KGEN Staking

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Mon Feb 17 2025

KGEN Staking Audit Report

1 Executive Summary

1.1 Project Information

Description	The staking contract is responsible to streamline smooth staking process where user can stake rkGeN tokens, harvest rkGeN after 24 hrs from last harvest and claim after staking duration has been completed.admin is responsible to create & manage apy table and its details. To make sure staking is done through KGeN platform, authorised platform signature is included
Type	Staking
Auditors	MoveBit
Timeline	Sun Feb 09 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
RKGNS	staking-rKGeN/sources/rKGeN_staking.move	4483d3c418bf10d5bb8b7f941179146fdd4cec4d

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	6	4	2
Informational	0	0	0
Minor	2	1	1
Medium	2	1	1
Major	2	2	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 Staking 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 6 issues of varying severity, listed below.

ID	Title	Severity	Status
RKG-1	Potential Integer Overflow in `get_available_rewards` Calculation	Major	Fixed
RKG-2	`harvet` Function Time Check Error	Major	Fixed
RKG-3	Missing Existence Check Before Removal in `remove_platform` Function	Medium	Fixed
RKG-4	The Claim Time is Ambiguous and does not Align with the Tntended Logic	Medium	Acknowledged
RKG-5	Unused Code	Minor	Fixed
RKG-6	The `get_available_rewards` Function Calculates the Risk	Minor	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the KGEN Staking 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.
- `add_platform` : Adds a platform to the list of allowed platforms.
- `remove_platform` : Removes a platform from the list of allowed platforms.
- `manage_apy_table` : Updates the APY table with new ranges for staking.
- `update_apy` : Updates the APY value for a specific range in the APY table.

User

- `add_stake` : Adds a stake for the user with a specified amount and duration. It verifies the platform, validates the input duration, fetches the applicable APY range, and then initializes or updates the user's stake record with the provided details. Finally, it transfers the staked amount from the user to the resource account.
- `harvest` : Harvests rewards from a specific stake. It verifies the platform, checks that at least 24 hours have passed since the last harvest and that the current time is within the staking duration, calculates the available reward (total rewards earned minus rewards already claimed), transfers the current reward to the user, and updates the stake's last harvest time and total claimed rewards.
- `claim` : Claims the full payout (staked amount plus unclaimed rewards) from a specific stake once the staking duration has completed. It verifies the platform, ensures the staking period is over, calculates the total payout, transfers the payout to the user, updates the stake's records, and removes the stake from the user's stake list.

4 Findings

RKG-1 Potential Integer Overflow in `get_available_rewards` Calculation

Severity: Major

Status: Fixed

Code Location:

staking-rKGeN/sources/rKGeN_staking.move#774

Descriptions:

```
let total_rewards_earned = amount * stake_apy__ * passed_time__
    / (seconds_in_year * 100);
```

The function `get_available_rewards` calculates rewards using u64 arithmetic:

When `passed_time__ = 365 * 24 * 60 * 60` (1 year) and `stake_apy__ = 1`, the maximum possible amount without overflow is 584,942,417,355.

Considering that amount should also include precision, if the staked amount is large and the staking duration is long, an integer overflow may occur, leading to a Denial of Service (DoS).

Suggestion:

Use u256 for calculations to prevent overflow.

Resolution:

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

RKG-2 `harvet` Function Time Check Error

Severity: Major

Status: Fixed

Code Location:

staking-rKGeN/sources/rKGeN_staking.move

Descriptions:

In harvest, the pledge end time is not calculated based on start_time, but stake.duration * 24 * 60 * 60 is directly used. The execution timestamp is much larger than this value.

