Periods: TimedCrowdsale enforces opening and closing times, controlling the crowdsale timeline and restricting contributions outside the intended period.	☑ **Passed**

Configurable Whitelist Validation: The useWhitelist feature allows toggling whitelist validation, providing flexibility between restricted and open participation modes.	☑ Passed
Potential Denial of Service in Batch Token Distribution: The distributeTokens() function could fail due to gas limits if the contributor list grows too large, preventing contributors from receiving tokens. **Severity**: Critical **Recommendation**: Implement pagination or split distribution calls into smaller batches, or enforce gas usage constraints to prevent the function from reaching gas limits. **Actions**: a pagination parameter was added to the distributeTokens function.	☑ Fixed
Unused TokenData Initialization Check **Severity**: High **Impact**: Setting a token rate to zero would result in getTokenAmountWithToken() division errors, which could halt token purchase functionality and cause failed transactions. **Recommendation (Audit team)**: Add a validation step in setTokenData() to ensure non-zero rates or handle this within the purchase functions. **Answer (Dev team):** Acknowledged, this is the intended behavior to deactivate tokens. This is controlled in the UI to avoid conflicts.	☑ Acknowledged
Reentrancy Guard Missing in Withdrawal **Severity**: High **Impact**: Absence of a reentrancy guard in withdrawTokens()	☑ Fixed

poses a reentrancy risk for token transfers, potentially allowing malicious actors to exploit the function. **Recommendation**: Add a nonReentrant modifier to withdrawTokens() to prevent reentrancy attacks on token transfers.	
Low Gas Optimization in Batch Processing **Severity**: Medium **Impact:** High gas costs in addWhitelistedBatch() and removeWhitelistedBatch() could lead to transaction failures, especially for larger lists. **Recommendation**: Limit batch sizes or allow self-registration of accounts with admin approval to lower gas usage. **Answer (Dev Team)**: this is an administrator only function, and the contract owner can define the batch size manually.	☑ **Acknowledged**
The following methods should be declared external: • openingTime() • closingTime() • addWhitelistAdmin(address) • renounceWhitelistAdmin() • addWhitelisted(address) • removeWhitelisted(address) • addWhitelistedBatch(address[]) • removeWhitelistedBatch(address[]) • renounceWhitelisted() • finalize() • withdrawTokens(address) • balanceOf(address) • primary() • transferPrimary(address) • transfer(IERC20,address,uint256) • buyTokensWithTokens(address,uint256,address)	☑ **Fixed**

Severity: Low **Recommendation:** Use the external attribute for functions never called from the contract, and change the location of immutable parameters to calldata to save gas.	

High-Level Assessment:

The CaramelCrowdsale contract has been thoroughly reviewed, with critical issues resolved to ensure security and functionality. It combines robust features like role-based access control, safe token transfer mechanisms, flexible token rate settings, reentrancy protection, and modular design. Overall, CaramelCrowdsale is well-architected for secure, efficient token sales, aligning with best practices and prepared for effective deployment.

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

Based on the provided requirements, here’s an assessment of the CaramelCrowdsale contract’s functionality:

Receive Contributions in Ether or the Native Coin of the Blockchain: The contract allows Ether contributions, enabling supporters to send funds in the blockchain’s native currency, compatible with chains where Ether or an equivalent native coin is used.	☑ **Passed**
Receive Contributions in Any ERC-20 Token: The contract supports multiple ERC-20 tokens, allowing contributions with any token added as a payment option. The owner can configure rates, minimum and maximum contributions for each supported ERC-20 token.	☑ **Passed**
Option to Restrict Participation to Whitelisted Contributors Only: The contract includes a flexible whitelisting mechanism. The owner can enforce whitelist restrictions to ensure that only approved contributors can participate.	☑ **Passed**
Distribute Tokens to Supporters Automatically During the Crowdsale, or in Batches at Its Finalization: The contract offers multiple delivery modes. Tokens can be delivered immediately or held for batch distribution, allowing owners flexibility in delivery timing. The PostDeliveryCrowdsale feature allows tokens to be distributed after finalization or upon individual withdrawal by contributors.	☑ **Passed**
Tokens Can Optionally Be Distributed Directly to a UBE Staking Contract:	☑ **Passed**

The contract includes an optional feature for automatic transfer to a UBE staking contract. If a staking contract is specified, tokens are sent to the staking contract for contributors instead of directly to their wallets.	
Receive the Funds Directly in Your ETH Wallet: Funds raised during the crowdsale are transferred directly to the specified wallet, ensuring that contributions are received in real-time by the crowdsale organizer's wallet.	☑ **Passed**
ERC-20 Token Distribution: ERC-20 tokens are distributed to contributors using safeTransfer methods from the SafeERC20 library, ensuring compatibility with ERC-20 standards.	☑ **Passed**
Allowance Crowdsale: The contract follows an allowance model, where tokens are held by a designated distribution wallet and approved for transfer to the crowdsale contract. This ensures token availability for distribution without transferring ownership.	☑ **Passed**
Opening and Closing Dates: The contract includes an optional TimedCrowdsale mechanism, allowing the owner to set specific opening and closing dates. Contributions are restricted to within this period.	☑ **Passed**
Ability to Finalize the Crowdsale: The contract can be finalized by the owner once the crowdsale ends. Finalization emits an event and prevents any further token purchases, which aligns with standard crowdsale finalization requirements.	☑ **Passed**

