QuillAudits

Audit Report
Septemeber, 2023

For

BabyDoge

Table of Content

Executive Summary

Project Name	BabyDogeCoin Report
Overview	BabyDogeCoin ChessBetting contract is intended to provide rewards to winners ascertained from an off-chain game. The contract inherits EIP 712 and Ownable contracts from the Openzeppelin library. These contracts aid in identifying the owner of the contract and the winner of the game through their hashed signature. The manager identifies when a winner signs the withdrawal signature and then the manager sends the reward to the correct winner.
Timeline	23rd August 2023 - 6th September 2023
Method	Manual Review, Functional Testing, Automated Testing, etc.
Audit Scope	The scope of this audit was to analyse BabyDogeCoin Chessbetting codebase for quality, security, and correctness. https://github.com/Baby-doge/BabyDogeChess-Contracts/commit/ fb215a885e1bc76db837b5e6ecdca6fa82794313
Fixed In	https://github.com/Baby-doge/BabyDogeChess-Contracts/commit/003d8983984359c384aa6c5189a56d08767b65e6

2
Issues Found
🟥 High
🟨 Medium
🟩 Low
🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	2

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 - ChessBetting

NOTE: Due to the employed mechanism for the execution and signing of transactions with the assistance of bots and servers, there are possibilities of users' funds getting stuck when the system halts. This prompted adding a function called emergencyWithdraw that allows users to withdraw their funds; the contract owner must set the emergencyWithdrawalsAllowed to true to allow this function call. There is also an inclusion of minimum fee strategy to ensure that gas cost is accumulated and covered for, serving as the fee charges, before the transfer is made to the winner. Although the collectFee function could be called by anyone, the accumulated fee gets sent to the assigned fee receiver for that token.

High Severity Issues

No issues found

Medium Severity Issues

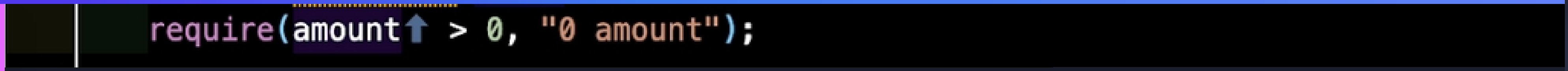
No issues found

Low Severity Issues

No issues found

Informational Issues

A.1 Use != 0 to Minimize Gas Instead of >0 for Unsigned Comparison



```
require(amount > 0, "0 amount");
```

Description

For gas optimization purposes, there are some unsigned comparisons made in the contract to ensure that the amount being sent into the contract, or withdrawn from the contract is not zero. With the use of "!=" over ">0", this saves some gas when these functions are called. - Functions affected are:

- deposit
- withdraw
- emergencyWithdraw

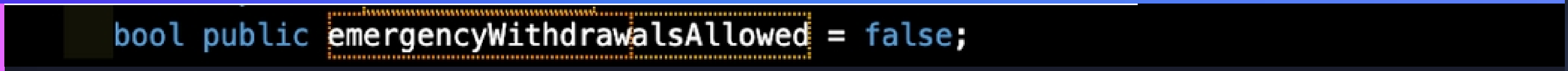
Remediation

use != 0 rather than >0.

Status

Resolved

A.2 Do Not Initialize Variables With Default Values



```
bool public emergencyWithdrawalsAllowed = false;
```

Description

State variables that are certain to be updated when called by several functions in the codebase should not be initialized and set to default values in order to save gas.

Remediation

Initialize these state variables without setting them a value since it is by default made a false variable.

Status

Resolved

Functional Tests

Some of the tests performed are mentioned below

- ✓ should get the token data of added
- ✓ should emit data of removed token
- ✓ should revert deposit with wrong token
- ✓ should allow new token in the chessbetting
- ✓ should allow user deposit for allowed tokens
- ✓ should revert if users have insufficient funds
- ✓ should revert if wrong address set manager role
- ✓ should revert if manager already exist
- ✓ should revert calling function with manager role
- ✓ should revert withdrawal for wrong account
- ✓ should revert withdrawal for invalid token account
- ✓ should revert withdrawal for outdated deadline
- ✓ should revert withdrawal for invalid amount
- ✓ should revert repeated withdrawal
- ✓ should revert setting winner with bad signature
- ✓ should revert setting winner with 2nd bad signature
- ✓ should revert setting winner with different game id
- ✓ should revert setting winner with different bet amount
- ✓ should revert setting winner with invalid token
- ✓ should revert setting winner with invalid winner
- ✓ should revert setting winner with too big bet
- ✓ should revert repeated setting winner for the same game
- ✓ Should get the withdrawal hash and the game hash
- ✓ Should calculate gas cost and ensure feeAmount does not exceed the minimum fee

Automated Tests

No major issues were found. 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 BabyDogeCoin ChessBetting. We performed our audit according to the procedure described above.

No Critical issues were Found During The Audit.

Disclaimer

QuillAudits Dapp security audit provides services to help identify and mitigate potential security risks in BabyDoge Smart Contract. 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 contract. 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 Septemeber, 2023

For

BabyDoge

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com