QuillAudits

Audit Report
July, 2023

For

BabyDoge

Table of Content

Executive Summary

Project Name	BabyDoge
Project URL	https://babydoge.com/
Overview	MultiTokenBurnPortal is a type of smart contract which gives discount on fees of fee-on-transfer tokens while swapping tokens based on burned token amount. DeflationarySwap is a router contract, which takes small swap fees (up to 1%), collects whitelisted tokens (like stablecoins) and burns non-whitelisted tokens.
Audit Scope	*https://github.com/Baby-doge/BurnPortal* *https://github.com/Baby-doge/DeflationarySwap*
Contracts in Scope	1]MultiTokenBurnPortal.sol 2]DeflationarySwap.sol
Commit Hash	*BurnPortal:* 9c0907bf69b53b5766a8a092a638ec8452199440 *DeflationarySwap:* de0d4b23bba3224a6fde6eec704fb1758d8a4703
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	20 June 2023 - 12 July 2023
Updated Code Received	11 July 2023
Review 2	12 July 2023
Fixed In	*https://github.com/Baby-doge/BurnPortal/commit/041416802e20d17f61dc7b4922212f440fd71202* *https://github.com/Baby-doge/DeflationarySwap/commit/5835d8cfe06115038c20ff124ff6d89c3ba896c1*

Executive Summary

Mainnet Address

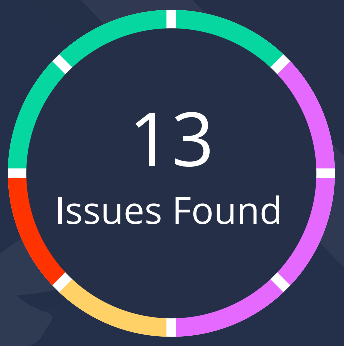
MultiTokenBurnPortal -
https://bscscan.com/address/0x10025F952Fd3dac40d52f09F58afC3bAaB4c2556#code

Pancake DeflationarySwap -
https://bscscan.com/address/0xAb6e1A058F15417E33c246fe68cB2525B02D13f8#code

BabySwap DeflationarySwap -
https://bscscan.com/address/0xC458090a9411109849222952aAbC1A7E6C4e655dB#code

ApeSwap DeflationarySwap -
https://bscscan.com/address/0xc2C25be4BB2ae8Fe4559B5aa9feC1a30dFe8FFf8#code

SushiSwap DeflationarySwap -
https://bscscan.com/address/0x9D8afA0C8Ff89c6BE5b919575A64497D86681e25#code



🟥 High 🟨 Medium

🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	2
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	1	4	3

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open
Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved
These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged
Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved
Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Using block.timestamp
- ✓ Multiple Sends
- ✓ Using SHA3
- ✓ Using suicide
- ✓ Using throw
- ✓ Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - MultiTokenBurnPortal

High Severity Issues

A.1 User can take discount without actually burning tokens for non-reverting tokens

Description

ERC20 token which doesn’t revert on transfer, transferFrom (or in other important operation) can make this code vulnerable. E.g. burnTokens() uses token's transferFrom function to transfer tokens to DEAD_WALLET to burn them. it can happen that a token that is getting transferred is a token that doesn't revert on failure (e.g while not having enough balance when transferring) instead, it returns a bool value. the returned value is not getting checked in the function and it proceeds on the next line to add the amount in burnedAmount[msg.sender][token] and in portals[token].totalBurned.

So in this way a user would be able to increase the amount of burned tokens that contract stores without burning the tokens. and after that, while swapping tokens the user can get a discount for the amounts that weren’t actually burned.

Remediation

Use *SafeERC20* for all ERC20 operations (especially for transfer and transfrerFrom) to protect against the mentioned scenario.

