ADDICTEDDOTFUN
Security Review

SEPTEMBER, 2025

www.chaindefenders.xyz
https://x.com/DefendersAudits

Lead Auditors

PeterSR

0x539.eth

Table of Contents

Protocol Summary

Addicted is a Solana-based decentralized farming simulation that integrates advanced tokenomics, a multi-level referral system, and Switchboard's VRF multicall pattern for verifiable randomness. It features a deflationary WEED token and a halving mechanism, alongside a dynamic farming system with 10 levels and 12 seed types. Security measures include comprehensive input validation, integer overflow protection, reentrancy prevention via Anchor, and VRF-specific safeguards like ownership verification and fee bounds.

Disclaimer

The Chain Defenders team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the implementation of the contracts.

Risk Classification

Likelihood/Impact	High	Medium	Low
High	H	H/M	M
Medium	H/M	M	M/L
Low	M	M/L	L

Audit Details

PR

PR 2

PR 3

Scope

Id	Files in scope
1	lib.rs
2	state.rs
3	error.rs
4	admin.rs
5	user.rs
6	farm.rs
7	seeds.rs
8	referral.rs
9	operator.rs
10	grow_powers.rs
11	invite.rs
12	economic_validation.rs
13	user_validation.rs
14	system_validation.rs
15	vrf_validation.rs
16	game_validation.rs
17	economics.rs
18	utils.rs
19	constants.rs

Roles

Id	Roles
1	Admin
2	Operator
3	User

Executive Summary

Issues found

Severity	Count	Description
High	0	Critical vulnerabilities
Medium	0	Significant risks
Low	0	Minor issues with low impact
Informational	6	Best practices or suggestions
Gas	0	Optimization opportunities

Findings

Informational

Info 01 Redundant `msg!` Logging

Location

admin.rs

Description

The `toggle_admin_referral_codes` and `toggle_player_referral_codes` functions in `admin.rs` each use multiple `msg!` macros to log the result of a state change.

For example, in `toggle_admin_referral_codes`:

1. A `msg!` call logs whether the status is “ENABLED” or “DISABLED”.
2. An `if/else` block follows, which logs a more descriptive message for the exact same status change.

This pattern is repeated in `toggle_player_referral_codes`. While not a security risk, this redundancy increases the transaction’s compute unit consumption and adds unnecessary noise to the program logs.

Recommendation

It is recommended to remove the redundant logging. For each function, you can keep either the initial `msg!` macro or the more descriptive messages within the `if/else` block, but not both. This will slightly optimize gas usage and improve log clarity.

For instance, you could remove the initial `msg!` call in both functions.

```
// ... existing code ...
pub fn toggle_admin_referral_codes(ctx: Context<ToggleAdminReferralCodes>) -> Result<()> {
    let config = &mut ctx.accounts.config;

    config.admin_referral_codes_enabled = !config.admin_referral_codes_enabled;

    // @audit-find: Info, redundant `msg!` logging in both toggle_admin_referral_codes and toggle_player_referral_codes. Consider keeping either the first in both functions or the second one.
    if config.admin_referral_codes_enabled {
        msg!("✅ Admin-created pre-registered codes can now be used. Status: ENABLED");
    } else {
        msg!("❌ Admin-created pre-registered codes are disabled. Status: DISABLED");
    }

    Ok(())
}

/// **Toggle Player Referral Codes (Admin Only)**
// ... existing code ...
// ... existing code ...
// ... existing code ...
pub fn toggle_player_referral_codes(ctx: Context<TogglePlayerReferralCodes>) -> Result<()> {
    let config = &mut ctx.accounts.config;

    config.player_referral_codes_enabled = !config.player_referral_codes_enabled;

    if config.player_referral_codes_enabled {
        msg!("✅ Players can now use referral codes to invite others. Status: ENABLED");
    } else {
```

```
29          msg!("⚠️ Player referral codes are disabled. Status: DISABLED")
        ;
30      }
31
32      Ok(())
33  }
34  // ... existing code ...
```

Status

Fixed

Info 02 Non Reasonable Upper Limit For `daily_pack_limits`

Location

farm.rs

Description

In the `update_farm_level_config` function within `farm.rs`, there is a validation loop for the `daily_pack_limits` input. A comment within this loop, `// Validate each limit is reasonable (1-255)`, indicates the intention to check that each daily limit is reasonable.

