FOMO Program Security Review

Table of Contents

1. Introduction

This audit focuses on the FOMO protocol, a solana-based program designed to create an engaging “last-man-standing” game that leverages the Fear of Missing Out (FOMO) phenomenon. FOMO addresses the challenge of creating a fair and exciting decentralized game by implementing a dynamic pricing mechanism for NFT minting and a time-based round system.

The audit was conducted between Oct 1st, 2024 and Oct 16th 2024. During this period, the FOMO program was thoroughly analyzed with particular attention paid to the round management system, the Key minting and burning processes, and the reward distribution mechanisms. Several recommendations were made and communicated to the development team for the current and future deployments, particularly regarding the management of liquidity pools and the potential for market manipulation in low-liquidity scenarios.

This report outlines the audit process, describes the methodology used, and certifies the program as secure to the best of our knowledge, while also highlighting areas for potential optimization and enhancement. The audit pays special attention to the game theory aspects of FOMO, ensuring that the incentive structures align with the project’s goals of creating a fair, engaging, and potentially profitable experience for all participants and the associated SEND token.

1.1 Overview

FOMO (Fear of Missing Out) is a Solana-based program that creates an engaging NFT minting and trading experience. Here's an overview of its key features:

Round-based Structure - The program operates in rounds, each with a defined start and undefined end time.

Dynamic NFT Minting - Users can mint NFTs during a round. The price of minting increases with each subsequent mint, encouraging early participation and eventual conclusion of each round.

Token Distribution - When users mint NFTs, the tokens they spend are distributed across three pools:

- Mint Fee Pool: Platform fees and team compensation
- NFT Pool: Shared among NFT holders to incentivize burning
- Main Pool: The grand prize for the round's winner

NFT Burning Option - Participants can choose to "burn" their NFT before the round ends. This allows them to claim a share of the NFT Pool.

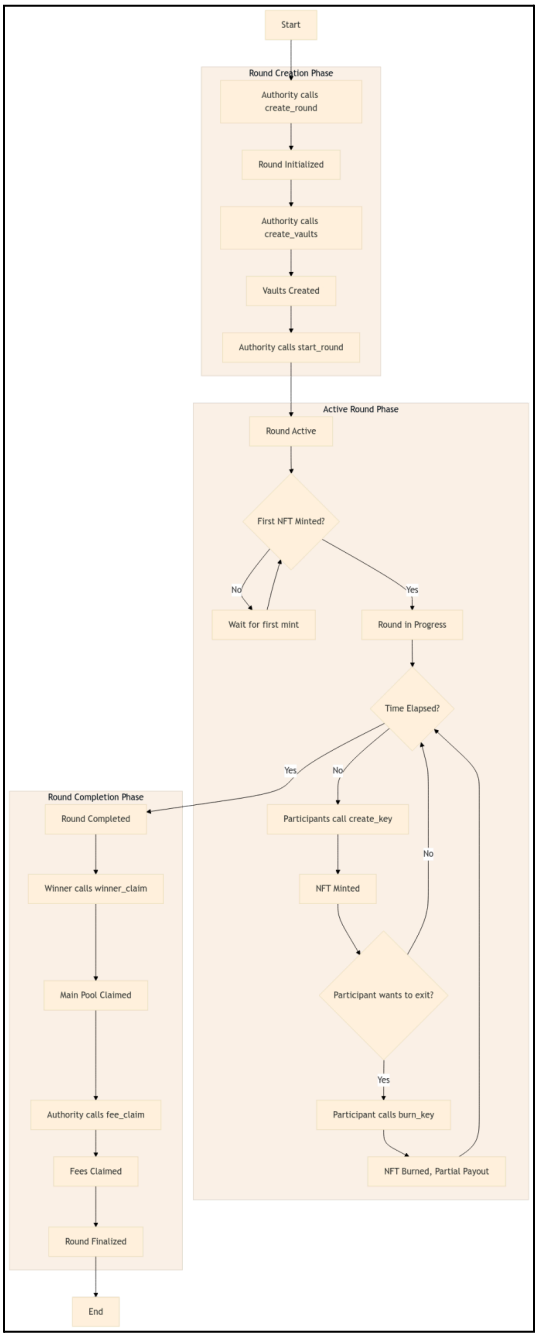
Winning Mechanism - The holder of the last minted NFT when the round ends wins the Main Pool.

