DEFIMOON

be secure

Smart Contract Audit Report

April, 2023

DEFIMOON PROJECT

Audit and
Development

CONTACTS

defimoon.org
audit@defimoon.org
defimoon_org
defimoonorg
defimoon
defimoonorg

DEFIMOON

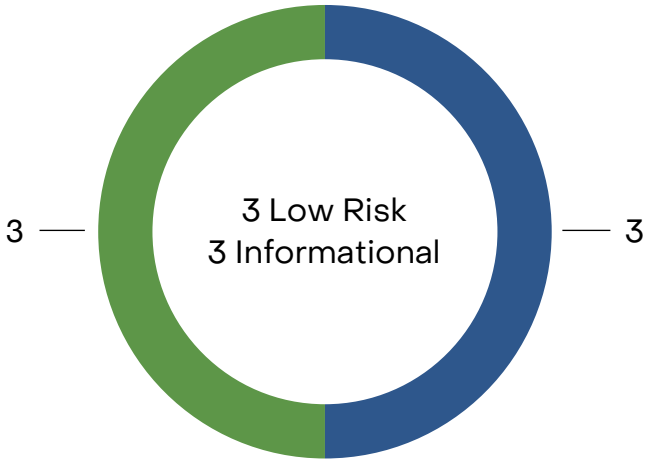
be secure

April 14th 2023

This audit report was prepared by Defimoon for VitalikCEO

Audit information

Description	VitalikCEO token contract
Audited files	Deployed BABYTOKEN contract
Timeline	13th April 2023 - 14th April 2023
Audited by	Daniil Rashin, Aleksey Zhelyabin
Approved by	Artur Makhnach, Kirill Minyaev
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Manual Review
Specification	N/A
Docs quality	N/A
Source code	0x33D469BBCEF48e556a759951F659d5620Bca2471
Network	BSC
Status	Passed



	High Risk	A fatal vulnerability that can cause the loss of all Tokens / Funds.
	Medium Risk	A vulnerability that can cause the loss of some Tokens / Funds.
	Low Risk	A vulnerability which can cause the loss of protocol functionality.
	Informational	Non-security issues such as functionality, style, and convention.

Disclaimer

This audit is not financial, investment, or any other kind of advice and could be used for informational purposes only. This report is not a substitute for doing your own research and due diligence should always be paid in full to any project. Defimoon is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Defimoon has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Defimoon is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. Defimoon has no connection to the project other than the conduction of this audit and has no obligations other than to publish an objective report. Defimoon will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Defimoon assumes that the provided information and materials were not altered, suppressed, or misleading. This report is published by Defimoon, and Defimoon has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Defimoon. In instances where an auditor or team member has a personal connection with the audited project, that auditor or team member will be excluded from viewing or impacting any internal communication regarding the specific audit.

Audit Information

Defimoon utilizes both manual and automated auditing approach to cover the most ground possible. We begin with generic static analysis automated tools to quickly assess the overall state of the contract. We then move to a comprehensive manual code analysis, which enables us to find security flaws that automated tools would miss. Finally, we conduct an extensive unit testing to make sure contract behaves as expected under stress conditions.

In our decision making process we rely on finding located via the manual code inspection and testing. If an automated tool raises a possible vulnerability, we always investigate it further manually to make a final verdict. All our tests are run in a special test environment which matches the "real world" situations and we utilize exact copies of the published or provided contracts.

While conducting the audit, the Defimoon security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Defimoon assesses the risks and assigns a risk level to each section together with an explanatory comment.

VitalikCEO Audit overview

BABYTOKEN.sol

No major issues were found.

ERC20 token with additional functionalities such as automatic liquidity provision and reflection-based yield distribution to token holders. The contract also includes a feature for swapping tokens for rewards and for automatically distributing rewards to token holders.

The contract has functions for excluding accounts from fees, setting fees, updating the gas limit for processing dividends, updating the token balance threshold for receiving dividends, and claiming dividends. It also has several internal functions for swapping tokens, adding liquidity, and distributing rewards.

BaseToken.sol

No major issues were found.

Contract defines an enumeration called TokenType, which specifies the type of token being created. The contract also includes an event called TokenCreated, which is emitted when a token is created.

BABYTOKENDividendTracker.sol

No major issues were found.

ERC20-compatible contract that tracks token holders' dividends and allows them to claim their share of the rewards periodically.

The process function is the main function that processes all token holders' dividend claims. It uses gas-unlimited while loop to optimize the gas usage and iteratively processes token holders' balances. It also tracks the gas usage, the number of iterations, and the number of claims made during the process.

Summary of findings

According to the standard audit assessment, the audited solidity smart contracts are secure and ready for production, but there are few aspects to keep in mind.

