Claynosaurz Solana

Security Assessment

February 24th, 2025 — Prepared by OtterSec

Xiang Yin soreatu@osec.io

Gabriel Ottoboni ottoboni@osec.io

Kevin Chow kchow@osec.io

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Claynosaurz engaged OtterSec to assess the `staking-smart-contract` program. This assessment was conducted between January 28th and February 18th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 10 findings throughout this audit engagement.

In particular, we identified a vulnerability where the staking function lacks proper validation of the NFT's collection, allowing an attacker to stake an NFT with a matching collection key but without the required verified status, bypassing collection integrity checks (OS-CSZ-ADV-00). Additionally, the admin account is not marked as mutable in god mode, preventing the deduction of lamports from the admin account and the transfer of excess lamports to it (OS-CSZ-ADV-03). Furthermore, if the admin misconfigures the creation or modification of a class with an incorrect token mint record, the multiplier may not be applied correctly, resulting in an underflow which will prevent users from unstaking their tokens (OS-CSZ-ADV-05).

We also made recommendations to ensure adherence to coding best practices (OS-CSZ-SUG-02) and suggested modifying the codebase for improved efficiency (OS-CSZ-SUG-01). Additionally, we advised verifying that the expiry time is in the future to prevent invalid multipliers and optimize the transfer CPI call by executing it only when additional lamports are needed (OS-CSZ-SUG-00).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/Claynosaurz-Inc/staking-smart-contract. This audit was performed against commit b47a8df.

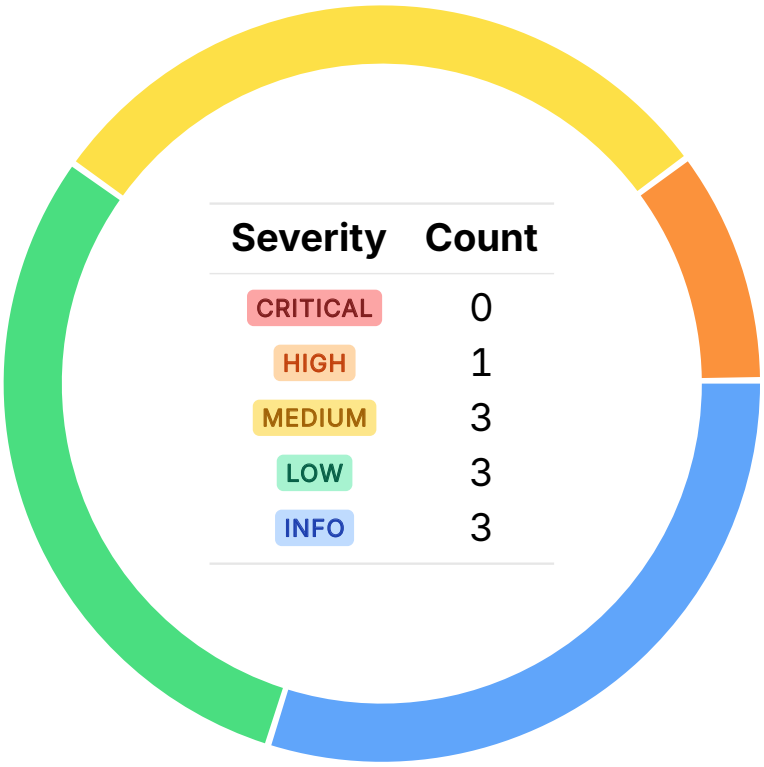
A brief description of the program is as follows:

Name	Description
staking-smart-contract	A staking and rewards system for the Claynosaurz NFT collection on Solana. It allows users to stake and unstake their assets, earn points or experience over time, and increase their NFT levels through a structured rewards mechanism.

03 — Findings

Overall, we reported 10 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	0
HIGH	1
MEDIUM	3
LOW	3
INFO	3

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-CSZ-ADV-00	HIGH	RESOLVED	The staking function lacks proper validation of the NFT's collection, allowing an attacker to stake an NFT with a matching collection key but without the required verified status, bypassing collection integrity checks.
OS-CSZ-ADV-01	MEDIUM	RESOLVED	`calculate_points_for_level` does not enforce `MAX_LEVEL`, allowing users to exceed the intended cap of 25. Also, due to unchecked point accumulation, the actual level cap reaches 256, enabling unintended over-leveling.
OS-CSZ-ADV-02	MEDIUM	RESOLVED	Currently, multipliers are added or removed without updating the user's points, leading to incorrect point calculations, particularly for users with `last_claimed` set to zero.
OS-CSZ-ADV-03	MEDIUM	RESOLVED	The `admin` account is not marked as mutable in `GodMode`, preventing the deduction of lamports from the `admin` account and the transfer of excess lamports to it.
OS-CSZ-ADV-04	LOW	RESOLVED	While updating points, the logic currently utilizes `size_of` to calculate the size of `EphemeralMultiplier` structure, which may not return the correct account size because of padding.

