



# SMART CONTRACT AUDIT REPORT

for

Illuvium MerkleDistributor



Prepared By: Xiaomi Huang

PeckShield  
April 27, 2023

## Document Properties

|                |                             |
|----------------|-----------------------------|
| Client         | Illuvium                    |
| Title          | Smart Contract Audit Report |
| Target         | Illuvium MerkleDistributor  |
| Version        | 1.0                         |
| Author         | Patrick Lou                 |
| Auditors       | Patrick Lou, Xuxian Jiang   |
| Reviewed by    | Xiaomi Huang                |
| Approved by    | Xuxian Jiang                |
| Classification | Public                      |

## Version Info

| Version | Date           | Author(s)   | Description   |
|---------|----------------|-------------|---------------|
| 1.0     | April 27, 2023 | Patrick Lou | Final Release |

## 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

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                        | <b>4</b>  |
| 1.1      | About Illuvium MerkleDistributor . . . . . | 4         |
| 1.2      | About PeckShield . . . . .                 | 5         |
| 1.3      | Methodology . . . . .                      | 5         |
| 1.4      | Disclaimer . . . . .                       | 7         |
| <b>2</b> | <b>Findings</b>                            | <b>9</b>  |
| 2.1      | Summary . . . . .                          | 9         |
| 2.2      | Key Findings . . . . .                     | 10        |
| <b>3</b> | <b>Detailed Results</b>                    | <b>11</b> |
| 3.1      | Trust Issue of Admin Keys . . . . .        | 11        |
| <b>4</b> | <b>Conclusion</b>                          | <b>13</b> |
|          | <b>References</b>                          | <b>14</b> |

# 1 | Introduction

Given the opportunity to review the design document and related smart contract source code of the `Illuvium MerkleDistributor`, 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 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 Illuvium MerkleDistributor

`Illuvium Staking V2` has a problem in that `Illuvium` has to manually update the staking weight of `v1`. As `stakers` withdraw, they should have a lower percentage of rewards. The team does this by running a script, and then executing a transaction in the multi-sig that sets the parameters to the value in the script. However, `stakers` will have missed out on some of their rewards for the last approximately 6 months and the `MerkleDistributor` smart contract aims to fix the gap in missing rewards.

The basic information of the audited protocol is as follows:

Table 1.1: Basic Information of The Illuvium MerkleDistributor

| Item                | Description   |
|---------------------|---|
| Name                | Illuvium  |
| Website             | <a href="https://www.illuvium.io/">https://www.illuvium.io/</a> |
| Type                | EVM Smart Contract  |
| Platform            | Solidity  |
| Audit Method        | Whitebox  |
| Latest Audit Report | April 27, 2023  |

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

- <https://github.com/lluviumGame/staking-contracts-v2/blob/main/contracts/MerkleDistributor.sol> (1774ec7)

## 1.2 About PeckShield

PeckShield Inc. [5] 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](mailto:contact@peckshield.com)).

Table 1.2: Vulnerability Severity Classification

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

## 1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [4]:

- 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 classified into four categories accordingly, 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

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            |

contract is considered safe regarding the check item. For any discovered issue, we might further 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) [3], 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.

## 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   |
|--|---|
| <b>Configuration</b>                                 | Weaknesses in this category are typically introduced during the configuration of the software.  |
| <b>Data Processing Issues</b>                        | Weaknesses in this category are typically found in functionality that processes data.   |
| <b>Numeric Errors</b>                                | Weaknesses in this category are related to improper calculation or conversion of numbers.   |
| <b>Security Features</b>                             | Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)   |
| <b>Time and State</b>                                | 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.  |
| <b>Error Conditions, Return Values, Status Codes</b> | 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.   |
| <b>Resource Management</b>                           | Weaknesses in this category are related to improper management of system resources.   |
| <b>Behavioral Issues</b>                             | Weaknesses in this category are related to unexpected behaviors from code that an application uses.   |
| <b>Business Logics</b>                               | 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.  |
| <b>Initialization and Cleanup</b>                    | Weaknesses in this category occur in behaviors that are used for initialization and breakdown.  |
| <b>Arguments and Parameters</b>                      | Weaknesses in this category are related to improper use of arguments or parameters within function calls.   |
| <b>Expression Issues</b>                             | Weaknesses in this category are related to incorrectly written expressions within code.   |
| <b>Coding Practices</b>                              | 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 `Illuvium MerkleDistributor` implementation. 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        | 0             |   |
| Low           | 0             |   |
| Informational | 1             |  |
| Total         | 1             |   |