ID	Description	Severity
DFM-1	Accumulated rewards and setBalance()	Low
DFM-2	Ignored return value	Low
DFM-3	Zero address check	Low
DFM-4	Uninitialized variables	Informational
DFM-5	Lack of events on Critical State Changes	Informational
DFM-6	External functions could be public	Informational

Application security checklist

Compiler errors	Passed
Possible delays in data delivery	Passed
Timestamp dependence	Passed
Integer Overflow and Underflow	Passed
Race Conditions and Reentrancy	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Private user data leaks	Passed
Malicious Events Log	Passed
Scoping and Declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Design Logic	Passed
Cross-function race conditions	Passed

Token security checklist

Description	Status
No mint function found, owner cannot mint tokens after initial deploy	✅
Owner can't set max tx amount	✅
Owner can't set fees over 25%	✅
Owner can't pause trading	✅
Owner can't blacklist wallets	✅

Detailed Audit Information

Contract Programming

Solidity version not specified	Passed
Solidity version too old	Passed
Integer overflow/underflow	Passed
Function input parameters lack of check	Passed
Function input parameters check bypass	Passed
Function access control lacks management	Passed
Critical operation lacks event log	Passed
Human/contract checks bypass	Passed
Random number generation/use vulnerability	Passed
Fallback function misuse	Passed
Race condition	Passed
Logical vulnerability	Passed
Other programming issues	Passed

Code Specification

Visibility not explicitly declared	Passed
Variable storage location not explicitly declared	Passed
Use keywords/functions to be deprecated	Passed
Other code specification issues	Passed

Gas Optimization

Assert () misuse	Passed
High consumption ‘for/while’ loop	Passed
High consumption ‘storage’ storage	Passed
“Out of Gas” Attack	Passed
Public function could be external	Passed

Findings

DFM-1 «Accumulated rewards and setBalance»

Severity: Low risk

Description:

In the `setBalance()` function, a token holder's balance is set to zero and removed from the tokenHoldersMap mapping if the new balance is below `minimumTokenBalanceForDividends`. However, this does not prevent previously accumulated dividends from being withdrawn by the user. If this is intentional, then it is not a vulnerability, but if it is not, it may lead to a loss of protocol functionality.

Recommendation:

Make sure that this behaviour is intentional, otherwise, modify the accumulated dividends value accordingly.

DFM-2 «Ignored return value»

Severity: Low risk

Description:

In the functions below, calls to external functions do not take into account the return value, which indicates whether the function was executed successfully. Thus, the function will not respond in the event that the call was unsuccessful.

Recommendation:
It is recommended to check the return value to properly handle any outcome of the call.

```
(L3238) BABYTOKEN.swapAndSendToFee(uint256) ignores return value by
IERC20(rewardToken).transfer(_marketingWalletAddress,newBalance)

(L3128) BABYTOKEN.claim() ignores return value by
dividendTracker.processAccount(address(msg.sender),false))
```

DFM-3 «Zero address check»

Severity: Low risk

Description:

Checking if an address is not zero is a crucial step in smart contract development because sending ether or tokens to a zero address is irreversible, resulting in a permanent loss of funds. Therefore, it is important to implement this check to ensure the safety and security of your smart contract.

Recommendation:
It is recommended to add a check to prevent assigning a null address, which can lead to a failure of the logic:

```
(L2869) BABYTOKEN.constructor(string,string,uint256,address[4],uint2
56[3],uint256,address,uint256) lacks a zero-check on "addrs"

(L2872) BABYTOKEN.constructor(string,string,uint256,address[4],uint2
56[3],uint256,address,uint256) lacks a zero-check on
"serviceFeeReceiver_"
```

DFM-4 «Uninitialized variables»

Severity: Informational

Description:

The local variables listed below are declared but never used.

Recommendation:
Remove redundant variables:

```
(L3215) BABYTOKEN._transfer(address,address,uint256)
lastProcessedIndex
(L3213) BABYTOKEN._transfer(address,address,uint256)
iterations
```

DFM-5 «Lack of events on Critical State Changes»

Severity: Informational

Description:

Both the `BABYTOKEN` and `BABYTOKENDividendTracker` contracts lack any kind of events beyond the ones emitted by the used library contracts.
Events are important as they allow state changes to easily and efficiently be tracked externally.

Recommendation:
Add more events to the critical parts of code.

DFM-6 «External functions could be public»

Severity: Informational

Description:

A number of functions can be declared as external which will save some gas

Recommendation:
Change function's visibility modifier to external:

```
(L2559) isExcludedFromDividends()
(L2644) getAccountAtIndex(uint256)
(L2692) process(uint256)
(L3005) updateGasForProcessing(uint256)
(L3045) isExcludedFromFees(address)
(L3049) withdrawableDividendOf(address)
(L3057) dividendTokenBalanceOf(address)
(L3069) isExcludedFromDividends(address)
```

Adherence to Best Practices

1. Add more comments to the source code to make it easier to read through.
2. Follow the natspec format while leaving the comments.

Methodology

Manual Code Review

We prefer to work with a transparent process and make our reviews a collaborative effort. The goal of our security audits is to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, review open issue tickets, and investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system to make a final decision.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Appendix A — Finding Statuses

Closed	Contracts were modified to permanently resolve the finding
Mitigated	The finding was resolved by other methods such as revoking contract ownership or updating the code to minimize the effect of the finding
Acknowledged	Project team is made aware of the finding
Open	The finding was not addressed