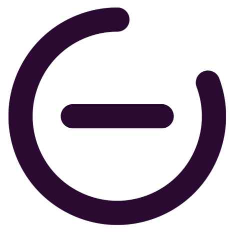


FOMOSolana

Audit

Presented by:

OtterSec contact@osec.io

Akash Gurugunti sud0u53r.ak@osec.io

Tamta Topuria tamta@osec.io

Contents

01 Executive Summary **2**
Overview . . . 2
Key Findings . . . 2

02 Scope **3**

03 Findings **4**

04 Vulnerabilities **5**
OS-FOMO-ADV-00 [high] | Disparity In Rewards Update . . . 6
OS-FOMO-ADV-01 [high] | Missing Referrer Validation . . . 8
OS-FOMO-ADV-02 [low] | Rounding Error . . . 9

05 General Findings **10**
OS-FOMO-SUG-00 | Overflow Check . . . 11
OS-FOMO-SUG-01 | Code Refactoring . . . 12
OS-FOMO-SUG-02 | Unreachable Error Code Blocks . . . 13
OS-FOMO-SUG-03 | Code Inconsistencies . . . 14
OS-FOMO-SUG-04 | Inconsistent Team Allocations . . . 16

Appendices

A Vulnerability Rating Scale **17**

B Procedure **18**

© 2023 Otter Audits LLC. All Rights Reserved.

01 | Executive Summary

Overview

FOMOSolana engaged OtterSec to assess the `fomo-game` program. This assessment was conducted between December 11th and December 20th, 2023. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 8 findings throughout this audit engagement.

In particular, we identified several high-risk vulnerabilities including the failure to update the total amount for a user with the sidepot rewards (OS-FOMO-ADV-00) and another issue where the referral code creation check is missing in the buy ticket instruction, allowing unintended referrer assignments without fee payment validation (OS-FOMO-ADV-01). Additionally, we highlighted a rounding error, where the jackpot amount and burned amount are rounded down resulting in their sum not being equal to the total amount (OS-FOMO-ADV-02).

We also provided recommendations regarding removing unnecessary calculations of team amounts during the initial phase of the game and the need for inclusion of calculations of user's share from the players amount on chain (OS-FOMO-SUG-01). We further advised the removal of unreachable and redundant error blocks to enhance code readability (OS-FOMO-SUG-02). Furthermore, we suggested specific code modifications to address certain inconsistencies (OS-FOMO-SUG-03).

© 2023 Otter Audits LLC. All Rights Reserved.

02 | Scope

The source code was delivered to us in a git repository at github.com/Doge-Capital/FOMO-GAME. This audit was performed against commit d9c7639.

A brief description of the programs is as follows:

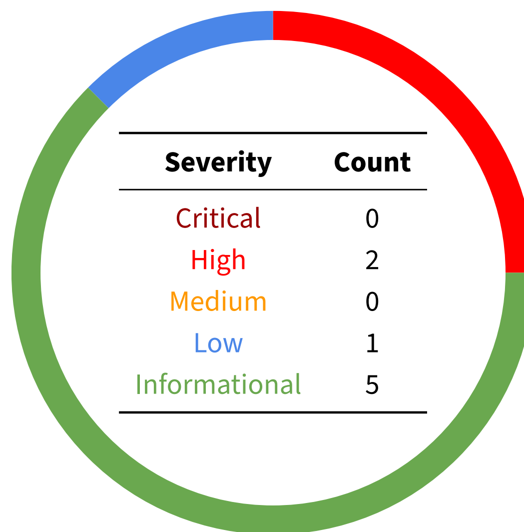
Name	Description
fomo-game	A Solana-based game where players compete to be the last to purchase a key before a 24-hour countdown reaches zero. Key prices increase with each purchase, and players strategically select teams, create referral codes, and contribute to various pots, including a jackpot and side pot, to maximize their chances of winning.

© 2023 Otter Audits LLC. All Rights Reserved.

03 | Findings

Overall, we reported 8 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	2
Medium	0
Low	1
Informational	5

© 2023 Otter Audits LLC. All Rights Reserved.

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-FOMO-ADV-00	High	Resolved	`user_acc.total_amount` is not updated in `buy_ticket` when sidepot money is won, leading to a potential discrepancy in accumulated rewards.
OS-FOMO-ADV-01	High	Resolved	A missing referral code creation check allows unintended referrer assignments without fee payment validation.
OS-FOMO-ADV-02	Low	Resolved	`jackpot_amount` and `burned_amount` are rounded down resulting in their sum not being equal to `total_amount`.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-FOMO-ADV-00 [high] | Disparity In Rewards Update

Description

`buy_ticket` fails to update `user_acc.total_amount`, when the user wins the sidepot money, resulting in an inconsistency between `user_acc.total_amount` and `user_acc.balance_amount`, which is correctly updated with the sidepot reward. Throughout the program `user_acc.total_amount` and `user_acc.balance_amount` variables are increased together as `total_amount` represents the total accumulated rewards by the user (including historical rewards), while `balance_amount` tracks accumulated rewards which haven't been withdrawn yet by the user.