Status

Resolved

A.2 Fee amount collected by contract can be used by user to buy tokens

Description

buyTokensWithERC20() uses transferFrom to transfer tokens from msg.sender to contract and then sends it to adminFeeReceiver,feeReceiver as fee and uses remaining for swapping. In the case where the token that is getting transferFrom doesn't revert on failure(e.g when the user isn't having enough balance to transfer) the code will run without reverting the transaction, Now because the user hasn’t transferred any tokens to the MultiTokenBurnPortal while sending token fees to adminFeeReceiver,feeReceiver the contract send tokens from collected fees (in the case if contract is getting used for collecting fees) and use collected fee for swapping tokens.

Remediation

Use *SafeERC20* for all ERC20 operations, especially for transferFrom in this case.

Status

Resolved

Medium Severity Issues

A.3 _checkDiscounts() fails to validate given discounts

Description

_checkDiscounts() fails to validate from the 2nd index. It sets prevDiscount = _discounts[0] on L589 but never reassigns it to the next one while iterating through the loop. so while checking it every time, it checks the current index's discount data (i.e discount and burnAmount) with the prevDiscount's data which was set as the 0th index of _discounts. So for the 1st index, it compares index 0's discount data but for the next i.e for the 2nd index also, it compares the 0th index's data and not the 1st one which would be previous in this case.

This happens because prevDiscount is not getting updated in the loop.

Example: this is the discount array:
[{ discount: 1000, burnAmount: 50e18 },
 { discount: 2000, burnAmount: 150e18 },
 { discount: 2000, burnAmount: 300e18 }]

While validating this array the _checkDiscounts() function checks 1st index with 0th and the 2nd also with 0th index and not with 1st. so it fails to identify that 1st index's discount and 2nd index's discount amount are the same.

Remediation

assign prevDiscount = _discounts[i] in the loop after the if statement, so here it will assign the current discount element to prevDiscount which will be used for comparison with next element.

Status

Resolved

Low Severity Issues

A.4 reentrancy guard can be added for extra security

Description

In buyTokensWithBNB() and buyTokensWithERC20() there are external calls for sending fees to addresses, for transferring tokens to and from the msg.sender. It can happen that tokens having hooks can give control to the external address e.g msg.sender while transferring tokens, on the other hand side while sending native tokens to addresses e.g. as a fee. Once they receive a callback, they can then try to reenter for calling buyTokensWithBNB(), buyTokensWithERC20() functions.

In some cases, msg.sender can make a reentrant call and can use the issue discussed in A.2 and can use collected fees. This issue gets fixed after using SafeERC20 wrapper, but we recommend using reentrancy guard for both buyTokensWithBNB(), buyTokensWithERC20() functions to avoid any other issues related to reentrancy because of having external calls.

Remediation

Use a reentrancy guard like *ReentrancyGuard* for both buyTokensWithBNB() and buyTokensWithERC20() for extra security.

Status

Resolved

A.5 Unlocked pragma (pragma solidity ^0.8.0)

Description

Contracts are using floating pragma (pragma solidity ^0.8.0), Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Using floating pragma does not ensure that the contracts will be deployed with the same version. It is possible that the most recent compiler version gets selected while deploying a contract which has higher chance of having bugs in it.

Remediation

Remove floating pragma and use a specific compiler version with which contracts have been tested.

Status

Resolved

A.6 Centralization

Description

Some critical functions like setAdminFeeSettings(), setTokenBaseTax() etc are getting called by the owner and some functions by the manager, In the case of account key compromisation it creates risk where malicious admin/manager can use these functions maliciously because of having the power to make mentioned state changes.

Remediation

Consider using a multisig wallet for the addresses which can call these functions to mitigate the centralization.

BabyDoge team's comment: Owner and Manger roles will belong to multisig wallets.

Status

Acknowledged

A.7 Change the data type size

Description

BurnPortal struct is using uint128 for totalBought and for totalBurned, it is hard to directly predict that the amount of burn token and bought tokens won't reach the max limit of what uint128 can store (2**128-1), our recommendation would be to use uint256 for storing these values to avoid any problems in future while incrementing these values.

Remediation

Use uint256 for BurnPortal struct's totalBought and totalBurned variables.

