



SMART CONTRACT AUDIT REPORT

for

Babypie Protocol



Prepared By: Xiaomi Huang

PeckShield
July 10, 2024

Document Properties

| | |
|----------------|-----------------------------|
| Client | Magpie |
| Title | Smart Contract Audit Report |
| Target | Babypie |
| Version | 1.0 |
| Author | Xuxian Jiang |
| Auditors | Jason Shen, Xuxian Jiang |
| Reviewed by | Xiaomi Huang |
| Approved by | Xuxian Jiang |
| Classification | Public |

Version Info

| Version | Date | Author(s) | Description |
|---------|---------------|--------------|----------------------|
| 1.0 | July 10, 2024 | Xuxian Jiang | Final Release |
| 1.0-rc | July 9, 2024 | Xuxian Jiang | Release Candidate #1 |

Contact

For more information about this document and its contents, please contact PeckShield Inc.

| | |
|-------|------------------------|
| Name | Xiaomi Huang |
| Phone | +86 183 5897 7782 |
| Email | contact@peckshield.com |

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 4 |
| 1.1 | About Babypie | 4 |
| 1.2 | About PeckShield | 5 |
| 1.3 | Methodology | 5 |
| 1.4 | Disclaimer | 7 |
| 2 | Findings | 9 |
| 2.1 | Summary | 9 |
| 2.2 | Key Findings | 10 |
| 3 | Detailed Results | 11 |
| 3.1 | Lack of Timelocked Enforcement in Proxy Upgrade | 11 |
| 3.2 | Redundant State/Code Removal | 12 |
| 3.3 | Trust Issue of Admin Keys | 13 |
| 4 | Conclusion | 15 |
| | References | 16 |

1 | Introduction

Given the opportunity to review the design document and related source code of the Babypie protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts could potentially be improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Babypie

Babypie is a next-generation Bitcoin staking protocol launched by Magpie with a focus on liquid staking services for BTC using Babylon. This Bitcoin staking protocol vastly increases the potential uses and engagement opportunities for BTC holders in the crypto world, including maximizing passive income potential. The basic information of audited contracts is as follows:

Table 1.1: Basic Information of Babypie

| Item | Description |
|---------------------|----------------|
| Name | Magpie |
| Type | Smart Contract |
| Language | Solidity |
| Audit Method | Whitebox |
| Latest Audit Report | July 10, 2024 |

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- <https://github.com/magpiexyz/babypie.git> (b5c662d)

And this is the commit ID after all fixes for the issues found in the audit have been addressed:

- <https://github.com/magpiexyz/babypie.git> (a238b46)

1.2 About PeckShield

PeckShield Inc. [7] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (<https://t.me/peckshield>), Twitter (<http://twitter.com/peckshield>), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification

| Impact | High | Medium | Low |
|------------|----------|--------|--------|
| | Critical | High | Medium |
| | High | Medium | Low |
| | Medium | Low | Low |
| Likelihood | | | |
| | | | |
| | | | |
| | | | |

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [6]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact, and can be accordingly classified into four categories, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further

Table 1.3: The Full List of Check Items

| Category | Check Item |
|-----------------------------|---|
| Basic Coding Bugs | Constructor Mismatch |
| | Ownership Takeover |
| | Redundant Fallback Function |
| | Overflows & Underflows |
| | Reentrancy |
| | Money-Giving Bug |
| | Blackhole |
| | Unauthorized Self-Destruct |
| | Revert DoS |
| | Unchecked External Call |
| | Gasless Send |
| | Send Instead Of Transfer |
| | Costly Loop |
| | (Unsafe) Use Of Untrusted Libraries |
| | (Unsafe) Use Of Predictable Variables |
| | Transaction Ordering Dependence |
| | Deprecated Uses |
| Semantic Consistency Checks | Semantic Consistency Checks |
| Advanced DeFi Scrutiny | Business Logics Review |
| | Functionality Checks |
| | Authentication Management |
| | Access Control & Authorization |
| | Oracle Security |
| | Digital Asset Escrow |
| | Kill-Switch Mechanism |
| | Operation Trails & Event Generation |
| | ERC20 Idiosyncrasies Handling |
| | Frontend-Contract Integration |
| | Deployment Consistency |
| | Holistic Risk Management |
| Additional Recommendations | Avoiding Use of Variadic Byte Array |
| | Using Fixed Compiler Version |
| | Making Visibility Level Explicit |
| | Making Type Inference Explicit |
| | Adhering To Function Declaration Strictly |
| | Following Other Best Practices |

deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.
- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [5], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings. Moreover, in case there is an issue that may affect an active protocol that has been deployed, the public version of this report may omit such issue, but will be amended with full details right after the affected protocol is upgraded with respective fixes.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.



Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

| Category | Summary |
|--|---|
| Configuration | Weaknesses in this category are typically introduced during the configuration of the software. |
| Data Processing Issues | Weaknesses in this category are typically found in functionality that processes data. |
| Numeric Errors | Weaknesses in this category are related to improper calculation or conversion of numbers. |
| Security Features | Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.) |
| Time and State | Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads. |
| Error Conditions, Return Values, Status Codes | Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function. |
| Resource Management | Weaknesses in this category are related to improper management of system resources. |
| Behavioral Issues | Weaknesses in this category are related to unexpected behaviors from code that an application uses. |
| Business Logics | Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application. |
| Initialization and Cleanup | Weaknesses in this category occur in behaviors that are used for initialization and breakdown. |
| Arguments and Parameters | Weaknesses in this category are related to improper use of arguments or parameters within function calls. |
| Expression Issues | Weaknesses in this category are related to incorrectly written expressions within code. |
| Coding Practices | Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained. |

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the design and implementation of the Babypie protocol smart contracts. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

| Severity | # of Findings | |
|---------------|---------------|---|
| Critical | 0 | |
| High | 0 | |
| Medium | 1 |  |
| Low | 2 |  |
| Informational | 0 | |
| Total | 3 | |

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others may involve unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in [Section 3](#).

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 medium-severity vulnerability and 2 low-severity vulnerabilities.

Table 2.1: Key Audit Findings

| ID | Severity | Title | Category | Status |
|---------|----------|---|-------------------|-----------|
| PVE-001 | Low | Lack of Timelocked Enforcement in Proxy Upgrade | Security Features | Resolved |
| PVE-002 | Low | Redundant State/Code Removal | Coding Practices | Resolved |
| PVE-003 | Medium | Trust Issue of Admin Keys | Security Features | Mitigated |

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 | Detailed Results

3.1 Lack of Timelocked Enforcement in Proxy Upgrade

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: TransparentUpgradeableproxy
- Category: Security Features [3]
- CWE subcategory: CWE-287 [1]

Description

DeFi protocols typically have a number of system-wide parameters that can be dynamically configured on demand. The Babypie protocol is no exception. Specifically, if we examine the TransparentUpgradeableproxy contract, it has defined two risk parameters, i.e., `timelockEndForUpgrade` and `timelockEndForTimelock`. In the following, we show the corresponding routines that allow for their changes.

```
35     function submitUpgrade(address newImplementation) external ifAdmin {
36         nextImplementation = newImplementation;
37         timelockEndForUpgrade = block.timestamp + timelockLength;
38     }
39     ...
40     function submitNewTimelock(uint256 _value) external ifAdmin {
41         nextTimelock = _value;
42         timelockEndForTimelock = block.timestamp + timelockLength;
43     }
```

Listing 3.1: TransparentUpgradeableproxy::submitUpgrade()

These parameters define various aspects of the protocol operation and maintenance and need to exercise extra care when configuring or updating them. Our analysis shows the `timelockEndForTimelock` risk parameter is not properly validated when the proxy is being upgrade, which undermines the intended management and operation of the proxy.

Recommendation Improve the above logic to ensure the `timelockEndForTimelock` risk parameter is properly honored and enforced.

Status The issue has been resolved as the team confirms no intention of using this custom transparent upgradeable proxy or proxy admin. Instead, they will be removed and replaced to use the OpenZeppelin's version.

3.2 Redundant State/Code Removal

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: Multiple Contracts
- Category: Coding Practices [4]
- CWE subcategory: CWE-563 [2]

Description

The Babypie protocol makes good use of a number of reference contracts, such as `OwnableUpgradeable`, `ERC20Upgradeable`, `ReentrancyGuardUpgradeable`, and `PausableUpgradeable`, to facilitate its code implementation and organization. For example, the `BabypieManager` smart contract has so far imported at least four reference contracts. However, we observe the inclusion of certain unused code or the presence of unnecessary redundancies that can be safely removed.

