Smart Contract Security Audit Report - Example Template

Project: ExampleToken (BEP20 BabyDoge Clone Style) **Contract Address (if deployed):** N/A (Code Review) **GitHub Repository:** https://github.com/frenkiofficial/Crypto-Token-Example (https://github.com/frenkiofficial/Crypto-Token-Example) **Commit Hash Audited:** `[Insert Specific Commit Hash Here, e.g., a1b2c3d4...]` **Auditor:** [Example Auditor Name / Company Name] **Date Range:** [Start Date] - [End Date] **Report Date:** [Date Report Issued]

Table of Contents

---

1. Executive Summary

This report details the findings of a security audit performed on the `ExampleToken.sol` smart contract, located at the specified commit hash. The audit aimed to identify potential security vulnerabilities, assess adherence to best practices, and provide recommendations for improvement.

Overall Assessment: [Provide a high-level statement here, e.g., "The contract implements complex tokenomics including reflection and auto-LP features. While core BEP20 functionality is present, several areas require attention, particularly regarding owner privileges and gas efficiency. No critical vulnerabilities exploitable under standard conditions were identified *in this example assessment*, but several high and medium-severity findings require mitigation."] **<-- THIS IS EXAMPLE TEXT ONLY.**

Key Findings Summary:

- Critical Issues: 0 *(Example)*
- High Issues: 1 *(Example)*
- Medium Issues: 2 *(Example)*
- Low Issues: 1 *(Example)*
- Informational Issues: 2 *(Example)*
- Gas Optimizations: 1 *(Example)*

Note: This executive summary is based on the *hypothetical findings listed below for template demonstration purposes only*.

---

2. Audit Scope

The scope of this audit was limited to the following smart contract file at the specified commit hash:

- `ExampleToken.sol` (Commit: `[Insert Specific Commit Hash Here]`)

Out of Scope:

- Deployment scripts and process.
- Off-chain components or interactions.
- Integration with PancakeSwap or other external protocols beyond the interface level.
- Economic modeling or tokenomics viability.
- Frontend or DApp interactions.
- Previous versions of the code.

---

3. Methodology

The audit involved a combination of the following techniques:

- **Manual Code Review:** Line-by-line analysis of the Solidity code to identify logical errors, potential vulnerabilities, and deviations from best practices.
- **Static Analysis:** Utilization of automated tools (e.g., Slither, Mythril - *hypothetical usage*) to detect common vulnerability patterns.
- **Testing Review:** Examination of existing test cases (if provided) and identification of areas needing further test coverage.
- **Best Practice Comparison:** Evaluating the code against known Solidity and DeFi security standards and common pitfalls.

---

4. Severity Levels

Findings are classified according to their potential impact:

- **Critical:** Issues that could lead to significant loss of funds, contract paralysis, or manipulation of core logic. Require immediate attention.
- **High:** Issues that could lead to loss of funds for some users, denial of service, or significant deviations from intended behavior. Require urgent attention.
- **Medium:** Issues that could impact usability, lead to unexpected behavior under certain conditions, or increase attack surface. Should be addressed.
- **Low:** Minor deviations from best practices or issues with limited impact. Recommended to fix.
- **Informational:** Suggestions for code clarity, documentation, or minor improvements without direct security implications.
- **Gas Optimization:** Recommendations to improve the gas efficiency of contract functions.

---

5. Findings Summary

ID	Title	Severity	Status
CRITICAL			
C-01	*(No Critical Issues Found in this Example)*	*N/A*	*N/A*
HIGH			
H-01	Centralization Risk via Owner Privileges	High	**Open**
MEDIUM			
M-01	Potential Gas Griefing in `includeInReward`	Medium	**Open**
M-02	Swap Slippage Not Explicitly Controlled	Medium	**Open**
LOW			
L-01	Missing Event Emission for Fee Changes	Low	**Open**
INFORMATIONAL			
I-01	Inconsistent NatSpec Documentation	Informational	**Open**
I-02	Use of Magic Numbers for Fees	Informational	**Open**
GAS OPTIMIZATION			
G-01	Array Search in `includeInReward`	Gas Optimization	**Open**

Disclaimer: The findings listed above are **hypothetical examples created solely for illustrating the format of an audit report.** They do **NOT** represent an actual security analysis of the `ExampleToken.sol` code provided by Frenki.

---

6. Detailed Findings

CRITICAL

(No Critical Issues Found in this Example)

HIGH

H-01: Centralization Risk via Owner Privileges

- **Severity:** High
- **Location:** `setFees()`, `excludeFromFee()`, `includeInFee()`, `excludeFromReward()`, `includeInReward()`, `setSwapAndLiquifyEnabled()`, `setNumTokensSellToAddToLiquidity()`, `rescueToken()`, `rescueBNB()`
- **Description:** The contract owner (deployer) possesses significant control over critical parameters, including transaction fees, reward eligibility, and the swap/liquify mechanism. The owner can also withdraw any BNB or other BEP20 tokens sent to the contract address.
- **Impact:** A compromised owner key or a malicious owner could drastically alter tokenomics (e.g., set fees to 100%), disable core features, or steal funds sent directly to the contract, undermining user trust and potentially causing financial loss.
- **Recommendation:** Consider implementing a multi-sig wallet or a DAO governance structure for ownership control after initial setup. Alternatively, clearly document these risks to users. For fee/parameter changes, consider adding a timelock mechanism.
- **Status: Open**

