Smart Contract Security Audit Report

Prepared for Babypie

Prepared by Supremacy

January 24, 2025

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:

Version	Repository	Commit Hash
1	Babypie	e0dfb193af79c469c53d82af179ee6cb80aa68aa
2	Babypie	0ff00acc6f37f0eebe2b6383ce894b610ff817fe
3	Babypie	c418ce8b1e439b85fe0a4c4c2efdf7ad4d0f5b96
4	Babypie	78f0e52798dafac7b25b78280bfa0619e2ba8948
5	Babypie	debea50989385b9cc422154fc9e754dd446451f6
6	Babypie	6f8fc1adec1ed1b70ad434d21910bc8d513a1de5

1.3 Changelogs

Version	Date	Description
0.1	July 08, 2024	Initial Draft
0.2	July 17, 2024	Release Candidate #1
0.3	December 06, 2024	Release Candidate #2
0.4	December 13, 2024	Release Candidate #3
0.5	January 16, 2025	Release Candidate #4

0.6	January 24, 2025	Release Candidate #5

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.

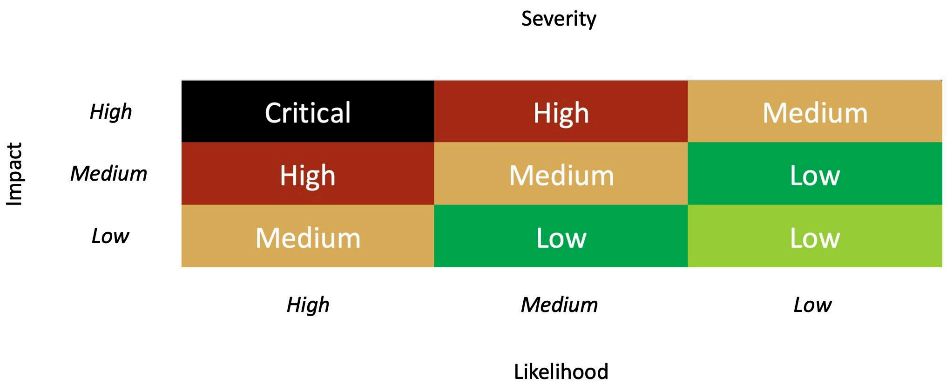
We are reachable at X (https://x.com/SupremacyHQ), or Email (contact@supremacy.email).

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.



Severity

Impact	*High*	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	*Medium*	*Low*
			Likelihood	

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	Medium	Lack of whenNotPaused modifier	Fixed
2	Medium	Missing reverse function	Fixed
3	Medium	Incorrect initialization	Fixed
4	Low	Lack of token verification	Fixed
5	Low	Lack of address validation	Undetermined
6	Informational	Lack of comments	Fixed
7	Informational	Follow Check-Effects-Interactions Pattern	Fixed
8	Informational	Lack of event records	Fixed