For example, if we examine closely the `BabypieReceiptToken` contract, it has a number of parent contracts, including `OwnableUpgradeable`, `ReentrancyGuardUpgradeable`, and `PausableUpgradeable`. However, the contract functionality does not make any use of the inherited contracts of `OwnableUpgradeable`, `ReentrancyGuardUpgradeable`, and `PausableUpgradeable`. Therefore, they can be simply removed.

```

21 contract BabypieReceiptToken is Initializable,
22     ERC20Upgradeable,
23     OwnableUpgradeable,
24     ReentrancyGuardUpgradeable,
25     PausableUpgradeable
26 {...}

```

Listing 3.2: The `BabypieReceiptToken` Contract

Similarly, the `BabypieManager` contract is a child contract of `ReentrancyGuardUpgradeable`, which can also be removed.

Recommendation Consider the removal of the redundant state (or code) with a simplified, consistent implementation.

Status The issue has been fixed by the following commits: `a91b93e` and `48b86d6`.

3.3 Trust Issue of Admin Keys

- ID: PVE-003
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: Multiple Contracts
- Category: Security Features [3]
- CWE subcategory: CWE-287 [1]

Description

The Babypie protocol has a privileged account, i.e., `owner`, that plays a critical role in governing and regulating the protocol-wide operations (e.g., configure protocol-wide risk parameters, manage oracles, and upgrade proxies). It also has the privilege to control or govern the flow of assets among various protocol components. In the following, we examine the privileged account and related privileged accesses in current contracts.

```

172     function setmBTC(address _mBTC) external onlyOwner {
173         mBTC = _mBTC;
174         emit SetmBTC(_mBTC);
175     }

177     function setChainlinkFunctions(address _verificationProvider, address
        _txnDataProvider) external onlyOwner {
178         verificationProvider = _verificationProvider;
179         txnDataProvider = _txnDataProvider;
180         emit ChainlinkfunctionsSet(_verificationProvider, _txnDataProvider);
181     }

183     function setMagpieCustodianWallet(string calldata _walletAddress) external onlyOwner
        {
184         magpieCustodianWallet = _walletAddress;
185         emit MagpieCustodianWalletSet(_walletAddress);
186     }

188     function setTxnFee(uint256 feeValue) external onlyOwner {
189         txnFeeValue = feeValue;
190         emit TxnFeeSet(feeValue);
191     }

193     function transferCollectedFeeToOwner() external onlyOwner {
194         uint256 feeCollected = address(this).balance;
195         (bool sent, ) = payable(msg.sender).call{ value: feeCollected }("");
196         if (!sent) revert FeeTransferFailed();
197         emit OwnerFeeTransferred(feeCollected);
198     }

200     function pause() public onlyOwner {
201         _pause();

```

```
202     }  
  
204     function unpause() public onlyOwner {  
205         _unpause();  
206     }
```

Listing 3.3: Example Privileged Operations in `BabypieManager`

We understand the need of the privileged functions for proper contract operations, but at the same time the extra power to these privileged accounts may also be a counter-party risk to the contract users. Therefore, we list this concern as an issue here from the audit perspective and highly recommend making these privileges explicit or raising necessary awareness among protocol users.

Moreover, it should be noted that current contracts have the support of being deployed behind a proxy. And there is a need to properly manage the proxy-admin privileges as they fall in this trust issue as well.

Recommendation Promptly transfer the `owner` privilege to the intended DAO-like governance contract. And activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status This issue has been mitigated with the use of a multisig as the admin.



4 | Conclusion

In this audit, we have analyzed the design and implementation of the `Babypie` protocol, which is a next-generation `Bitcoin` staking protocol launched by `Magpie` with a focus on liquid staking services for `BTC` using `Babylon`. This `Bitcoin` staking protocol vastly increases the potential uses and engagement opportunities for `BTC` holders in the crypto world, including maximizing passive income potential. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Meanwhile, we need to emphasize that `Solidity`-based smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.



References

- [1] MITRE. CWE-287: Improper Authentication. <https://cwe.mitre.org/data/definitions/287.html>.
- [2] MITRE. CWE-563: Assignment to Variable without Use. <https://cwe.mitre.org/data/definitions/563.html>.
- [3] MITRE. CWE CATEGORY: 7PK - Security Features. <https://cwe.mitre.org/data/definitions/254.html>.
- [4] MITRE. CWE CATEGORY: Bad Coding Practices. <https://cwe.mitre.org/data/definitions/1006.html>.
- [5] MITRE. CWE VIEW: Development Concepts. <https://cwe.mitre.org/data/definitions/699.html>.
- [6] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.
- [7] PeckShield. PeckShield Inc. <https://www.peckshield.com>.