OS-CSZ-ADV-05	LOW	RESOLVED	If the admin misconfigures `create_class` or `modify_class` with an incorrect token mint record, the multiplier may not be applied correctly, resulting in an underflow which will prevent users from unstaking their tokens.
OS-CSZ-ADV-06	LOW	RESOLVED	`add_ephemeral_multiplier` resizes the account after transferring lamports, which may result in issues due to a known Solana bug.

Bypassing of NFT Collection Integrity Checks HIGH OS-CSZ-ADV-00

Description

`staking_action::stake` fails to properly verify the collection associated with the staked NFT. Currently, the function checks whether the `collection.key` field in the `nft_metadata` matches a predefined collection address (`CLAYNO_COLLECTION_ADDRESS`). However, this validation is insufficient, as it only checks the `key` field of the `collection` structure but it overlooks the `verified` field. The `verified` field ensures that the `collection` has been officially validated by the authority responsible for the `collection` .

>_ *claynosaurz-staking/src/instructions/staking_action.rs* RUST

```rust
/// Stakes an NFT by delegating it to the global authority PDA.
pub fn stake(ctx: Context<StakingAction>) -> Result<()> {
    [...]
    // Deserialize Metadata to verify collection
    let nft_metadata = Metadata::safe_deserialize(&mut
        ↪  ctx.accounts.nft_metadata.to_account_info().data.borrow_mut()).unwrap();
    if let Some(collection) = nft_metadata.collection {
        if collection.key.to_string() != CLAYNO_COLLECTION_ADDRESS {
            return Err(error!(StakingError::WrongCollection));
        }
    } else {
        return Err(error!(StakingError::InvalidMetadata));
    };
    [...]
}
```

This may only be set to true if the authority has run one of the Token Metadata Verify instructions over the NFT. Thus, only verifying the `key` field enables an attacker to create an NFT with a `collection.key` that matches the expected `CLAYNO_COLLECTION_ADDRESS`. Consequently, the NFT appears valid even if it is not really part of the intended collection. This means that it is possible for an attacker to create a fake NFT with the correct `key` but without having the `verified` field set to true, which implies the collection is not validated, enabling the attacker to stake an invalid NFT.

Remediation

Ensure that `collection.verified` field is set to true along with verifying that `collection.key` matches the mint address of the appropriate collection parent.

Patch

Fixed in 0b87d6e.

Uncapped Level Progression MEDIUM OS-CSZ-ADV-01

Description

`state::calculate_points_for_level` does not properly enforce `MAX_LEVEL`, allowing users to potentially accumulate enough experience points to exceed the intended maximum level. The function only handles the case where `level == MAX_LEVEL`, returning a fixed `LEVEL_25_POINTS`. It does not return an error if `level > MAX_LEVEL`, which implies the function may calculate required points for any level, even beyond the intended cap. This allows users to continue leveling up indefinitely as long as they accumulate enough points. Also, due to point accumulation, users may theoretically reach level 256, which is significantly higher than the existing cap of 25.

>_ *claynosaurz-staking/src/state.rs* RUST

```rust
/// Calculates the points required for a given level.
fn calculate_points_for_level(&self, level: u8) -> Result<u64> {
    if level == MAX_LEVEL {
        return Ok(LEVEL_25_POINTS);
    }
    [...]
}
```

Remediation

Enforce the `MAX_LEVEL` limit to prevent over-leveling and also adjust the value of `MAX_LEVEL` to be consistent with the point calculation.

Patch

Fixed in 0b87d6e.

Failure to Update Multiplier MEDIUM OS-CSZ-ADV-02

Description

`add_multiplier` and `remove_multiplier` in `godmode` do not call `update_points`. This may result in situations where the user's `last_claimed` timestamp is not updated correctly when multipliers are added or removed. As a result, when `update_points` is eventually called, it will calculate points based on the multiplier value that was present since the `last_claimed` time. This issue is particularly problematic for users who have never staked, as their `last_claimed` timestamp is set to zero. In such cases, calling `update_points` after adjustments will grant these users an unexpectedly high number of points, as the calculation will span from the Unix epoch.

>_ claynosaurz-staking/src/state.rs RUST

```rust
pub fn update_points(&mut self, current_time: i64, staking_account: &AccountInfo) -> Result<()>
    ↪ {
    [...]
    // Calculate points for the active period with combined multiplier
    let active_time = current_time
        .checked_sub(self.last_claimed)
        .ok_or(StakingError::Underflow)? as u64;
    [...]
}
```

Remediation

Ensure that whenever a multiplier is added or removed (via `add_multiplier` or `remove_multiplier`), the `last_claimed` timestamp is updated accordingly.