2.1 Medium

1. Lack of whenNotPaused modifier [Medium]

Severity: Medium Likelihood: Medium Impact: Medium

Status: Fixed

Description

Both the `verifyAndBurnMBTC()` and `verifyAndMintMBTC()` functions involve minting and burning operations. These are critical functions that should be restricted during a pause.

```
    /*//////////////////////////////////////////////////////////////
                            Write functions
    //////////////////////////////////////////////////////////////*/

    function verifyAndBurnMBTC(string calldata _txnHash) external nonReentrant {

        TxnInfo memory txn = txnInfo[_txnHash];
        address userEVMAddress = IBabypieManager(babypieManager).getUserEVMAddress(txn.userBTCAddress);

        if(txn.isMinted == false)
            revert NotMintedForThisTxn();
        if(txn.isBurnt == true)
            revert AlreadyBurnt();
        if((!txn.registered) || txn.status != 2)
            revert InvalidTxn();
        if(userEVMAddress == address(0))
            revert UserNotRegisteredOnEVMChain();

        ClientData storage clientData = allowedClients[userEVMAddress];

        IMintableERC20(mBTC).burn(clientData.mBTCWallet, txn.amount);
        clientData.mBTCMinted -= txn.amount;
        totalmBTCMinted -= txn.amount;
        txnInfo[_txnHash].isBurnt = true;

        emit BurntForValidStakeTxn(userEVMAddress, txn.userBTCAddress, _txnHash, txn.amount);

    }

    /*//////////////////////////////////////////////////////////////
                            Client functions
    //////////////////////////////////////////////////////////////*/

    function verifyAndMintMBTC(string calldata _txnHash) external nonReentrant onlyAllowedClient {

        TxnInfo memory txn = txnInfo[_txnHash];
        address userEVMAddress = IBabypieManager(babypieManager).getUserEVMAddress(txn.userBTCAddress);

        if(txn.isMinted ==true)
            revert AlreadyMintedMBTC();
        if((!txn.registered) || txn.status != 0)
            revert InvalidTxn();
        if(userEVMAddress == address(0))
            revert UserNotRegisteredOnEVMChain();
        if(userEVMAddress != msg.sender)
            revert OnlyClientCanMint();

        ClientData storage clientData = allowedClients[userEVMAddress];
        if (clientData.mBTCWallet == address(0)) {
            clientData.mBTCWallet = _deploymBTCWallet(msg.sender);
        }

        IMintableERC20(mBTC).mint(clientData.mBTCWallet, txn.amount);
        clientData.mBTCMinted += txn.amount;
        totalmBTCMinted += txn.amount;
        txnInfo[_txnHash].isMinted = true;

        emit MintedForValidStakeTxn(userEVMAddress, txn.userBTCAddress,
_txnHash, txn.amount);
    }
```

BabypieEnterprise.sol

Recommendation

To make the contract more secure and to follow the principle of least privilege, add the `whenNotPaused` modifier to any function that could alter state or make significant changes to the system. This ensures that critical actions are halted during emergencies or maintenance periods.

Feedback: Fixed at commit `5da6cd9`.

2. Missing reverse function [Medium]

`Severity: Medium`　　`Likelihood: Medium`　　`Impact: Medium`

`Status: Fixed`

Description

In the `BabypieCCIPBridge` contract, it implements an `addTokens()` function for adding cross-chain assets whitelists. However, the reverse function `delTokens()` is missing, which means that it is not possible to remove malicious tokens when the whitelist is not trustworthy.

```
  /// @dev This function will add new isValidToken.
  /// @param _tokens The array of addresses of the isValidToken.
  function addTokens(address[] calldata _tokens) external onlyOwner {
    for (uint256 i; i < _tokens.length; i++) {
      if (_tokens[i] == address(0)) {
        revert AddressZero();
      }
      if (isValidToken[_tokens[i]]) {
        revert AlreadyAdded();
      }
      isValidToken[_tokens[i]] = true;
    }
  }
```

BabypieCCIPBridge.sol

Recommendation

Consider adding this new feature.

Feedback: Fixed at commit `cb0806e`.

3. Incorrect initialization [Medium]

`Severity: Medium` `Likelihood: Medium` `Impact: Medium`

`Status: Fixed`

Description

In OpenZeppelin's upgradeable contracts, each contract that uses the `initializer` modifier (like `OwnableUpgradeable`, `PausableUpgradeable`, or `ReentrancyGuardUpgradeable`) has its own `initialize` function, and each of these functions must be explicitly called by the child contract to properly set up the parent contracts. If this step is missed, the parent contract's state variables and behavior might not be correctly initialized, which can lead to unexpected behavior, and other issues.

```
    function initialize(
        address _client,
        address _babypieEnterprise,
        address _babypieManager
    ) external initializer {

        client = _client;
        babypieEnterprise = IBabypieEnterprise(_babypieEnterprise);
        babypieManager = IBabypieManager(_babypieManager);

    }
```

MBTCWallet.sol

Recommendation

Using the OpenZeppelin upgradeable contracts, each base contract that has an `initialize` function must have its own `initialize` function called in the derived contract's `initialize` function. This ensures that the state variables of the parent contracts are properly set up during initialization. By calling the `initialize` functions for these inherited contracts, the contract's state is properly set up, ensuring that all upgradeable components work as expected.

