Smart Contract
Security Audit Report

Prepared for Babypie

Prepared by Supremacy

July 17, 2024

Contents

1 Introduction

Given the opportunity to review the design document and related codebase of the Babypie, we outline in the report our systematic approach to evaluate potential security issues in the smart contract(s) implementation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Client

Magpie XYZ is an ecosystem of DeFi protocols that provide yield and veTokenomics boosting services across multiple blockchain networks.

Babypie is a top-tier SubDAO developed by Magpie that concentrates on liquid staking services for BTC using Babylon. As a liquid staking platform for Bitcoin, Babypie allows users to stake their Bitcoin as mBTC. Created by Babypie, mBTC is a liquid staked version of BTC, enabling users to earn rewards from Bitcoin staking without any required lockup period and providing passive income opportunities across DeFi.

Item	Description
Client	Magpiexyz
Project	Babypie
Type	Smart Contract
Languages	Solidity
Platform	EVM-compatible

1.2 Audit Scope

In the following, we show the Git repository of reviewed file and the commit hash used in this security audit:

- Repository: https://github.com/magpiexyz/babypie/tree/version2/contracts

Version	Commit Hash
1	`e0dfb193af79c469c53d82af179ee6cb80aa68aa`
2	`0ff00acc6f37f0eebe2b6383ce894b610ff817fe`

And this is the commit hash after all fixes for the issues found in the first round of security audit have been checked in:

- Commit Hash: a238b46d41e2c6b8a1589aed51bd1a657b2142db

1.3 Changelogs

Version	Date	Description
0.1	July 08, 2024	Initial Draft
0.2	July 17, 2024	Release Candidate #1

1.4 About Us

Supremacy is a leading blockchain security firm, composed of industry hackers and academic researchers, provide top-notch security solutions through our technology precipitation and innovative research.

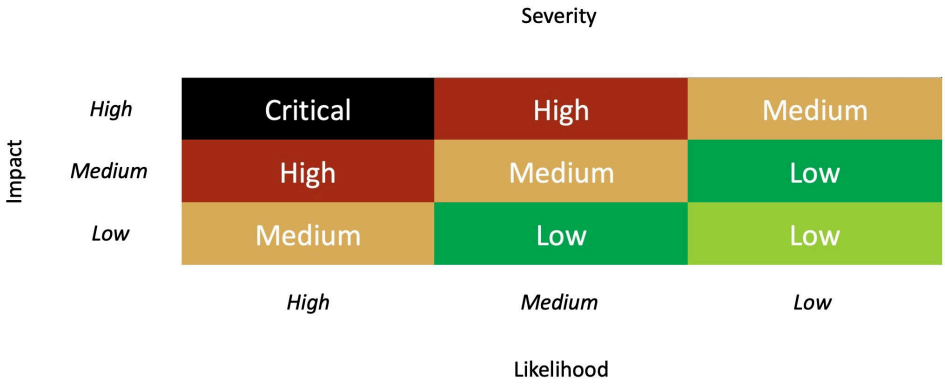
We are reachable at Twitter (https://twitter.com/SupremacyHQ), or Email (contact@supremacy.email).

1.5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- Likelihood represents the likelihood of a finding to be triggered or exploited in practice
- Impact specifies the technical and business-related consequences of a finding
- Severity is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.



As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

2 Findings

The table below summarizes the findings of the audit, including status and severity details.

ID	Severity	Description	Status
1	Low	Lack of address validation	Fixed
2	Informational	Lack of comments	Fixed
3	Informational	Follow Check-Effects-Interactions Pattern	Fixed
4	Informational	Lack of event records	Fixed

2.1 Low

1. Lack of address validation [Low]

Severity: Low Likelihood: Low Impact: Low

Status: Fixed

Description

In the `BabypieManager` contract, multiple configuration functions were missing zero address and original address validation.

```
136     /* ============ Admin Functions ================== */
137     function setmBTC(address _mBTC) external onlyOwner {
138         mBTC = _mBTC;
139     }
140 
141     function setChainlinkFunctions(address _verificationProvider, address
    _txnDataProvider) external onlyOwner {
142         verificationProvider = _verificationProvider;
143         txnDataProvider = _txnDataProvider;
144     }
145 
146     function setMagpieCustodianWallet(string calldata _walletAddress) external
    onlyOwner {
147         magpieCustodianWallet = _walletAddress;
148     }
```

BabypieManager.sol

Recommendation: Consider adding zero address validation and non-previous address validation.

2.2 Informational

2. Lack of comments [Informational]

Status: Fixed

Description

Throughout the codebase there are numerous functions missing or lacking documentation. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, comments improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Recommendation: Consider thoroughly documenting all functions (and their parameters) that are part of the smart contracts' public interfaces. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing comments, consider following the Ethereum Natural Specification Format (NatSpec).

3. Follow Check-Effects-Interactions Pattern [Informational]

`Status: Fixed`

Description

In the `BabypieManager::mintForVerifiedTransaction()`, the minting of `mBTC` does not follow the Check-Effects-Interactions Pattern.

```
121     /* ====================== Chainlink Function Callbacks ============ */
122
123     function mintForVerifiedTransaction(string calldata user, uint256 amount,
    string calldata txnHash, address referrer) external _onlyTxnDataProvider {
124
125         address userEVMAddress = userInfo[user].evmAddress;
126         if(btcTxnInfo[txnHash].isMinted)
127             revert alreadyMintedForThisTxn();
128         if(userEVMAddress == address(0))
129             revert txnOwnerNotUpdatedYet();
130
131         IMintableERC20(mBTC).mint(userEVMAddress, amount);
132         btcTxnInfo[txnHash].userBTCAddress = user;
133         btcTxnInfo[txnHash].amount = amount;
134         btcTxnInfo[txnHash].isMinted = true;
135
136         emit MintedReceiptForTxn(userEVMAddress, amount, txnHash, referrer);
137     }
```

BabypieManager.sol

Recommendation: Revise the code logic accordingly.

```
121     /* ====================== Chainlink Function Callbacks ============ */
122
123     function mintForVerifiedTransaction(string calldata user, uint256 amount,
    string calldata txnHash, address referrer) external _onlyTxnDataProvider {
124
125         address userEVMAddress = userInfo[user].evmAddress;
126         if(btcTxnInfo[txnHash].isMinted)
127             revert alreadyMintedForThisTxn();
128         if(userEVMAddress == address(0))
129             revert txnOwnerNotUpdatedYet();
130
131         btcTxnInfo[txnHash].userBTCAddress = user;
132         btcTxnInfo[txnHash].amount = amount;
133         btcTxnInfo[txnHash].isMinted = true;
134         IMintableERC20(mBTC).mint(userEVMAddress, amount);
135
136         emit MintedReceiptForTxn(userEVMAddress, amount, txnHash, referrer);
137     }
```

BabypieManager.sol

4. Lack of event records [Informational]

`Status: Fixed`

Description

In the `BabypieManager` contract, the `setmBTC()`, `setChainlinkFunctions()`, and `setMagpieCustodianWallet()` functions are missing event records. However, events are important because off-chain monitoring tools rely on them to index important state changes to the smart contract(s).

Recommendation: Always ensure that all functions that trigger state changes have event logging capabilities.

3 Disclaimer

This security audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. This security audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues, also cannot make guarantees about any additional code added to the assessed project after the audit version. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contract(s). Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.