Patch

Fixed in 0b87d6e.

Inability to Modify Admin Account MEDIUM OS-CSZ-ADV-03

Description

In `GodMode`, the `admin` account handles lamport transfers when resizing the staking account. However, the `transfer` CPI call in `add_ephemeral_multiplier` attempts to deduct lamports from the `admin` account, while `reclaim_rent` (shown below) transfers excess lamports back to it. Since the `admin` account is currently read-only, these operations will fail—preventing lamport deductions in `add_ephemeral_multiplier` and blocking modifications in `reclaim_rent`.

>_ *claynosaurz-staking/src/instructions/admin/godmode.rs* RUST

```rust
/// Reclaims rent from unused space in the staking account.
pub fn reclaim_rent(ctx: Context<GodMode>) -> Result<()> {
    [...]
    // Verify if the data_len is less than the current data_len, if so, resize the account
    if account_info.lamports() > minimum_balance {
        **ctx.accounts.admin.to_account_info().try_borrow_mut_lamports()? +=
            ↪  account_info.lamports() - minimum_balance;
        **account_info.try_borrow_mut_lamports()? = minimum_balance;
    }
    Ok(())
}
```

Remediation

Mark the `admin` account as `mut`.

Patch

Fixed in 0b87d6e.

Incorrect Account Size Calculation LOW OS-CSZ-ADV-04

Description

In `state::update_points`, `std::mem::size_of::<EphemeralMultiplier>` is utilized in the calculation of the new size for the staking account when reallocating. While `std::mem::size_of::<EphemeralMultiplier>` returns the actual size of the `EphemeralMultiplier` structure in memory, this may not always be the size that the contract expects for the account to function properly, especially if there is padding between structure fields for alignment purposes.

>_ *claynosaurz-staking/src/state.rs* RUST

```rust
/// Updates the points based on the current time and active multipliers.
pub fn update_points(&mut self, current_time: i64, staking_account: &AccountInfo) -> Result<()>
    ↪ {
    [...]
    // Realloc account to new size if needed
    if self.ephemeral_multiplier.len() > active_multipliers.len() {
        let new_size = StakingData::INIT_SPACE + active_multipliers.len() *
            ↪ std::mem::size_of::<EphemeralMultiplier>();
        staking_account.realloc(new_size, true)?;
    }
    self.ephemeral_multiplier = active_multipliers;
    Ok(())
}
```

Remediation

Replace `std::mem::size_of::<EphemeralMultiplier>` with `EphemeralMultiplier::INIT_SPACE` to get the correct space allocation.

Patch

Fixed in 0b87d6e.

Possibility of Underflow Due to Misconfiguration LOW OS-CSZ-ADV-05

Description

`create_class` and `modify_class` do not correctly guarantee that an NFT's multiplier is properly updated. The admin can supply an incorrect `token_mint_record`, which results in skipping the necessary updates to the class metadata. In `staking_actions::unstake`, the `current_multiplier` is adjusted based on the `class_pda`. If an incorrect `token_mint_record` is utilized by the admin when calling `create_class` or `modify_class` and the multiplier is not properly applied, it may result in an underflow issue in `unstake` if the multiplier is not updated, as `staking_account.current_multiplier` may be less than `class.multiplier`.

>_ *claynosaurz-staking/src/instructions/staking_action.rs* RUST

```rust
/// Unstakes an NFT by revoking the delegation and unlocking the NFT.
pub fn unstake(ctx: Context<StakingAction>) -> Result<()> {
    // Update staking data
    let staking_account = &mut ctx.accounts.staking_account;
    [...]
    // Adjust multiplier based on class PDA ownership (default to 1 if no class PDA)
    if let Ok(class) = Class::try_deserialize(&mut
        ↪  &ctx.accounts.class_pda.to_account_info().data.borrow_mut()[..]) {
        staking_account.current_multiplier = staking_account.current_multiplier
            .checked_sub(class.multiplier)
            .ok_or(StakingError::Overflow)?;
    }
    [...]
}
```

As a result, the function will attempt to subtract a larger multiplier than what was originally assigned, resulting in an underflow, triggering the `StakingError::Overflow error`. Consequently, users will be unable to unstake their tokens.

Remediation

Ensure that `create_class` and `modify_class` perform strict validation on `token_mint_record` before modifying multipliers.

Patch

Fixed in 0b87d6e.

