HashEx

BabyMatic token

smart contracts audit report

Prepared for:

babymatic.io

Authors: HashEx audit team

August 2021

Contents

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.
HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (https://hashex.org).

Introduction

HashEx was commissioned by the BabyMatic team to perform an audit of their smart contracts. The audit was conducted between July 27 and August 02, 2021.

The audited contracts are deployed to the Binance Smart Chain (BSC) at:
`0xB2ce41B71D93D7f5878F985C3e1A87A6229019BE`.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts.
- Formally check the logic behind given smart contracts.

Information in this report should be used to understand the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The contracts are mostly forks of Tiki token (audit available [1]) and for this reason, we focused on the unaudited parts of code, as well as modifications made by the BabyMatic team.

Contracts overview

`BabyMatic`

ERC20 token with the default values of 3% liquidity fee, 5% marketing fee, and 7% dividends to token holders and blacklist under the owner's control.

`BABYMATICDividendTracker`

Dividend tracker contract implementing the exclude from dividends mechanism and auto-processed payout.

`DividendPayingToken`

ERC20-like token with blocked transfers. The modified version of DividendPayingToken by Roger-Wu [2].

`ERC20`

Implementation of ERC20 token standard [3] with the possibility of increase/decrease allowance.

`IterableMapping.sol`

Library for iterable mapping from address to uint.

`Context, Ownable, SafeMath, SafeMathInt, SafeMathUint`

Support contracts directly forked from Tiki token.

Found issues

ID	Title	Severity	Response
01	BabyMatic: fees values are not limited	High	Acknowledged
02	BabyMatic: excludeFromDividends() is permanent	Medium	Acknowledged
03	BabyMatic: fees differ from website	Medium	Acknowledged
04	BabyMatic: Router update problem	Medium	Acknowledged
05	BabyMatic: swapTokensForEth uses 100% slippage	Medium	Acknowledged
06	BabyMatic: locked tokens	Medium	Acknowledged
07	BabyMatic: update balances with try method	Medium	Acknowledged
08	BabyMatic: addLiquidity() recipient	Medium	Acknowledged
09	BabyMatic: updateDividendTracker() not excluding dead address	Medium	Acknowledged
10	BabyMatic: excessive computations	Low	Acknowledged
11	BabyMatic: hardcoded addresses	Low	Acknowledged
12	DividendPayingToken: hardcoded addresses	Low	Acknowledged
13	Multiple: lack of error messages	Low	Acknowledged
14	DividendPayingToken: transfers are denied	Low	Acknowledged
15	IterableMapping: inserted[] not needed	Low	Acknowledged
16	General recommendations	Low	Acknowledged

`#01 BabyMatic: fees values are not limited` High

`setMATICRewardsFee()`, `setLiquiditFee()` and `setMarketingFee()` functions update fees parameters without checking new values.

Recommendation: transfer BabyMatic's ownership to a contract with limited fees in set functions.

Response from the BabyMatic team: The team has noticed this oversight, the intention of allowing these values to be changed was to adjust taxes on volume for project stability and also to allow the team the ability to provide better rewards once the liquidity pool reaches stability. The fees on buys will never exceed 15% as vehemently discussed within the team. The extra rewards adjustment will come from the Liquidity fee being adjusted only. On that note sells will never exceed 16% as per the above reasoning.

`#02 BabyMatic: excludeFromDividends() is permanent` Medium

`excludeFromDividends()` function of BabyMatic contract calls the same name function of the BABYMATICDividendTracker contract. Thus the arbitrary address could be restricted from taking the dividends as there's no inclusion of mistakenly excluded account.

`#03 BabyMatic: fees differ from website` Medium

Tokenomics section on the team's website shows different total fees and its distribution:

There's a 16% tax on both Buy and Sell

10% goes to Matic rewards	— 7% in deployed contract
1% goes to Liquidity Pool	— 3% in deployed contract
5% goes to marketing/buy back wallet	— 5% in deployed contract

Symbol: babymatic Type: BEP20

BEP20 standard [4] isn't fully supported as the token lacks the `getOwner()` function.

At the moment of the audit (2021-08-02 06:00+00), 16% tax with different distribution is taken only on sell transfers, buys take 15%.

Response from the BabyMatic team: This has been fixed, we change the tax to 10% rewards while the volume was low so that investors could get better rewards, but due to a significant uptick in volume the tax has been reverted back to how it was before and now it has been updated on the website too.

`#04 BabyMatic: Router update problem` — `Medium`

`updateUniswapV2Router()` function calls of BabyMatic contract updates `uniswapV2Router` variable and tries creating a new pair with `WETH()` of that new router. Very likely that pair would be already created at the moment of calling `updateUniswapV2Router()` and the transaction would be reverted.