The code correctly implements a lower-bound check with `require!(limit > 0, GameError::InvalidConfig);`. It also implements an upper bound check due to the `limit` variable being a `u8` type, which has a natural maximum of 255. However, 255 is not a reasonable upper limit for `daily_pack_limits`

Recommendation

To align the implementation with the documented intent and enforce a sensible maximum for daily pack purchases, an additional explicit upper-bound check should be added to the validation loop. A reasonable upper limit, such as `50`, can be defined as a constant and used in the check.

```
// ... existing code ...
    // Update daily pack limits if provided
    if let Some(limits) = daily_pack_limits {
        // Validate daily pack limits
        require!(limits.len() == max_unlocked_level as usize, GameError::InvalidConfig);

        for (i, &limit) in limits.iter().enumerate() {
            require!(limit > 0 && limit ⩽ crate::constants::MAX_DAILY_PACK_LIMIT, GameError::InvalidConfig);
            // Ensure ascending order (each level should have higher or equal limit)
            if i > 0 {
                require!(limit ⩾ limits[i-1], GameError::InvalidConfig);
            }
        }

        // Update daily pack limits for all levels up to max_unlocked_level
// ... existing code ...
```

You would also need to define `MAX_DAILY_PACK_LIMIT` in your `constants.rs` file.

```
// ... existing code ...
pub const MAX_DAILY_PACK_LIMIT: u8 = 50;
// ... existing code ...
```

Status

Acknowledged

Info 03 Redundant Seconds In A Day Variables

Location

seeds.rs

user.rs

Description

In the `purchase_seed_pack` function located in `seeds.rs` and `view_daily_pack_purchase` located in `user.rs`, a local variable `seconds_in_day` is defined with the hardcoded value of `86400`. This value is used to determine if a user's daily seed pack purchase counter should be reset.

While this is functionally correct, the project maintains a dedicated `constants.rs` file for such values to ensure consistency and improve maintainability. In fact, the `constants.rs` file already contains a constant `SECONDS_IN_DAY` for this exact purpose. Using a locally defined "magic number" is redundant and goes against the DRY (Don't Repeat Yourself) principle.

Recommendation

It is recommended to remove the local definition of `seconds_in_day` from the mentioned functions and instead use the global constant already defined in `constants.rs`. This change will improve code readability and make future modifications easier, as the value will be managed in a single, centralized location.

Status

Fixed

Info 04 Missing Events Emission

Location

admin.rs

Description

The `admin.rs` :: `toggle_admin_referral_codes` function updates the configuration state by flipping the `admin_referral_codes_enabled` boolean flag. It logs status messages using `msg!` but does not emit an event.

The same applies to the `admin.rs` :: `toggle_player_referral_codes` function.

Recommendation

Update the functions to emit a structured event whenever the status changes. Example:

```
#[event]
pub struct AdminReferralCodesToggled {
    pub enabled: bool,
}

pub fn toggle_admin_referral_codes(ctx: Context<
    ToggleAdminReferralCodes>) -> Result<()> {
    let config = &mut ctx.accounts.config;

    config.admin_referral_codes_enabled = !config.
    admin_referral_codes_enabled;

    emit!(AdminReferralCodesToggled {
        enabled: config.admin_referral_codes_enabled
    });

    Ok(())
}
```

This way, off-chain consumers can subscribe to events instead of parsing logs. Apply the same pattern to `toggle_player_referral_codes`.

Status

Fixed

Info 05 Purchase Limit Not Always Applicable

Location

farm.rs

Description

The `daily_pack_purchase_count` tracks how many packs a user has purchased within the last 24 hours, with an intended daily purchase limit. However, the enforcement is flawed.

Example scenario:

- `daily_pack_purchase_count = 0`
- The limit is set to `10`
- A user purchases 1 pack to start his counter
- Time passes and the counter will reset in 1 minute
- A user purchases 9 packs immediately
- After 1 minute, the counter resets and the user can purchase another 10 packs

As a result, the user ends up buying 19 packs within a 24-hour window, effectively bypassing the intended daily cap.

Recommendation

The enforcement logic should be redesigned so that the limit applies to a true rolling 24-hour window per user, rather than resetting at a fixed time. This ensures that no more than the maximum allowed packs can be purchased in any given 24-hour period.

Status

Acknowledged

Info 06 `farm.rs` Constants Should Be Moved

Location

farm.rs

Description

Currently, the constants `ALL_CAPACITIES`, `ALL_THRESHOLDS`, and `ALL_DAILY_LIMITS` are defined directly within `farm.rs`. This mixes core logic with configuration-like values, reducing maintainability and making it harder to manage or reuse constants across modules.

Recommendation

Move the constants `ALL_CAPACITIES`, `ALL_THRESHOLDS`, and `ALL_DAILY_LIMITS` from `farm.rs` into `constants.rs`.

Status

Fixed