Inconsistency in Solana Realloc LOW OS-CSZ-ADV-06

Description

The current implementation of `add_ephemeral_multiplier` in `godmode` dynamically resizes the staking account utilizing Solana's `realloc` function after transferring additional lamports (if needed). However, there is a known issue with Solana that affects this approach.

>_ *claynosaurz-staking/src/instructions/admin/godmode.rs* RUST

```rust
pub fn add_ephemeral_multiplier(ctx: Context<GodMode>, multiplier: u8, expiry_time: i64) ->
    ↪  Result<()> {
    [...]
    if data_len > staking_account.to_account_info().data_len() {
        let new_minimum_balance = Rent::get()?.minimum_balance(data_len);
        let lamports_diff =
            ↪  new_minimum_balance.saturating_sub(staking_account.to_account_info().lamports());
        transfer(
            CpiContext::new(
                ctx.accounts.system_program.to_account_info(),
                Transfer {
                    from: ctx.accounts.admin.to_account_info(),
                    to: staking_account.to_account_info(),
                },
            ),
            lamports_diff,
        )?;

        staking_account.to_account_info().realloc(data_len, false)?;
    }
    Ok(())
}
```

Remediation

Call `realloc` before the transfer occurs in the `if` block. This ensures `realloc` is executed when the existing balance is still valid. Alternatively, utilize Anchor's `realloc` constraint instead of the Solana function. This may be achieved by only resizing the account in `add_ephemeral_multiplier` and `remove_ephemeral_multiplier`. Also, since `update_points` will no longer resize the account, `reclaim_rent` will become redundant and may be removed.

Patch

Fixed in 0b87d6e.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-CSZ-SUG-00	We advised to ensure `expiry_time` is set in the future to prevent invalid multipliers and optimize the transfer CPI call by executing it only when additional lamports are needed.
OS-CSZ-SUG-01	Recommendation for modifying the codebase for improved efficiency.
OS-CSZ-SUG-02	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.

Missing Validations OS-CSZ-SUG-00

Description

Currently, `add_ephemeral_multiplier` does not validate whether the new multiplier's `expiry_time` is set in the future. If `expiry_time` is in the past or equal to the current time, the multiplier will immediately expire, rendering it useless. Also, the function always calls the transfer instruction, even when no additional `SOL` is required. If the account already has sufficient lamports, calling the transfer is unnecessary.

Remediation

Add a check to ensure the new multiplier `expiry_time > current_time` and call the transfer CPI only if `lamports_diff > 0`.

Code Refactoring OS-CSZ-SUG-01

Description

As `calculate_points_for_level` will never overflow within `u128`, it may be more efficient to simplify the calculations with a predefined constant array (`POINTS_FOR_LEVEL`) which directly holds the points required for each level.

>_ example RUST

```
pub const POINTS_FOR_LEVEL: [u64; 26] = [0, 2560, 40960, 207360, 655360, 1600000, 3317760,
    ↪  6146560, 10485760, 16796160, 25600000, 37480960, 53084160, 73116160, 98344960,
    ↪  129600000, 167772160, 213813760, 268738560, 333621760, 409600000, 497871360, 599695360,
    ↪  716392960, 849346560, 1000000000];
```

Remediation

Incorporate the above refactors into the codebase.

Code Maturity OS-CSZ-SUG-02

Description

1. Avoid calling `update_points` twice in `godmode::remove_ephemeral_multiplier` because it is already called inside `reclaim_rent`.

>_ claynosaurz-staking/src/instructions/admin/godmode.rs RUST

```rust
/// Removes all ephemeral multipliers from the staking account.
pub fn remove_ephemeral_multiplier(ctx: Context<GodMode>) -> Result<()> {
    [...]
     // Update current points
    let account_info = staking_account.to_account_info();
    staking_account.update_points(Clock::get()?.unix_timestamp, &account_info)?;
    [...]
    // Reclaim rent
    reclaim_rent(ctx)?;
    Ok(())
}
```

2. In `initialize`, remove the unnecessary `.clone()` and utilize `ctx.accounts.staking_account.owner` directly to emit the event, as the `staking_data` variable is already utilized to set the `staking_account` data, rendering cloning unnecessary.

>_ claynosaurz-staking/src/instructions/initialize.rs RUST

```rust
/// Initializes a new staking account with default values.
pub fn initialize(ctx: Context<Initialize>) -> Result<()> {
    [...]
    // Set the staking account's inner data to the newly created StakingData
    ctx.accounts.staking_account.set_inner(staking_data.clone());

    // Emit an event indicating the creation of a new staking account
    emit!(StakingAccountCreated {
        owner: staking_data.owner,
        timestamp: Clock::get()?.unix_timestamp,
    });

    Ok(())
}
```

Remediation

Implement the above-mentioned suggestions.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.