MEDIUM

M-01: Potential Gas Griefing in `includeInReward`

- **Severity:** Medium
- **Location:** `includeInReward()`
- **Description:** The function iterates through the `_excludedFromReward` array to find and remove an address. If this array becomes very large, the gas cost to execute this function could exceed the block gas limit, making it impossible to include an address back into rewards.
- **Impact:** An attacker could potentially spam exclusions (if allowed by other logic, though currently owner-only) or the owner might inadvertently make the array too large, leading to a denial-of-service for the `includeInReward` function.
- **Recommendation:** Consider alternative data structures for managing exclusions that allow for constant-time removal (e.g., mapping address to index and swapping with the last element on removal). Alternatively, limit the maximum size of the exclusion list.
- **Status: Open**

M-02: Swap Slippage Not Explicitly Controlled

- **Severity:** Medium
- **Location:** `_swapTokensForBNB()`, `_addLiquidity()`
- **Description:** The calls to `swapExactTokensForETHSupportingFeeOnTransferTokens` and `addLiquidityETH` use `0` for the `amountOutMin` / `amountETHMin` / `amountTokenMin` parameters. This means the contract accepts any resulting amount, making it vulnerable to front-running or high slippage during volatile market conditions when performing the swap-and-liquify action.
- **Impact:** The contract might receive significantly less BNB than expected during swaps or create the LP pair under unfavorable conditions, negatively impacting the amount of liquidity added and potentially benefiting MEV bots.

- **Recommendation:** Calculate an acceptable minimum output amount based on a reasonable slippage tolerance (e.g., 0.5% - 1%) before calling the swap and add liquidity functions. This parameter should ideally be configurable by the owner.
- **Status: Open**

LOW

L-01: Missing Event Emission for Fee Changes

- **Severity:** Low
- **Location:** `setFees()`
- **Description:** The `setFees` function modifies the `_taxFee` and `_liquidityFee` state variables but does not emit an event indicating the change. The existing `FeesChanged` event is present but commented out in the provided code (assuming it might be uncommented later, but good to note).
- **Impact:** Off-chain services or users monitoring the contract cannot easily track historical fee changes without parsing transaction data.
- **Recommendation:** Ensure an event (like the existing `FeesChanged`) is emitted within the `setFees` function, logging the new fee values.
- **Status: Open**

INFORMATIONAL

I-01: Inconsistent NatSpec Documentation

- **Severity:** Informational
- **Location:** Throughout `ExampleToken.sol`
- **Description:** Some functions lack NatSpec comments (`@notice`, `@dev`, `@param`, `@return`), while others have them. Consistent documentation improves code readability and maintainability.
- **Recommendation:** Add comprehensive NatSpec comments to all public and external functions, explaining their purpose, parameters, and return values.
- **Status: Open**

I-02: Use of Magic Numbers for Fees

- **Severity:** Informational
- **Location:** `_calculateFees()`, `setFees()`
- **Description:** The fee percentages (e.g., `5` for 5%) and the divisor (`100`) are used directly as numeric literals ("magic numbers").
- **Impact:** Reduces code readability slightly and makes it harder to understand the context without looking at the calculation.
- **Recommendation:** Define constants for divisor (e.g., `PERCENTAGE_DIVISOR = 100;`) and potentially for maximum allowed fees to improve clarity.
- **Status: Open**

GAS OPTIMIZATION

G-01: Array Search in `includeInReward`

- **Severity:** Gas Optimization
- **Location:** `includeInReward()`
- **Description:** Searching through the `_excludedFromReward` array using a loop can be gas-intensive, especially as the array grows.
- **Impact:** Higher gas costs for the owner when calling this function.
- **Recommendation:** As mentioned in M-01, use a data structure that allows for more efficient removal, such as mapping the address to its index in the array and performing a swap-and-pop removal.
- **Status: Open**

---

7. General Recommendations

- **Increase Test Coverage:** Develop a comprehensive test suite (using Hardhat/Truffle) covering standard transfers, fee calculations, edge cases, reward exclusions, swap & liquify triggers, and owner function access controls.
- **Formal Verification (Optional):** For critical contracts, consider formal verification techniques to mathematically prove certain properties.
- **Monitoring:** Implement off-chain monitoring for critical events and contract state changes after deployment.

---

8. Disclaimer

This **example audit report** is provided for **illustrative and educational purposes only**. The findings listed herein are **hypothetical placeholders** and **do not represent the results of an actual security assessment** of the `ExampleToken.sol` contract found in Frenki's repository (`https://github.com/frenkiofficial/Crypto-Token-Example`).

This document does not guarantee the security of the contract. Smart contract security is complex, and even after an audit, vulnerabilities may exist. No liability is assumed by the creator of this template for any losses incurred through the use of the associated code or reliance on this example report format.

It is strongly recommended to obtain a thorough security audit from a reputable third-party firm before deploying any smart contract, especially one intended to handle user funds, to a Mainnet environment.