Collection Management - Each round creates a unique NFT collection, managed using the latest and simpler mpl-core program for compatibility and rent efficiencies with Solana NFT ecosystems.

The FOMO program gamifies the NFT minting process, creating a dynamic where participants must balance the risk of minting at higher prices against the potential reward of winning the main pool or exiting early with a share of the NFT pool.

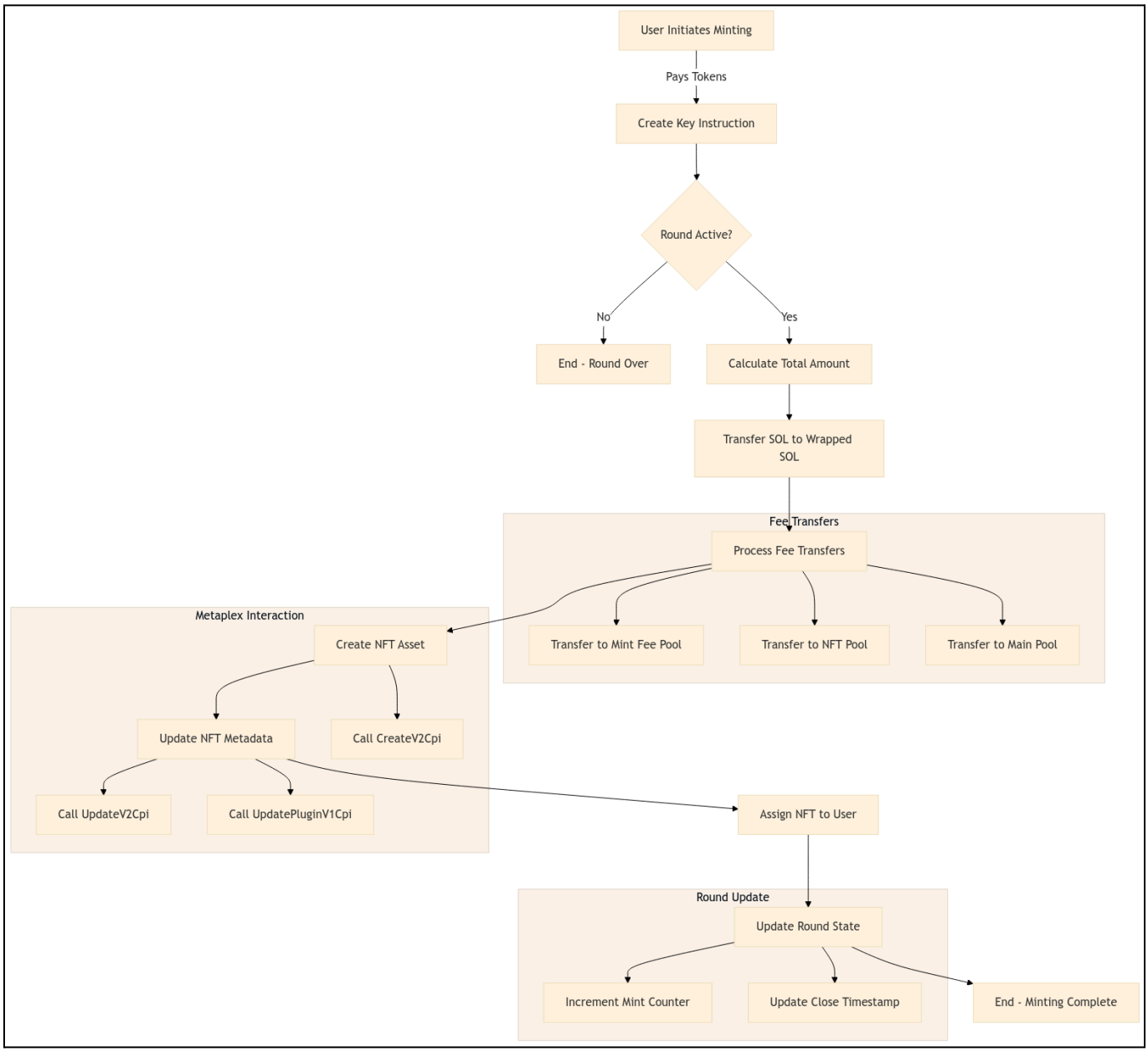
1.2 FOMO Round Lifecycle

The FOMO round lifecycle begins when an authority creates a new round. Once initialized, the round becomes active with the minting of the first NFT. As time progresses, the round remains in an active state, allowing participants to mint NFTs. The round continues until a predetermined end condition is met, at which point it transitions to a completed state. The cycle concludes with the winner claiming the main pool, marking the round as finalized and ready for a new cycle to begin.



