Security Audit Report

Gnogen gng-minting
smart contract

by Arda

on May 12, 2023

Table of Content

Audit Summary **3**

Code issues & Recommendations **4**
C1: Users can’t withdraw and claim if battle can’t finish 4
C2: No guarantee that rewards can be claimed 6
C3: Changing NFT internal attributes can prevent some withdrawals 8
C4: Users who participated in battle with 0 power can’t withdraw 9
C5: Can’t run battles if operator rewards are 0 10
C6: User can have infinite power without staking 11
C7: Changing total rewards leads unexpected rewards for past battles 12
C8: State activation in init can have unexpected consequences 13
C9: No protection against past first_battle_timestamp 14
C10: No protection against SFT deposits 15
C11: Unnecessary reward calculations and constant DIVISION_PRECISION 16
C12: The zero address unexpectedly owns all withdrawn NFTs 17
C13: Misleading name daily_reward_amount 18

Disclaimer **19**

Audit Summary

Scope of initial review

- **Repository:** https://github.com/horizonnlabs/sc-gnogen-gng-minting-rs
- **Commit:** `0ffba0ef91e8a8186fe34dbd4ef0a9819605d58c`
- **Path to Smart contract:** `./`

Scope of final review

- **Repository:** https://github.com/horizonnlabs/sc-gnogen-gng-minting-rs
- **Commit:** `97ed8f71de13a3362acf90f264ada24bd49a83f1`
- **Path to Smart contract:** `./`

Report objectives

1. Reporting all issues in smart contract **code** alongside recommendations
2. Reporting all issues in smart contract **test** alongside recommendations
3. Reporting all **other** issues alongside recommendations

Issues

Number of issues reported in the initial review and remaining in the final review:

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	0	0	0	0	0	0
Major	3	0	0	0	0	0
Medium	5	0	0	0	0	0
Minor	5	0	0	0	0	0

Code issues & Recommendations

C1: Users can’t withdraw and claim if battle can’t finish

Severity: Major **Status:** Fixed

Description

Users can’t withdraw their NFTs if the smart contract is in `Battle` status. The smart contract might not exit this status if admins don't provide enough rewards for battles to finish (see e.g. No guarantee that rewards can be claimed) or if the `battle` endpoint fails (see e.g. Users who participated in battle with 0 power can’t withdraw).

Moreover, even if no such issue happens, the user would still have to run all the clashes if he really wants to withdraw and could pay a significant amount of gas for it. Users should be able to withdraw their NFTs in any circumstance, without assuming the proper behaviour of admins or having to execute additional transactions.

Recommendation

We recommend deleting the requirement that the smart contract status is `Preparation` from the endpoints `withdraw` and `claim`.

Because the list of NFTs `battle_stack` does not have fixed length anymore while the smart contract is in `Battle` status, we make a few modifications.

We remove the storage `unique_id_battle_stack`. Instead, we add a storage `remaining_nfts_in_battle -> u32`, which indicates that the NFTs of `battle_stack` that should still fight each other in the current battle are those with indices in `[1, remaining_nfts_in_battle]`. More precisely:

- When a battle starts, `remaining_nfts_in_battle` is initialised to `battle_stack.len()`.
- In `get_and_remove_token_from_stack`, we sample the NFT index in `[1, remaining_nfts_in_battle]`. Then, we permute the NFT in `battle_stack` with the NFT at index `remaining_nfts_in_battle`. This can be done using

the method `swap_indexes` of `UnorderedSetMapper`. Finally, we decrement `remaining_nfts_in_battle` by 1.

In `withdraw`, before removing an NFT from `battle_stack`, we get its index `battle_stack(nft).get_index()`. We then consider two cases:

- If the NFT index is greater than `remaining_nfts_in_battle`, we do nothing.
- Else, we permute the NFT in `battle_stack` with the NFT at index `remaining_nfts_in_battle`, and decrease `remaining_nfts_in_battle` by 1.

Finally, to calculate the operator's rewards in `calculate_clash_operator_rewards`, we add a storage `total_number_clashes_current_battle -> u32`, which is set to `battle_stack.len() / 2` when a battle starts.

We also recommend adding a test witnessing that this mechanism allows a user to withdraw even while a battle is ongoing.

C2: No guarantee that rewards can be claimed

Severity: Major **Status:** Fixed

Description

When a new battle starts, there is no guarantee that rewards deposited by admins are sufficient for users and clash operators to claim.

In addition, if there are not enough rewards in the smart contract for users to claim, then users won't be able to withdraw their NFTs (see Users can't withdraw and claim if battle can't finish).