src/lib.rs RUST

```rust
pub fn buy_ticket(ctx: Context<BuyTicket>, team: String, quantity: u64) ->
    ↪  Result<()> {
    [...]
    if curr_time > game_acc.start_time + INITIAL_PHASE_DURATION {
        vault_acc.sidepot_amount += total_amount / 100;
        vault_acc.sidepot_probability += quantity;
        if random_num_acc.is_used {
            msg!("Random number not generated");
        } else {
            msg!("Random number used : {}", random_num_acc.random_num);
            if random_num_acc.random_num < vault_acc.sidepot_probability {
                let amount = vault_acc.sidepot_amount;
                user_acc.balance_amount += amount;
                user_acc.sidepot_amount += amount;
                user_acc.sidepot_wins += 1;
            [...]
            }
        }
    }
}
```

Thus, if these two values do not increase in tandem, `total_amount` may fall below `balance_amount`, resulting in a disparity between the presented total rewards and the withdrawable rewards. Any computations or conditions dependent on `user_acc.total_amount` will yield inaccurate outcomes, influencing the fairness and precision of reward distribution.

Remediation

Ensure that `user_acc.total_amount` is consistently updated alongside `user_acc.balance_amount` whenever rewards are accrued or modified, including when winning the sidepot.

© 2023 Otter Audits LLC. All Rights Reserved.

Patch

Resolved in ff0d967.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-FOMO-ADV-01 [high] | Missing Referrer Validation

Description

`buy_ticket` checks if a referrer account (`referrer_acc`) exists, and if so, it ensures that the authority of the referrer is not the same as the buyer's authority. Additionally, if the user has already utilized a referral code (`user_acc.is_referral_code_used` is `true`), it checks that the referrer's authority matches the stored referrer authority in the user's account.

However, the code does not explicitly check whether the referrer has created a referral code or paid the required referral creation fee. This may allow users to set any user as their referrer, even if that referrer has not paid the fee to become a referrer.

Remediation

Ensure the referrer has created a referral code before allowing users to set them as their referrer.

Patch

Resolved in 9af94da.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-FOMO-ADV-02 [low] | Rounding Error

Description

The issue arises from rounding errors in the calculation of `jackpot_amount` and `burned_amount` during the initial phase of the game in `buy_ticket`. `jackpot_amount` and `burned_amount` are rounded down during percentage calculations, potentially causing a discrepancy between the sum of `jackpot_amount` and `burned_amount` and the actual `total_amount`.

Remediation

Calculate `burned_amount` as `burned_amount = total_amount - jackpot_amount`. This ensures that any rounding errors are captured in the subtraction, maintaining consistency with the `total_amount`.

Patch

Resolved in ff0d967.

© 2023 Otter Audits LLC. All Rights Reserved.

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-FOMO-SUG-00	`settle_reward` lacks an explicit overflow check.
OS-FOMO-SUG-01	Suggestions regarding removal of `team_amount` calculations which are unnecessary during the initial phase of the game, and inclusion of `settle_rewards` calculations on chain.
OS-FOMO-SUG-02	The code contains certain error blocks which are unreachable and redundant.
OS-FOMO-SUG-03	Recommendations regarding inconsistencies in `get_ticket_cost` and `buy_ticket`.
OS-FOMO-SUG-04	In `buy_ticket`, during the initial phase, `total_amount` is added to `game_acc.team_wise_amount` for variable ticket allocations, introducing inconsistency in team-wise amounts.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-FOMO-SUG-00 | Overflow Check

Description

There is risk of accidentally releasing a debug version where overflow behavior might differ from the release version. Thus, using explicit overflow checks in `settle_reward` for the calculation of `user_amount_gain` will help mitigate this risk.

src/lib.rs RUST

```rust
pub fn settle_reward(ctx: Context<SettleReward>, players_amount: u64) ->
    ↪  Result<()> {
        let game_acc = &mut ctx.accounts.game_account;
        let user_acc = &mut ctx.accounts.user_account;
        [...]
        let user_amount_gain = players_amount - user_acc.players_amount;
        [...]
    }
```

Remediation

Ensure to add explicit overflow check in `settle_reward` for the calculation of `user_amount_gain` as shown below:

src/lib.rs RUST

```rust
let user_amount_gain = players_amount.checked_sub(user_acc.players_amount);
// handle overflow error
```

Patch

Resolved in ff0d967.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-FOMO-SUG-01 | Code Refactoring

Description

1. In `buy_ticket`, the `team_amount` calculations occur outside of the `else` block. They execute in both the initial phase of the game and subsequent phases. However, these calculations are not relevant during the initial phase of the game.

src/lib.rs RUST

```rust
pub fn buy_ticket(ctx: Context<BuyTicket>, team: String, quantity: u64) ->
    ↪  Result<()> {
    [...]
    // This block executes during both initial and subsequent phases
    game_acc.total_amount += total_amount;
    game_acc.total_tickets += quantity;
    game_acc.team_wise_amount[get_team_index(&team) as usize] += total_amount;

    let team_amount = total_amount * team_percentage / 100;
    game_acc.team_amount += team_amount;
    vault_acc.total_team_amount += team_amount;
    vault_acc.balance_team_amount += team_amount;

    vault_acc.total_amount += total_amount - burned_amount;
    vault_acc.balance_amount += total_amount - burned_amount;
    [...]
}
```

2. In `settle_reward`, it would be better to move the calculation of the user's share from the `players_amount` on-chain, as opposed to being calculated off-chain, in order to enhance decentralization.