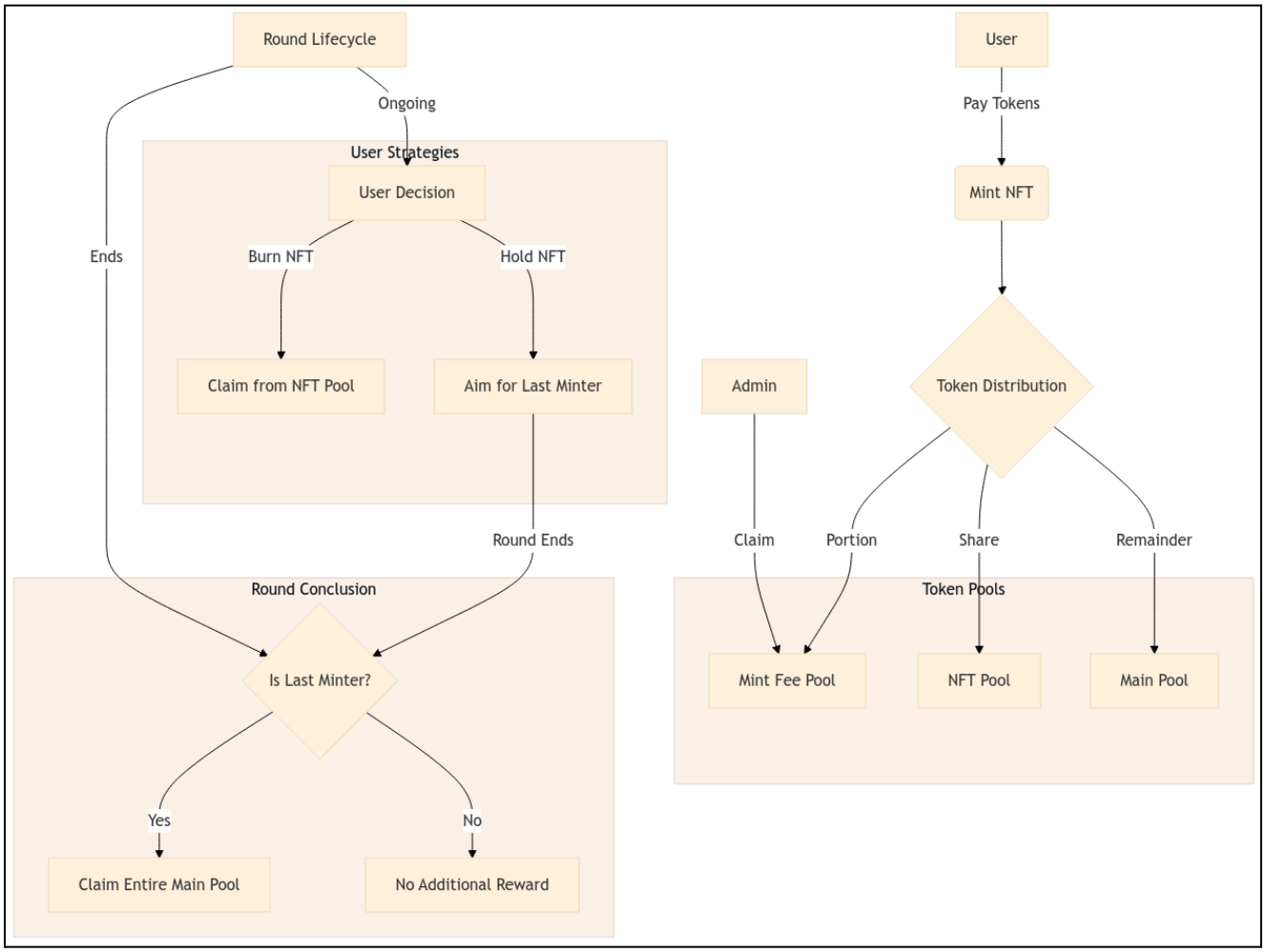
1.3 NFT Minting Process

The NFT minting process in the FOMO program starts when a user decides to participate by paying tokens. This action triggers the Create Key instruction, which manages the distribution of the paid tokens across three distinct pools: the Mint Fee Pool, the NFT Pool, and the Main Pool. Simultaneously, the program interacts with the Metaplex protocol to create a unique NFT. Once minted, this newly created NFT is assigned to the user, completing their participation in the current round and potentially positioning them as the winner if they're the last to mint before the round ends.