Feedback: Fixed at commit `a656fdf`.

2.2 Low

4. Lack of token verification [Low]

`Severity: Low` `Likelihood: Low` `Impact: Low`

`Status: Fixed`

Description

In the `BabypieCCIPBridge::addTokens()` and `BabypieManagerSideChain::setSupportedToken()` function, lack of validation of token parameters is likely to lead to unintended consequences if configured incorrectly.

```
    /// @dev This function will add new isValidToken.
    /// @param _tokens The array of addresses of the isValidToken.
    function addTokens(address[] calldata _tokens) external onlyOwner {
        for (uint256 i; i < _tokens.length; i++) {
            if (_tokens[i] == address(0)) {
                revert AddressZero();
            }
            if (isValidToken[_tokens[i]]) {
                revert AlreadyAdded();
            }
            isValidToken[_tokens[i]] = true;
        }
    }
```

BabypieCCIPBridge.sol

```
    function setSupportedToken(address _token, bool _isSupportedToken) external onlyOwner {
        if(_token == address(0))
            revert AddressZero();

        isSupportedToken[_token] = _isSupportedToken;
        emit SupportedTokenSet(_token, _isSupportedToken);
    }
```

BabypieManagerSideChain.sol

Recommendation

Consider adding `isContract()` validation.

Feedback: Fixed at commit `1acc337`.

5. Lack of address validation [Low]

`Severity: Low` `Likelihood: Low` `Impact: Low`

`Status: Undetermined`

Description

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

```
    /* ============ Admin Functions =================== */
    function setmBTC(address _mBTC) external onlyOwner {
        mBTC = _mBTC;
    }

    function setChainlinkFunctions(address _verificationProvider, address _txnDataProvider) external onlyOwner {
        verificationProvider = _verificationProvider;
        txnDataProvider = _txnDataProvider;
    }

    function setMagpieCustodianWallet(string calldata _walletAddress) external onlyOwner {
        magpieCustodianWallet = _walletAddress;
    }
```

BabypieManager.sol

```
    function setZircuitStakingPool(address _zircuitStakingPoolAddress) external onlyOwner {
        zircuitStakingPoolAddress = _zircuitStakingPoolAddress;
        emit ZircuitStakingPoolSet(_zircuitStakingPoolAddress);
    }
```

MBTCWalletZircuit.sol

Recommendation

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

2.3 Informational

6. 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).

7. 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.

```
    /* ======================= Chainlink Function Callbacks ============ */

    function mintForVerifiedTransaction(string calldata user, uint256 amount, string calldata txnHash, address referrer) external _onlyTxnDataProvider {

        address userEVMAddress = userInfo[user].evmAddress;
        if(btcTxnInfo[txnHash].isMinted)
            revert alreadyMintedForThisTxn();
        if(userEVMAddress == address(0))
            revert txnOwnerNotUpdatedYet();

        IMintableERC20(mBTC).mint(userEVMAddress, amount);
        btcTxnInfo[txnHash].userBTCAddress = user;
        btcTxnInfo[txnHash].amount = amount;
        btcTxnInfo[txnHash].isMinted = true;

        emit MintedReceiptForTxn(userEVMAddress, amount, txnHash, referrer);
    }
```

BabypieManager.sol

Recommendation

Revise the code logic accordingly.

```
    /* ======================= Chainlink Function Callbacks ============ */

    function mintForVerifiedTransaction(string calldata user, uint256 amount, string calldata txnHash, address referrer) external _onlyTxnDataProvider {

        address userEVMAddress = userInfo[user].evmAddress;
        if(btcTxnInfo[txnHash].isMinted)
            revert alreadyMintedForThisTxn();
        if(userEVMAddress == address(0))
            revert txnOwnerNotUpdatedYet();

        btcTxnInfo[txnHash].userBTCAddress = user;
        btcTxnInfo[txnHash].amount = amount;
        btcTxnInfo[txnHash].isMinted = true;
        IMintableERC20(mBTC).mint(userEVMAddress, amount);

        emit MintedReceiptForTxn(userEVMAddress, amount, txnHash, referrer);
    }
```

BabypieManager.sol

8. 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.