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 refer to 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 that 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, the smart contract is well-designed and engineered, though the implementation can be improved by resolving the identified issue (shown in Table 2.1), including 1 informational suggestion.

Table 2.1: Key Illuvium MerkleDistributor Audit Findings

| ID      | Severity      | Title                     | Category          | Status    |
|---------|---------------|---------------------------|-------------------|-----------|
| PVE-001 | Informational | Trust Issue of Admin Keys | Security Features | Mitigated |

Besides recommending specific countermeasures to mitigate the issue, we also emphasize that 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 need to kick in at the very moment when the contracts are being deployed in mainnet. Please refer to Section 3 for details.



## 3 | Detailed Results

### 3.1 Trust Issue of Admin Keys

- ID: PVE-001
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: MerkleDistributor
- Category: Security Features [2]
- CWE subcategory: CWE-287 [1]

#### Description

In the MerkleDistributor contract, there is a privileged account, i.e., factory owner. The factory owner account plays a critical role in governing and regulating the system-wide operations (e.g., pause/unpause the MerkleDistributor contract, set the merkle root, etc.). Our analysis shows that this privileged account needs to be scrutinized. In the following, we show the representative functions potentially affected by the privileges of the actory owner account.

```

112     function pause(bool shouldPause) external isFactoryController {
113         if ((isPaused && shouldPause) (!isPaused && !shouldPause)) revert AlreadyPaused
114             ();
115         if (shouldPause) {
116             isPaused = true;
117             emit Paused(msg.sender);
118         } else {
119             isPaused = false;
120             emit Unpaused(msg.sender);
121         }
122     }
123
124     /**
125     * @dev Sets the yield weight tree root
126     * @notice Can only be called by the owner
127     * @param merkleRoot_ 32 bytes tree root, must be non-zero
128     */
129     function setMerkleRoot(bytes32 merkleRoot_) external isFactoryController {
130         if (merkleRoot_ == bytes32(0)) revert ZeroBytes();
131         merkleRoot = merkleRoot_;

```

```
131     emit SetMerkleRoot(msg.sender, merkleRoot_);  
132 }
```

Listing 3.1: Privileged Operations in `MerkleDistributor`

We understand the need of the privileged functions for contract maintenance, but at the same time the extra power to the owner may also be a counter-party risk to the protocol users. It would be worrisome if the privileged `factory owner` account is a plain EOA account. Note that a multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role to a community-governed DAO.

**Recommendation** Promptly transfer the privileged account to the intended DAO-like governance contract. All changes to privileged operations may need to be mediated with necessary timelocks. Eventually, 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 as the team confirms that the owner of the factory is a multi-sig wallet owned by Illuvium.



## 4 | Conclusion

In this audit, we have analyzed the `Illuvium MerkleDistributor` contract design and implementation. The current code base is well structured and neatly organized. The identified issue 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 CATEGORY: 7PK - Security Features. <https://cwe.mitre.org/data/definitions/254.html>.
- [3] MITRE. CWE VIEW: Development Concepts. <https://cwe.mitre.org/data/definitions/699.html>.
- [4] OWASP. Risk Rating Methodology. [https://www.owasp.org/index.php/OWASP\\_Risk\\_Rating\\_Methodology](https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology).
- [5] PeckShield. PeckShield Inc. <https://www.peckshield.com>.