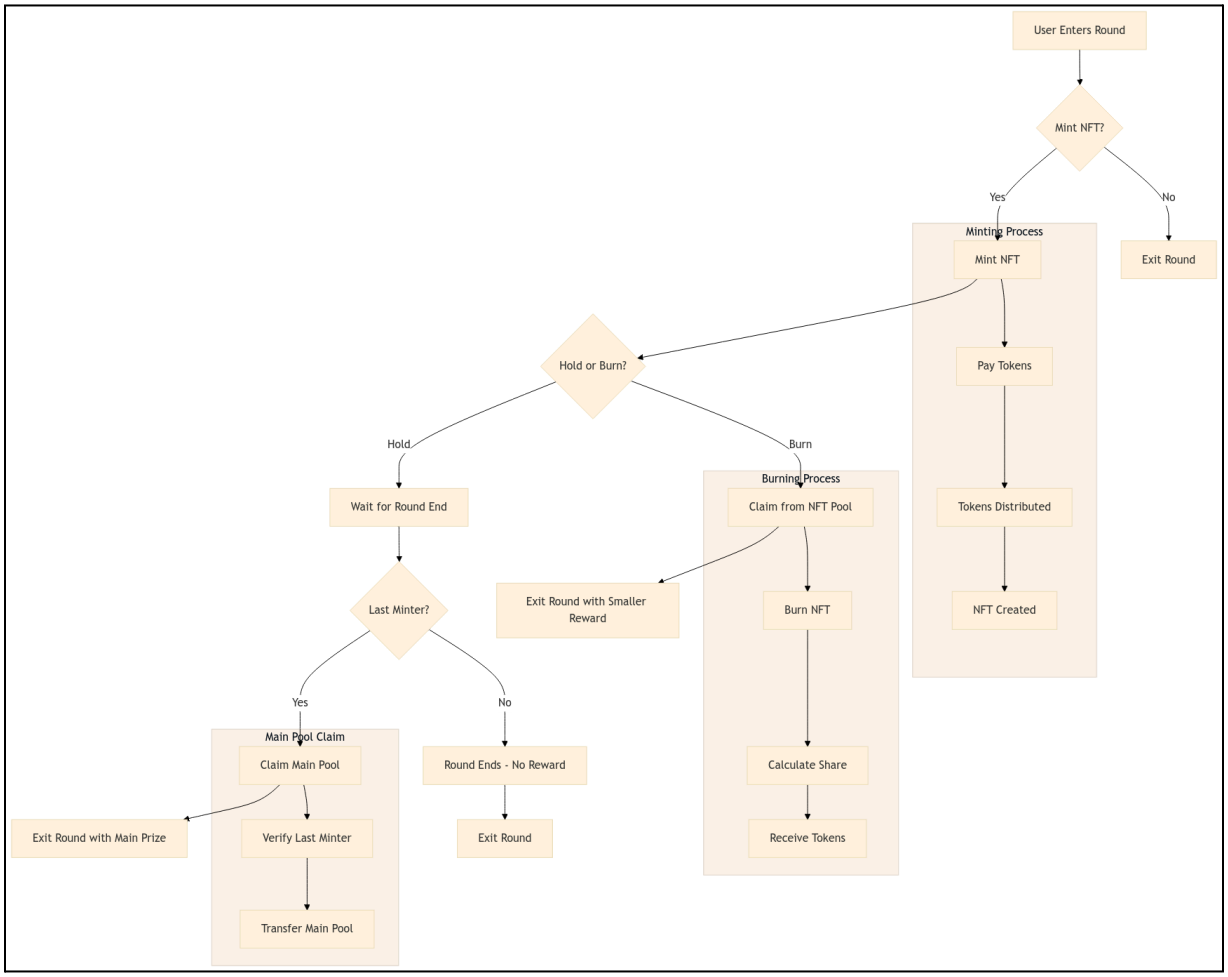
1.5 Token Flow in FOMO

The token flow within the FOMO program is designed to create an engaging economic model. Users enter the system by paying tokens to mint NFTs. These tokens are then strategically distributed by the program: a portion is allocated to the Mint Fee Pool, another share goes to the NFT Pool, and the remainder is added to the Main Pool. This distribution creates two potential paths for users to reclaim tokens: they can either burn their NFT to claim a share from the NFT Pool at any time during the round, or aim to be the final minter and claim the entire Main Pool when the round concludes. This dual-path system encourages both strategic gameplay and sustained participation throughout the round.



1.6 User Actions and Outcomes

The FOMO program presents users with a series of strategic choices and potential outcomes. Upon entering a round, users can choose to mint and hold an NFT, positioning themselves for a chance at the main prize, or they can opt to burn their NFT for a guaranteed, but potentially smaller, return from the NFT Pool. Those who hold their NFTs play a waiting game as the round progresses. If a user finds themselves as the last minter when the round concludes, they're rewarded with the ability to claim the entire Main Pool. However, if they're not the last minter, their participation in the round ends without additional rewards. This structure creates a dynamic environment where users must balance the risk of holding against the certainty of an early exit, driving the core "fear of missing out" mechanic that gives the program its name.



2. Scope and Objectives

The primary objectives of this audit are defined as:

- Minimizing the possible presence of any critical vulnerabilities in the program. This would include detailed examination of the code and edge case scrutinization to find as many vulnerabilities.

