KGeN

Preliminary Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Mon Aug 11 2025

MOVEBIT VER
MOVEB

KGeN Preliminary Audit Report

1 Executive Summary

1.1 Project Information

Description	KGEN token implementation on Aptos that follows the latest fungible asset standard, with features for freezing, minting, burning, and admin control
Type	DeFi
Auditors	MoveBit
Timeline	Mon Aug 04 2025 - Mon Aug 11 2025
Languages	Move
Platform	Aptos
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/kgen-protocol/smartcontracts
Commits	df69ae101540f1cf2a5743ebc566b2d0c859eccd ec2b2fe424541cd7250e02d11e60ec04d2386fd7

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV8	KGeN-Token/Move.toml	8d874597d752f5f9f9a5a485036321b73183a533
KGE1	KGeN-Token/sources/kgen.move	d76f9ebd2462f42ec9f1ce0cfc38065b3d78acf7
KGE1	KGeN-Token/APTOS/sources/kgen.move	0bedfcad525ae7f548493b12cb395974e1367db3

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	3	0
Informational	0	0	0
Minor	0	0	0
Medium	3	3	0
Major	0	0	0
Critical	0	0	0
Discussion	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 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
KGE-1	Mint Function That Doesn't Check The Contract's Paused State	Medium	Fixed
KGE-2	The `transfer_store` Function Lacks A Check For The Frozen Status Of The Recipient Account	Medium	Fixed
KGE-3	Logic Flaw In Address Removal May Delete The First Vector Element By Mistake	Medium	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the KGeN Smart Contract :
Admin:

- **add_treasury_address:** Add authorized treasury addresses for managing funds.
- **remove_treasury_address:** Remove authorized treasury addresses for managing funds.
- **add_minter:** Add minter addresses with permission to mint new tokens.
- **remove_minter_address:** remove minter addresses with permission to mint new tokens.
- **freeze_accounts:** Freeze specified accounts'sending and receiving capabilities to restrict token transfers.
- **unfreeze_accounts:** Unfreeze specified accounts to restore their token transfer capabilities.
- **transfer_admin:** Nominate a new admin to initiate role transfer.
- **accept_admin:** The nominated address accepts and assumes the admin role.
- **set_pause:** Pause or resume critical contract functions during emergencies.
- **update_burn_vault:** Update the address of the burn vault where tokens are destroyed.
- **transfer / transfer_store:** Transfer tokens on behalf of frozen accounts.
- **burn / burn_store:** Burn tokens from specified accounts or stores.
- **mutate_project_and_icon_uri:** Update project metadata including project URI and icon URI.

User:

- **deposit:** Deposit tokens into the account's asset store.
- **withdraw:** Withdraw tokens from the account's token storage.

Minter:

- **mint:** Mint new tokens to the given treasury address and deposit them into the treasury's asset storage.

4 Findings

KGE-1 Mint Function That Doesn't Check The Contract's Paused State

Severity: Medium

Status: Fixed

Code Location:

KGeN-Token/APTOS/sources/kgen.move#261

Descriptions:

The `mint` function does not call `assert_not_paused()` to check the paused state, allowing tokens to be minted even when the contract is in Paused mode.

```
//mint function

    let management = get_kgen_management_ref();
    management.assert_is_minter(&address_of(minter));
    management.assert_is_treasury(&to);
    let primary_store =
        primary_fungible_store::ensure_primary_store_exists(to, metadata());
    management.assert_not_frozen(to, false, true);
    let tokens = fungible_asset::mint(&management.mint_ref, amount);
    fungible_asset::deposit_with_ref(
            &management.transfer_ref, primary_store, tokens
      );
```

This undermines the integrity of the pause mechanism and may introduce governance control risks and potential attack vectors.

Suggestion:

Add the following line to the `mint` function to ensure the operation is restricted by the `paused` state:

```
// mint function

 let management = get_kgen_management_ref();
 management.assert_is_minter(&address_of (minter));
 management.assert_is_treasury(&to);
 management.assert_not_paused(); // <--add
```

Resolution:

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

KGE-2 The `transfer_store` Function Lacks A Check For The Frozen Status Of The Recipient Account

Severity: Medium

Status: Fixed

Code Location:

KGeN-Token/APTOS/sources/kgen.move#340-348

Descriptions:

The `transfer_store` function allows transfers from the specified asset storage of a frozen account under administrator operations, but it only checks the frozen status of the **sender account (** `from_store` **)** and does not verify the frozen status of the **recipient account (** `to_store` **)**.

```
// transfer_store function

  let management = get_kgen_management_ref();
  management.assert_is_admin(admin);
// Check that the sender account is frozen
  management.assert_is_frozen(object::owner(from_store));
  let to_store = primary_fungible_store::ensure_primary_store_exists(to, metadata());
// Missing a check for whether `to_store` (the recipient account) is frozen
  fungible_asset::transfer_with_ref(&management.transfer_ref, from_store, to_store,
amount);
```

This allows frozen accounts to still receive assets, undermining the integrity of the freeze mechanism.

Suggestion:

Add a check for the frozen status of the recipient account within the function.

```
management.assert_not_frozen(to_store, false, true);
```

Resolution:

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

KGE-3 Logic Flaw In Address Removal May Delete The First Vector Element By Mistake

Severity: Medium

Status: Fixed

Code Location:

KGeN-Token/APTOS/sources/kgen.move#478-482;

KGeN-Token/APTOS/sources/kgen.move#500-504

Descriptions:

In the `remove_treasury_address` and `remove_minter_address` functions, `vector::index_of()` is used to search for the target address. However, the `index_of()` function is defined in `atops-core` as follows:

```
/// Return `(true, i)` if `e` is in the vector `self` at index `i`.
/// Otherwise, returns `(false, 0)`.
  public fun index_of<Element>(self: &vector<Element>, e: &Element): (bool, u64) {
      let i = 0;
      let len = self.length();
      while (i < len) {
          if (self.borrow(i) == e) return (true, i);
          i += 1;
      };
      (false, 0)
  }
```

https://github.com/aptos-labs/aptos-core/blob/4c39d3a0274b40655b589e5b6ee57b676abea149/aptos-move/framework/move-stdlib/sources/vector.move#L217-L230

In both the `remove_treasury_address` and `remove_minter_address` functions, the returned boolean value is not checked.

```
 // remove_treasury_address function
      let (_, i) = management.treasury_vec.index_of(&treasury_addr);
      management.treasury_vec.remove(i);

// remove_minter_address function
      let (_, i) = management.minter_vec.index_of(&minter_addr);
      management.minter_vec.remove(i);
```

This causes the first address to be incorrectly removed if `index_of()` returns `(false, 0)`.

Suggestion:

Add a boolean check after using `index_of` to ensure the address exists in the vector before calling `.remove(i)`.

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.