The CaramelCrowdsale contract meets all specified requirements. It provides a versatile and secure framework for running a crowdsale with flexible contribution and distribution options, token support, whitelisting, and automated or batch-based delivery. The contract can be integrated with a user-facing GUI and supports custom configurations for ERC-20 tokens and Ether, making it suitable for various crowdsale structures.

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 function does and how it should be used.	☑ **Passed**
Modifiers Usage: The onlyOwner and whenNotPaused modifier are	☑ **Passed**

appropriately used to restrict access to functions.	
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, **more than 200 unit test cases** 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.

1. Whitelist Management

- **Description**: Tests the access control functionality, specifically enforcing participation restrictions based on whitelist status.
- **Tests**:
  - Verifies that only whitelisted addresses can participate.
  - Confirms that only admins can add/remove whitelisted addresses.
  - Tests batch whitelisting and un-whitelisting.
  - Ensures purchase acceptance/rejection based on whitelist status.

2. Affiliate Program

- **Description**: Assesses affiliate program features, such as commission handling, blacklisting, and pausing functionality.
- **Tests**:
  - Ensures affiliates receive commissions and admins can set rates.
  - Validates blacklisting to prevent commissions for blacklisted affiliates.
  - Tests pausing/unpausing the program to control commission flows.

3. Allowance Crowdsale

- **Description**: Tests functionality related to managing token allowances and enforcing limits on contributions.

- **Tests**:
  - Confirms that the correct token allowance is reported.
  - Checks that token wallet creation reverts if the address is zero.
  - Validates fund forwarding, allowance limits, and event logging during high-level purchases.

4. Ether Contribution Limits

- **Description**: Verifies the handling of minimum and maximum contribution limits for Ether-based contributions.
- **Tests**:
  - Ensures that Ether contributions respect the configured minimum and maximum limits.
  - Validates admin-only access for setting contribution boundaries.

5. Factory and Token Management

- **Description**: Tests the factory contract's ability to create tokens, configure rewards, and manage factory settings.
- **Tests**:
  - Confirms master and manager roles for token creation.
  - Validates fee management and updates to master addresses.
  - Tests deployment under various reward ratios and ensures correct token creation behavior.

6. Crowdsale Finalization

- **Description**: Verifies the finalization process, ensuring that actions like token distribution are managed correctly post-crowdsale.
- **Tests**:
  - Ensures only the owner can finalize the crowdsale.
  - Verifies that tokens cannot be bought after finalization.

  - Tests finalization restrictions to prevent multiple finalizations.

7. Post-Delivery Crowdsale

- **Description**: Assesses delayed token delivery, including individual withdrawals and batch distributions.
- **Tests**:
  - Validates that tokens are held until crowdsale ends.
  - Ensures token distribution to either contributors or a staking contract.
  - Tests batch distribution and multiple withdrawal prevention.

8. Rate Testing

- **Description**: Tests token distribution at various rate configurations, from large to fractional rates, across tokens with different decimals.
- **Tests**:
  - Verifies that rates above, equal to, and below 1 behave as expected for tokens with varying decimals.
  - Tests contributions directly and through purchase functions for multiple rate scenarios.

9. Timed Crowdsale

- **Description**: Tests the timed crowdsale functionality to ensure proper enforcement of opening and closing times.
- **Tests**:
  - Confirms that contributions are only accepted within specified time frames.
  - Tests extending and updating the closing time.
  - Validates that timed contributions reject payments outside allowed periods.

10. Edge Case Handling

- **Description**: Ensures robustness by testing for various edge cases and unusual conditions.
- **Tests**:
    - Verifies behavior when crowdsale runs out of tokens.
    - Tests handling of maximum Ether contributions.
    - Assesses behavior with zero and extreme token balances or rates.
    - Simulates contributions from multiple participants and non-approval tokens.

11. Ownership and Access Control

- **Description**: Tests ownership transfer and renouncement for secure access control.
- **Tests**:
    - Ensures only the owner can transfer ownership.
    - Confirms that ownership renouncement prevents non-owners from controlling the contract.

Each category's tests collectively confirm that the contract's core functionality, access control, and edge case handling are thoroughly assessed. This broad coverage supports the contract's compliance with the expected functional requirements.

Status: ☑ Passed

Conclusion

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

- The global security ranking is satisfactory achieving a score of 9 / 10
- One critical and one high-severity issue were identified in the code, and it has been effectively resolved in the revised contract code.

All audited contracts exhibit a strong commitment to security and adhering to best practices.

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