- 2-way communication during the audit process. This included for Mad Shield to reach a perfect understanding of the design of the system and the goals of the team.

- Provide clear and thorough explanations of all vulnerabilities discovered during the process with potential suggestions and recommendations for fixes and code improvements.

- Clear attention to the documentation of the vulnerabilities with an eventual publication of a comprehensive audit report to the public audience for all stakeholders to understand the security status of the programs.

The FOMO program was delivered to Mad Shield with the following targets:

Start of primary review period	01/10/2024
End of primary review period	16/10/2024
Repository	https://github.com/thesendcoin/fomo-contract2
Version	1689190c3876b8c9901ccfbcbed60ae2c7018ce4

3. Methodology

The FOMO program, a complex Program system on the Solana blockchain, requires a comprehensive security review to ensure its integrity, fairness, and resistance to potential exploits. Our approach to reviewing and testing the FOMO program focuses on several key areas:

Access Control Analysis - We thoroughly examine the program's access control mechanisms, particularly in critical functions like `create_round`, `start_round`, and `winner_claim`. This ensures that only authorized entities can perform privileged operations.

Token Flow Verification - Given the program's multi-pool structure (Mint Fee, NFT, and Main pools), we meticulously trace token flows to verify correct distribution and prevent any unauthorized token leakage.

Randomness and Fairness Evaluation - While FOMO doesn't rely on explicit randomness, we assess the fairness of the last-minter-wins mechanism to ensure it cannot be gamed or manipulated.

Overflow and Underflow Checks - We conduct comprehensive checks for potential arithmetic overflow or underflow, especially in functions handling token amounts and round counters.