`#05 BabyMatic: swapTokensForEth uses 100% slippage` — `Medium`

`swapTokensForEth()` function calls ApeRouter with 100% slippage and no deadline set. The transactions sent from this contract may be frontrun resulting in swaps with an undesired rate (sandwich attacks). Also if a transaction is sent with a small gas price it can be mined for a long time resulting in swaps with a significantly changed rate against the moment the transaction was added to the block.

`#06 BabyMatic: locked tokens` — `Medium`

`swapAndSendDividends()` function of BabyMatic contract could lock swapped amount of MATIC tokens in the contract if transfer to BABYMATICDividendTracker fails.

`#07 BabyMatic: update balances with try method` — `Medium`

`_transfer()` function of BabyMatic token calls for `dividendTracker.setBalance()` via `try` method which makes a successful transfer with unchanged balances of dividends tokens possible. The current dividendTracker implementation should not fail on setting balances. It must be noted that `dividendTracker` can be updated and in case the function `setBalance` fails, discrepancies in token balances can take place.

`#08 BabyMatic: addLiquidity() recipient` — `Medium`

`addLiquidity()` function calls for `uniswapV2Router.addLiquidityETH()` with the parameter of lp tokens recipient set to zero address. This prevents any future liquidity migrations.

#09 BabyMatic: updateDividendTracker() not excluding dead address — Medium

Function `updateDividendTracker()` does not exclude the “dead” address (`0x000000000000000000000000000000000000dEaD`) from dividends. If the “dead” address is not excluded it can lead to burning dividends.

Response from the BabyMatic team: The dead address has been excluded from all dividends and in the event that the dividend tracker needs to be updated the team has updated their workflow to ensure that this address is excluded again to avoid any situations where dividends are incorrectly burnt.

#10 BabyMatic: excessive computations — Low

`_transfer()` function performs 3 swaps instead of 2 and calls for swapAndSendToFee() function that makes 2 identical transfers to the marketing wallet.

#11 BabyMatic: hardcoded addresses — Low

Hardcoded `_devWalletAddress` address makes it harder to test contracts. Dev wallet address can’t be updated if compromised.

#12 DividendPayingToken: hardcoded addresses — Low

Hardcoded `MATIC` address leads to UpgradeableProxy contract in BSC.

#13 Multiple: lack of error messages — Low

Require statements in `BABYMATICDividendTracker:excludeFromDividends()`, `BabyMatic:_transfer()`, `DividendPayingToken:distributeMATICDividends()` functions lack error messages. SafeMathInt library contains zero specific error messages.

#14 DividendPayingToken: transfers are denied — Low

All the transfers of DividendPayingToken are blocked which makes it a non-ERC20. It may be slightly confusing as many explorers will show BABYMATIC_Dividend_Tracker as an ERC20 token.

#15 IterableMapping: inserted[] not needed — Low

IterableMapping library could save gas by getting rid of `inserted[]` mapping and use `indexOf[]` instead.

#16 General recommendations Low

We strongly recommend using original OpenZeppelin contracts as they are widely used and well audited. If some changes are needed to the original contracts, implement them via inheritance.

We recommend following Solidity naming conventions [5], i.e. UPPERCASE for constants and immutables.

BABYMATICDividendTracker constructor contains a typo in the token name.

Conclusion

1 high severity issue was found. The contracts are highly dependent on the owner's account. Users using the project have to trust the owner and that the owner's account is properly secured.

Audit includes recommendations on the code improving and preventing potential attacks.

References

1. TIKI audit
2. DividendPayingToken by Roger Wu
3. ERC-20 standard
4. BEP-20 standard
5. Solidity Docs: Naming Styles

Appendix A. Issues’ severity classification

We consider an issue critical, if it may cause unlimited losses or breaks the workflow of the contract, and could be easily triggered.

High severity issues may lead to limited losses or break interaction with users or other contracts under very specific conditions.

Medium severity issues do not cause a full loss of functionality but break the contract logic.

Low severity issues are typically nonoptimal code, unused variables, errors in messages. Usually, these issues do not need immediate reactions.

Appendix B. List of examined issue types

Business logic overview

Functionality checks

Following best practices

Access control and authorization

Reentrancy attacks

Front-run attacks

DoS with (unexpected) revert

DoS with block gas limit

Transaction-ordering dependence

ERC/BEP and other standards violation

Unchecked math

Implicit visibility levels

Excessive gas usage

Timestamp dependence

Forcibly sending ether to a contract

Weak sources of randomness

Shadowing state variables

Usage of deprecated code