Recommendation

We recommend to add a condition that a new battle can start only if `reward_capacity` is sufficient for rewarding both users and clash operators for this battle. More precisely, we would have the following changes:

- Adding a new storage `total_rewards_for_stakers(battle_id: u64) -> BigUint` which is set to `get_daily_reward_amount_with_halving` when a new battle starts. This is the total amount of rewards to be distributed for users in a single battle.
- Adding a new storage `total_rewards_for_clash_operators(battle_id: u64) -> BigUint` which is set to `daily_battle_operator_reward_amount` when a new battle starts. This is the total amount of rewards to be distributed for clash operators in a single battle.

Then, when a new battle starts, `battle` subtracts `total_rewards_for_stakers(battle_id) + total_rewards_for_clash_operators(battle_id)` from `reward_capacity`. In particular this would fail if `reward_capacity` is too small, indicating that rewards in the smart contract are insufficient. The only other location where `reward_capacity` should be used is in `deposit_gng` where it is increased when admins deposit rewards. We should remove other occurrences of `reward_capacity` (e.g. in `claim_rewards` or in the `run_while_it_has_gas` loop of `battle`).

Finally, we use `total_rewards_for_stakers(battle_id)` to compute users rewards for a specific battle. Similarly, we use `total_rewards_for_clash_operators(battle_id)` to compute clash operators rewards for a specific battle.

C3: Changing NFT internal attributes can prevent some withdrawals

Severity: Major **Status:** Fixed

Location

```
src/config.rs
    set_attributes
```

Description

The owner endpoint `set_attributes` changes the internal attributes of a specific NFT. This can prevent users to withdraw their NFTs.

Example:

- Alice stakes an NFT with power 10. Her power is 10.
- The owner increases the power of that NFT to 16.
- Alice tries to withdraw her NFT, which fails when trying to decrease Alice's power by 16 since it would lead to a negative value.

Recommendation

We recommend removing the `power` field from `UserStats`, as it is not used by the smart contract. The user's total power can be computed directly in the front-end.

Alternatively, if it is desirable to keep this storage, we can add the following in `set_attributes` in case `nft_owner(token_id, nonce)` contains a non-empty user address:

- We get the current power of the NFT: `get_token_attributes(token_id, nonce).power`.
- We remove that power from the total power of the user.
- We set the new attributes.
- We add the new power of the NFT to the total power of the user.

C4: Users who participated in battle with 0 power can't withdraw

Severity: Medium **Status:** Fixed

Location

```
src/lib.rs
    calculate_clash_rewards
```

Description

`calculate_clash_rewards` fails because of a division by 0 in case the total users power for a specific `battle_id` is 0. Therefore users whose NFTs were in that battle will never be able to withdraw or claim.

Example: Assume that a battle `battle_id` contains 10 NFTs, 5 with power 0 are fighting 5 other NFTs with non-zero power. The battle happens on a Sunday, so the 5 NFTs with power 0 win all the clashes and the total users power for the battle is 0. Therefore `calculate_clash_rewards` fails each time it is called for this specific battle `battle_id`, and consequently:

- Each winner of `battle_id` will never be able to withdraw any of his NFTs.
- Each winner of `battle_id` will never be able to claim rewards.
- All subsequent battles fail each time a clash is won by one winner of `battle_id`, preventing all users to earn rewards for these battles.

Moreover, as long as the smart contract can't exit the `Battle` status, users can't withdraw their NFTs or claim rewards (see Users can't withdraw and claim if battle can't finish).

Recommendation

We suggest that `calculate_clash_rewards` returns 0 if the total users power corresponding to `battle_id` is 0.

C5: Can’t run battles if operator rewards are 0

Severity: Medium **Status:** Fixed

Location

```
src/lib.rs
    battle
```

Description

If `daily_battle_operator_reward_amount` is 0, users are still incentivised to run the `battle` endpoint to earn rewards from clashes. However, `battle` will fail when calling `direct_esdt` with an amount of 0 ESDT. Therefore, the battle can’t terminate and users will not be able to earn rewards from clashes.

Moreover, the smart contract can’t exit the `Battle` status and consequently users can’t withdraw their NFTs or claim rewards (see Users can’t withdraw and claim if battle can’t finish).

Recommendation

In `battle`, we recommend to send rewards to the caller only if they are non-zero.

C6: User can have infinite power without staking

Severity: Medium **Status:** Fixed

Location

`src/lib.rs`

Description

`stake` and `withdraw` modify the total power of the user based on the NFTs staked or withdrawn. This is done by incrementing a variable `total_power: u16`, which for large quantities of NFTs can lead to overflow. This can be exploited by a user to make his total power much bigger than other users.

Example: Alice has enough NFTs to reach a total power of `u16::MAX + 1 = 65536`.

- Alice calls `stake` several times to stake all her NFTs. Her total power is `65536`.
- Alice withdraws all her NFTs at once, which due to `u16` overflow reduces her power by 0. So her total power remains `65536`.
- Alice repeats the previous steps an arbitrary number of times to make her total power much higher than other users.

Recommendation

As recommended in Changing NFT internal attributes can prevent some withdrawals, We suggest removing the `power` field from `UserStats` because it is not used in the smart contract.