Remediation

1. Move the calculations for `team_amount` to the `else` block above within `buy_ticket`.
2. Ensure the calculation of the user's share from the `players_amount` are done on chain.

Patch

1. Resolved in 5628e34.
2. Resolved in ff0d967.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-FOMO-SUG-02 | Unreachable Error Code Blocks

Description

1. The error code block for the `GameAlreadyInitialized` error in `initialize_game` is inaccessible as the `#[account(init)]` attribute on `game_account` initializes a new `GameAccount`.

src/lib.rs RUST

```rust
pub fn initialize_game(ctx: Context<InitializeGame>) -> Result<()> {
    let game_acc = &mut ctx.accounts.game_account;
    if game_acc.is_initialized {
        return err!(CustomError::GameAlreadyInitialized);
    }
    [...]
}
```

2. In `buy_ticket`, there is an `else if` block that checks if `user_acc.is_referral_code_used` is `true` and `referrer_account` is None. This condition may be redundant since if the user has utilized a referral code (`is_referral_code_used` is `true`), the previous `if let referrer_acc = &mut ctx.accounts.referrer_account` block would have already executed, rendering it impossible for `referrer_account` to be None in the `else if` block.

Remediation

1. Remove the `GameAlreadyInitialized` error block.
2. Remove the redundant `else if` block.

Patch

1. Resolved in 31dcc91.
2. Resolved in 2f4144b.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-FOMO-SUG-03 | Code Inconsistencies

Description

1. In the formula for calculating the cost in `get_ticket_cost`, there is an inconsistency in the base value utilized, which is 1.002, contrary to the base value of 1.0002 as specified in the documentation. Similarly, the referral creation fee in the documentation is mentioned as 0.1 SOL, while the code indicates 0.001 SOL.
2. `buy_ticket` computes the `burned_amount` at the commencement of the game. However, it neglects to incorporate this value into the `burned_amount` attributes of both `game_acc` and `vault_acc`. This discrepancy may result in an inconsistency, not accurately reflecting the burned amount in the game and vault records.

src/lib.rs RUST

```rust
pub fn buy_ticket(ctx: Context<BuyTicket>, team: String, quantity: u64) ->
    ↪ Result<()> {
    [...]
    if curr_time <= game_acc.start_time + INITIAL_PHASE_DURATION {
        total_amount = get_ticket_cost(1, 1) * quantity;
        game_acc.const_tickets += quantity;
        game_acc.jackpot_amount += total_amount * 90 / 100;
        if user_acc.is_referral_code_used {
            referrer_percentage = 10;
        } else {
            burned_amount = total_amount / 10;
        }
    }
    [...]
}
```

3. In its current implementation, `buy_ticket` lacks a check to ensure that all the percentages from the team info add up to 86.

Remediation

1. Ensure the documentation is consistent with the code.
2. Add the burned amount to both the game account (`game_acc`) and the vault account (`vault_acc`).
3. Implement a check to ensure all the percentages add up to 86.

© 2023 Otter Audits LLC. All Rights Reserved.

Patch

1. Resolved in c95329d.
2. Resolved in 5628e34.
3. Resolved in 5628e34.

© 2023 Otter Audits LLC. All Rights Reserved.

OS-FOMO-SUG-04 | Inconsistent Team Allocations

Description

In `buy_ticket`, `total_amount` is added to `game_acc.team_wise_amount` even during the initial game phase. The constant ticket allocation logic applies in the initial phase (`curr_time <= game_acc.start_time + INITIAL_PHASE_DURATION`). `total_amount` should contribute to `game_acc.team_wise_amount` only for the constant tickets allocated (`game_acc.const_tickets`), not for the variable ticket allocations (`quantity - game_acc.const_tickets`).

src/lib.rs RUST

```rust
pub fn buy_ticket(ctx: Context<BuyTicket>, team: String, quantity: u64) ->
    ↪  Result<()> {
    [...]

    game_acc.total_amount += total_amount;
    game_acc.total_tickets += quantity;
    game_acc.team_wise_amount[get_team_index(&team) as usize] += total_amount;

    [...]
}
```

The inconsistency arises as the program adds the total amount to `game_acc.team_wise_amount` without distinguishing between constant and variable tickets during the initial phase. This may result in inaccurate team-wise amounts during the initial phase, potentially affecting subsequent calculations and rewards.

Remediation

Modify the logic in the initial phase to add `total_amount` only for the constant ticket allocation and adjust the team-wise amounts accordingly to ensure consistency in the allocation process.

© 2023 Otter Audits LLC. All Rights Reserved.

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 section.

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.
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants. • Improved input validation.

© 2023 Otter Audits LLC. All Rights Reserved.

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 the other 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.

© 2023 Otter Audits LLC. All Rights Reserved.