KGEN Airdrop

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Mon Feb 17 2025

KGEN Airdrop Audit Report

1 Executive Summary

1.1 Project Information

Description	The airdrop claim system is designed to streamline the distribution of rKGEN tokens to eligible recipients through a secure and verifiable process
Type	DeFi
Auditors	MoveBit
Timeline	Wed Feb 12 2025 - Mon Feb 17 2025
Languages	Move
Platform	Aptos
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/kgen-protocol/smartcontracts
Commits	97e39c804ec5df22a21b7f2e862546497750f755 047d3601f2c83eab744b1a868bb6e9c1145b5397

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
AIR	airdrop-rKGeN/sources/airdrop.move	911d63bc1a2cbb4329e2e9514239a6eb01aa564e

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	3	0
Informational	0	0	0
Minor	1	1	0
Medium	2	2	0
Major	0	0	0
Critical	0	0	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by KGEN to identify any potential issues and vulnerabilities in the source code of the KGEN Airdrop smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 3 issues of varying severity, listed below.

ID	Title	Severity	Status
AIR-1	Insufficient Nonce Range in `ManagedNonce` Struct	Medium	Fixed
AIR-2	`signature_verification` Lacks Sufficient Information for Checking	Medium	Fixed
AIR-3	Redundancy of User `nonce` Data Structure in `ensure_nonce` Function	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the KGEN Airdrop Smart Contract :

Admin

- `nominate_admin` : Allows the admin to nominate a new admin.
- `accept_admin_role` : Allows the nominated admin to accept their role and become the new admin.
- `update_signer_key` : Allows the admin to update the `reward_signer_key` stored in `AdminStore` by providing a new key (`vector<u8>`).

User

- `claim_reward` : Allows a user (claimer) to claim rewards by submitting the reward amount along with a cryptographic `signature` . The function constructs a signed message (including the user's address, metadata, amount, and nonce), verifies the signature, and if valid, transfers the reward from the resource account to the user. It then emits a `Transfer` event and updates the nonce.

4 Findings

AIR-1 Insufficient Nonce Range in `ManagedNonce` Struct

Severity: Medium

Status: Fixed

Code Location:

airdrop-rKGeN/sources/airdrop.move#46

Descriptions:

```
 /// Manages user-specific counters (nonces) to prevent duplication
 struct ManagedNonce has key {
     nonce: vector<simple_map::SimpleMap<address, u8>> // A mapping of contract
addresses to nonces
 }
```

The `ManagedNonce` struct stores nonces using u8, which has a maximum value of 255. Once a address reaches this limit, any attempt to increment the nonce will result in an overflow error.

Suggestion:

To prevent nonce overflow, replace u8 with u64 in the ManagedNonce struct.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

AIR-2 `signature_verification` Lacks Sufficient Information for Checking

Severity: Medium

Status: Fixed

Code Location:

airdrop-rKGeN/sources/airdrop.move#237

Descriptions:

The `SignedMessage` structure does not include a `chainID`, which means that under certain conditions, a signature from the test environment could also be valid in the mainnet.
On the other hand, there is also a lack of `Deadline` in the structure, which means that even a valid signature can be intercepted by an attacker if it is not used, and the nonce is correct but can still be valid.

Suggestion:

It is recommended to add the `chainID` field in `SignedMessage` to distinguish different environments and prevent potential cross-chain signature replay.
In addition, add `Deadline` to complete the operation within a defined time frame (for example, like the time limit for reward collection in the current contract). If the deadline is exceeded, the operation should be invalid to reduce the risk of long-term signature abuse.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

AIR-3 Redundancy of User `nonce` Data Structure in `ensure_nonce` Function

Severity: Minor

Status: Fixed

Code Location:

airdrop-rKGeN/sources/airdrop.move

Descriptions:

In the ensure_nonce function, when recording nonce values for different metadata_addresses for each user, a design scheme of storing multiple simple_map<address, u8> in a vector is adopted. There is obvious redundancy and complexity in the recording, which not only increases the level of the storage structure, but also brings additional traversal and update overhead. For each user, it is only necessary to record the nonce value corresponding to each metadata_address, which is essentially a key-value pair relationship. Directly using a simple_map<address, u8> (where the key is metadata_address and the value is nonce) can meet the needs without nesting it in a vector array.

```
  inline fun ensure_nonce(user: &signer, metadata_address: &address): u8 acquires
ManagedNonce {
        let n = 0;
        if (!exists<ManagedNonce>(signer::address_of(user))) {
            let v = vector::empty<simple_map::SimpleMap<address, u8>>();
            move_to(user, ManagedNonce { nonce: v });
        };
        let managed_nonce = borrow_global_mut<ManagedNonce>
(signer::address_of(user));
        vector::for_each(
            managed_nonce.nonce,
            |s| {
                if (simple_map::contains_key(&s, metadata_address)) {
                    n = *simple_map::borrow(&s, metadata_address);
                }
            }
        );
        if (n == 0) {
            let s = simple_map::new<address, u8>();
            simple_map::add(&mut s, *metadata_address, 0);
            vector::push_back(&mut managed_nonce.nonce, s);
        };
        n
    }
```

Suggestion:

It is recommended to simplify it to a single simple_map<address, u8> data structure, which can not only reduce code complexity, but also improve operation efficiency and reduce potential error risks.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.