Alternatively, if it is desirable to keep this storage, we recommend that `total_power` in `stake` and `withdraw` has type `u64` to prevent the overflow.

C7: Changing total rewards leads unexpected rewards for past battles

Severity: Medium **Status:** Fixed

Location

```
src/config.rs
    set_daily_reward_amount
```

Description

If the owner changes the total user rewards for one battle, this can make users who did not yet claim for past battles to earn less rewards than expected.

Example: Assume that the total user rewards for one battle are 2000 GNG. Alice and Bob both stake 1 NFT with the same power.

- There are 4 NFTs involved in the 1st battle. The NFTs of Alice and Bob both win their respective clashes.
- Alice claims and earns 1000 GNG.
- The owner calls `set_daily_reward_amount` and sets the total user rewards for one battle to 200 GNG.
- Bob claims and earns 100 GNG.

To sum up, Bob won 10x less rewards than Alice although he expected to earn the same amount.

Recommendation

We recommend adding a new storage `total_rewards_for_stakers(battle_id: u64) -> BigUint` which is set to `get_daily_reward_amount_with_halving` when a new battle starts (see No guarantee that rewards can be claimed).

Then, `calculate_clash_rewards(battle_id)` can calculate a user's rewards for a specific battle based on `total_rewards_for_battle(battle_id)` instead of `daily_reward_amount`.

C8: State activation in init can have unexpected consequences

Severity: Medium **Status:** Fixed

Location

```
src/lib.rs
    init
```

Description

`init` sets the state of the smart contract to `State::Active` by default, which can be unexpected and unwanted. For instance, at a smart contract upgrade, if the proper functioning of the smart contract first requires the owner to call some `#[owner]` endpoints, then user interactions should no be allowed in the meantime.

Recommendation

We suggest not to set the state in `init`, and instead to let the owner explicitly call the `pause` and `resume` endpoints to change the state.

C9: No protection against past first_battle_timestamp

Severity: Minor **Status:** Fixed

Location

`src/lib.rs`
`init`

Description

`first_battle_timestamp` is set once at deployment and can't be modified later on. If by mistake it is set to a e.g. 30 days in the past, then users won't be able to stake before we perform the last 30 battles with no participants.

Recommendation

We suggest checking that `first_battle_timestamp` is bigger than `current_timestamp`.

C10: No protection against SFT deposits

Severity: Minor **Status:** Fixed

Location

`src/lib.rs`
`stake`

Description

Even if the owner doesn't intend to authorise SFTs in smart contract, it is safer to explicitly protect the smart contract of such an event. This is because if an SFT collection is authorised, then users who stake multiple SFTs with the same nonce would not be able to withdraw.

Recommendation

We recommend to make `stake` fail if `nft_owner(token_id, nonce)` is non-empty and contains an address distinct from the zero address.

C11: Unnecessary reward calculations and constant DIVISION_PRECISION

Severity: Minor **Status:** Fixed

Location

`src/lib.rs`

Description

To perform rewards calculations, `calculate_clash_rewards` and `calculate_clash_operator_rewards` use a constant `DIVISION_PRECISION = 1000000` and perform the calculation `power.mul(DIVISION_PRECISION).div(total_power).mul(rewards).div(DIVISION_PRECISION)`.

However simply doing `power * rewards / total_power` gives a more accurate result and removes the need of `DIVISION_PRECISION`.

Recommendation

We suggest using the calculation `power * rewards / total_power` and removing the constant `DIVISION_PRECISION`.

C12: The zero address unexpectedly owns all withdrawn NFTs

Severity: Minor **Status:** Fixed

Location

`src/lib.rs`
`withdraw`

Description

When a user withdraws an NFT, no one should hold this NFT in the smart contract anymore. However, the storage `nft_owner(token_id, nonce)` is set to `ManagedAddress::zero()`, indicating that a valid address on the blockchain is the owner of that NFT.

Recommendation

We suggest to clear the storage `nft_owner(token_id, nonce)` instead of setting it to the zero address.

C13: Misleading name daily_reward_amount

Severity: Minor **Status:** Fixed

Location

`src/config.rs`

Description

The storage `daily_reward_amount` and method `get_daily_reward_amount_with_halving` refer to the rewards for one battle, and give a false impression that a battle lasts one day although it can have arbitrary duration, e.g. it can last 5 minutes or 2 months.

Recommendation

We suggest changing the names, e.g. to `battle_rewards_amount` and `get_battle_reward_amount_with_halving`.

Disclaimer

The security audit report makes no statements or warranties, either expressed or implied, regarding the security of the code, the information herein or its usage. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract.

This report does not constitute legal or investment advice. It is for informational purposes only and is provided on an "as-is" basis. You acknowledge that any use of this report and the information contained herein is at your own risk. The authors of this report shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.