Status

Resolved

Informational Issues

A.8 Care should be taken while setting discounts.

Description

For tokens with different decimals the burn amounts should be passed accordingly in the Discount array. E.g if any token uses 6 decimals to represent one whole tokens then while burning some amount lets say 100 tokens, the user would be burning 100e6 tokens and not 100e18 tokens. So its important that while specifying burnAmounts for 6 decimal tokens (or any other decimals) it should be specified carefully according to decimals.

Remediation

The burn amount should be specified according to decimals.

Status

Acknowledged

A.9 Unused imports

Description

Some imported interfaces (IBabyDogeFactory and IBabyDogePair) are not getting used and can be removed.

Remediation

Remove unused imports.

Status

Resolved

A.10 General recommendations

Description

set trusted address for adminFeeReceiver and feeReceiver because contract makes external calls while sending fees.

Remediation

Follow the recommendations mentioned above.

BabyDoge team's comment: adminFeeReceiver and feeReceiver will be trusted addresses.

Status

Acknowledged

A.11 Implemented code mismatches the functionality mentioned in readme/documentation

Description

Readme file states that "Swap path must be [WBNB -> BabyDoge] for buyTokensWithBNB". The code only checks that swap path should contain two elements and the first element would be weth but doesn't contain any check for checking if the second element (token address) is BabyDoge or not.

Remediation

Verify and add the statement to check the mentioned condition if code is not implementing the intended logic or change the condition stated in the readme if the mentioned condition is incorrect.

Auditor’s comment: The condition in the readme is now changed to [WBNB -> Token] which matches the implemented code.

Status

Resolved

B. Contract - DeflationarySwap

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

B.1 Unlocked pragma (pragma solidity ^0.8.0)

Description

Contracts are using floating pragma (pragma solidity ^0.8.0), Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Using floating pragma does not ensure that the contracts will be deployed with the same version. It is possible that the most recent compiler version gets selected while deploying a contract which has higher chances of having bugs in it.

Remediation

Remove floating pragma and use a specific compiler version with which contracts have been tested.

Status

Resolved

Informational Issues

B.2 Unused imports

Description

Some imported imports (SafeOwnable and IFactory) are not getting used and can be removed.

Remediation

Remove unused imports.

Status

Resolved

Functional Testing

Some of the tests performed are mentioned below:

MultiTokenBurnPortal:

- ✓ Should be able to buy tokens with BNB
- ✓ Should be able to buy tokens with ERC20
- ✓ User should be able to burn tokens
- ✓ Owner should be able to set manager
- ✓ Owner should be able to set admin fee and admin fee receiver
- ✓ Owner should be able to approve tokenIn
- ✓ Owner should be able to remove the approved tokenIn
- ✓ Owner should be able to appove router
- ✓ Owner should be able to withdraw ERC20 and BNB
- ✓ Owner or manager should be able to add token
- ✓ Owner or manager should be able to remove token
- ✓ Owner or manager should be able to set base tax
- ✓ Owner or manager should be able to set fee receiver
- ✓ Owner or manager should be able to set discounts
- ✓ User should be able to get discount based on burned token amount
- ✓ functions with access modifiers revert when called by unauthorized address

DeflationarySwap

- ✓ Should be able to swap tokens with swap functions
- ✓ Should be able to swap tokens with swap functions for fee-on-transfer tokens
- ✓ getAmountOut should return amountOut for amountIn minus fees
- ✓ getAmountsOut should return amountOut for amountIn minus fees for provided path
- ✓ getAmountIn should return amountIn plus fees for given amountOut
- ✓ getAmountsIn should return amountIn plus fees for given amountOut for provided path

Automated Tests

Static analysis covered some issues like unchecked return values and usage of old solc version. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Summary

In this report, we have considered the security of the BabyDoge. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in BabyDoge smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.
Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of BabyDoge smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.
QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the BabyDoge to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

800K
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2023

For

BabyDoge

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com