The root cause is that the start time is not taken into account.

```
//  ensure the current time is within the staking duration
assert!(
    current_time <= (stake.duration * 24 * 60 * 60),
    error::invalid_argument(EHARVEST_DURATION_OVER)
);
```

Suggestion:

It is recommended to modify the code to:

```
assert!(
    current_time <= stake.start_time + (stake.duration * 24 * 60 * 60),
    error::invalid_argument(EHARVEST_DURATION_OVER)
);
```

Resolution:

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

RKG-3 Missing Existence Check Before Removal in `remove_platform` Function

Severity: Medium

Status: Fixed

Code Location:

staking-rKGeN/sources/rKGeN_staking.move#318

Descriptions:

```
public entry fun remove_platform(admin: &signer, platform: address) acquires Admin
{
    assert!(
        signer::address_of(admin) == get_admin(),
        error::permission_denied(E_NOT_ADMIN)
    );

    let platform_list = &mut borrow_global_mut<Admin>(@KGeNAdmin).platform_list;

    // Ensure the platform address is not empty
    assert!(
        platform != @0x0,
        error::invalid_argument(EINVALID_ARGUMENT)
    );

    let (_exists, index) = smart_vector::index_of(platform_list, &platform);

    // Remove the platform from the list using its index
    smart_vector::remove(platform_list, index);

    event::emit(
        PlatformRemovedEvent {
            admin_address: signer::address_of(admin),
            platform_address: platform
        }
    );

}
```

In the `remove_platform` function, the code retrieves the index of the platform to be removed using `smart_vector::remove`.
However, the function does not check whether the platform actually exists before attempting removal. If the platform does not exist, index_of will return (false, 0), and the function will proceed to remove the element at index 0, potentially deleting an unintended platform from the list.

Suggestion:

Before calling `smart_vector::remove`, add a check to ensure the `platform` exists in the list. This can be done by verifying the `_exists` value returned.

Resolution:

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

RKG-4 The Claim Time is Ambiguous and does not Align with the Tntended Logic

Severity: Medium

Status: Acknowledged

Code Location:

staking-rKGeN/sources/rKGeN_staking.move#594

Descriptions:

The user can call the `claim()` function to claim the rewards and the staked amount once the staking duration is completed. This means that the current time should satisfy the condition:

```
current_time > stake.start_time + (stake.duration * 24 * 60 * 60)
```

However, the protocol currently allows users to claim when:

```
current_time == stake.start_time + (stake.duration * 24 * 60 * 60)
```

This implementation might lead to potential issues, as the exact moment when `current_time` equals the end of the staking period could be ambiguous or misaligned with the intended logic.

Suggestion:

It is recommended to strictly enforce the condition where `current_time` must be **greater than** the staking end time to ensure clarity and avoid edge cases.

RKG-5 Unused Code

Severity: Minor

Status: Fixed

Code Location:

staking-rKGeN/sources/rKGeN_staking.move#501

Descriptions:

Unused constants defined in the contract.

```
const EINVALID_MIN_AMOUNT_SEQUENCE: u64 = 5;
```

Unused or unnecessary comments.

```
// Debug message to confirm the new stake addition
```

Invalid update In the `claim` function, the `stake` record will be deleted at the end, so it is meaningless to update the `stake` time `update_last_harvest_time` and the total reward of the current `stake` , because it has been deleted by `remove_stake` after the call, and this record does not exist in `harvest/claim` .

```
update_total_claimed(stake, current_reward);
update_last_harvest_time(stake, current_time);
```

Suggestion:

It is recommended to remove unused comments.

Resolution:

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

RKG-6 The `get_available_rewards` Function Calculates the Risk

Severity: Minor

Status: Acknowledged

Code Location:

staking-rKGeN/sources/rKGeN_staking.move

Descriptions:

The time period is not properly constrained.

The calculation of rewards in the `get_available_rewards` function depends on `current_time`, not `endtime`. The user's staked interest will increase independently of `duration`.

`total_rewards_earned` is calculated as `passed_time = current_time - start_time;`, if the user continues to stake after the period ends (30 days period at apy=10%) and does not call the `claim` function, for example (apy=5%), then the user will bypass the required re-staking execution flow and continue to earn interest at a higher or lower apy.

Suggestion:

It is recommended to modify the pledge validity period, such as automatically paying interest after the end time, or calculating interest according to the maximum validity period.

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.