Logic Consistency Verification - We verify that the program's logic, especially around NFT minting, burning, and winner determination, is consistent and free from loopholes that could be exploited.

Metadata and NFT Integrity Checks - We verify the integrity of NFT creation and metadata assignment processes to prevent forgery or unauthorized modifications.

Our security review process involves both manual code review by experienced blockchain security experts and automated analysis using specialized tools designed for Solana smart contract auditing. We also conduct simulations of various attack scenarios to test the program's resilience in real-world conditions.

This comprehensive approach aims to identify and mitigate potential security risks, ensuring the FOMO program operates securely and fairly for all participants.

4. Findings & Recommendations

Our severity classification system adheres to the criteria outlined here.

Severity Level	Exploitability	Potential Impact	Examples
Critical	Low to moderate difficulty, 3rd-party attacker	Irreparable financial harm	Direct theft of funds, permanent freezing of tokens/NFTs
High	High difficulty, external attacker or specific user interactions	Recoverable financial harm	Temporary freezing of assets
Medium	Unexpected behavior, potential for misuse	Limited to no financial harm, non-critical disruption	Escalation of non-sensitive privilege, program malfunctions, inefficient execution
Low	Implementation variance, uncommon scenarios	Zero financial implications, minor inconvenience	Program crashes in rare situations, parameter adjustments by authorized entities
Informational	N/A	Recommendations for improvement	Design enhancements, best practices, usability suggestions

In the following, we enumerate some of the findings and issues we discovered and explain their implications and corresponding resolutions.

Finding	Description	Severity Level
FOMO_01	Incorrect Royalty Configuration Leads to Misallocation of Funds	*Critical*
FOMO_02	Unverifiable Swap Function from Meteora Introduces Unknown Security Risks	*Critical*

1. FOMO_01 - Incorrect Royalty Configuration Leads to Misallocation of Funds

A key aspect of the FOMO protocol is the creation and trading of NFTs where the protocol utilizes the Metaplex's MPL-Core program. We noticed the collection royalty setup is currently not configured correctly. This oversight represents a potential missed opportunity for additional income, as users might choose to trade the NFTs on secondary markets rather than burning them within the protocol.

Finding Description

The current implementation incorrectly configures the royalty receiver in the NFT collection. Instead of directing royalties to the team or admin account, they are set to be received by the round account. This misconfiguration is occurring in the `utils.rs` file, specifically in `get_master()`:

```
pub fn get_master(round_account: Pubkey) -> Config {
    Config {
        name: String::from("Master Key"),
        uri: String::from("https://purple-quickest-catshark-409.mypinata.cloud/ipfs/QmXEFnXdMLeSCtzEk8gaiEcDq18DncZ8aRduSNmDgUa2kr"),
        plugins: Vec::from([
            PluginAuthorityPair {
                plugin: Plugin::FreezeDelegate(FreezeDelegate { frozen: true }),
                authority: Some(PluginAuthority::UpdateAuthority),
            },
            PluginAuthorityPair {
                plugin: Plugin::VerifiedCreators(VerifiedCreators {
                    signatures: Vec::from([
                        VerifiedCreatorsSignature {
                            verified: true,
                            address: round_account,
                        },
                    ]),
                }),
                authority: Some(PluginAuthority::UpdateAuthority),
            },
        ]),
    }
}
```

The `get_master()` function (or others function in utils.rs) is responsible for setting up the configuration for the master NFT of each round. However, it's not correctly specifying the royalty receiver.

This issue stems from incorrectly setting the royalty receiver to the round account (a program-derived address) instead of the team or admin account. As there's no instruction to redeem funds from program-owned accounts, any royalties sent to this address become inaccessible, resulting in lost revenue for the project stakeholders.

Impact Explanation

Critical. This misconfiguration directly affects the financial aspects of the protocol. Royalties, which are meant to provide ongoing revenue to the project team or administrators, are instead being sent to an account that may not be easily accessible. This could result in a significant loss of intended revenue for the project stakeholders.

Likelihood Explanation

High. This issue will affect every round created and every NFT minted within the system. As long as the protocol is in use and NFTs are being traded, royalties will consistently be misdirected.

Recommendation

To fix this issue, the royalty receiver should be explicitly set to the team or admin address when creating the collection. This can be done by modifying functions in the `utils.rs` files.

Mi

2. FOMO_02 - Unverifiable Swap Function Introduces Security Risks in Key Creation Process

The FOMO protocol implements a key creation system where users can mint NFTs (keys) by paying with various tokens. The protocol integrates with Meteora, a closed-source aggregator developed by Jupiter, to handle token swaps during the key creation process.

Finding Description

The `create_key()` function in the protocol allows users to pay for minting keys using any token, which is then swapped to the desired token (likely SOL or SEND) using Meteora's swap function. This implementation introduces significant security and verifiability issues:

The `CreateKeyContext::create_key()` , which call `process_fee_transfers()` function contains the logic for interacting with Meteora's swap function, which cannot be audited or verified due to its closed-source nature.

```
        // Perform token transfers for various fees (Burn, Team, Pool, Treasure).
        Self::process_fee_transfers(&ctx, total_amount_for_index)?;

        // Process asset creation and plugin update for the new key.
        Self::process_asset_creation(&ctx)?;

        Ok(())
    } fn create_key

    /// Helper function to process fee-related token transfers.
    fn process_fee_transfers(ctx: &Context<CreateKeyContext>, total_amount: u64) -> Result<()> {
        let fee_bps: [(u64, &Box<Account<'_, TokenAccount>>); … = [
            (MAIN_FEE_BASIS_POINTS, &ctx.accounts.main_pool_vault),
            (NFT_FEE_BASIS_POINTS, &ctx.accounts.nft_pool_vault),
            (MINT_FEE_BASIS_POINTS, &ctx.accounts.mint_fee_vault),
        ];

        for (bps: &u64, vault: &&Box<Account<'_, TokenAccount>>) in fee_bps.iter() {
            let amount: u64 = total_amount.checked_mul(*bps).unwrap().checked_div(10_000).unwrap();

            let accounts: Swap<'_> = dynamic_amm::cpi::accounts::Swap {
                pool: ctx.accounts.pool.to_account_info(),
                user_source_token: ctx.accounts.user_source_token.to_account_info(),
                user_destination_token: vault.to_account_info(),
                a_vault: ctx.accounts.a_vault.to_account_info(),
                b_vault: ctx.accounts.b_vault.to_account_info(),
                a_token_vault: ctx.accounts.a_token_vault.to_account_info(),
                b_token_vault: ctx.accounts.b_token_vault.to_account_info(),
                a_vault_lp_mint: ctx.accounts.a_vault_lp_mint.to_account_info(),
                b_vault_lp_mint: ctx.accounts.b_vault_lp_mint.to_account_info(),
                a_vault_lp: ctx.accounts.a_vault_lp.to_account_info(),
                b_vault_lp: ctx.accounts.b_vault_lp.to_account_info(),
                admin_token_fee: ctx.accounts.admin_token_fee.to_account_info(),
                user: ctx.accounts.authority.to_account_info(),
                vault_program: ctx.accounts.vault_program.to_account_info(),
                token_program: ctx.accounts.token_program.to_account_info(),
            };

            let cpi_context: CpiContext<'_, '_, '_, '_, …> = CpiContext::new(
                program: ctx.accounts.dynamic_amm_program.to_account_info(),
                accounts
            );

            let _ = dynamic_amm::cpi::swap(ctx: cpi_context, in_amount: amount, minimum_out_amount: 0);
        }

        Ok(())
    } fn process_fee_transfers
```

This design choice compromises the transparency and security of the entire key creation process. The use of a closed-source component in a critical financial operation goes against the principles of open and verifiable smart contract development.

Impact Explanation

Critical. The inability to verify the swap function introduces potential security vulnerabilities that could affect the entire key creation process. This could lead to various attack vectors, including but not limited to:

- Incorrect token valuations during swaps
- Potential for manipulation of swap rates
- Unexpected behavior or bugs that cannot be identified or fixed due to lack of source code access

Likelihood Explanation

High. This issue affects every key creation transaction that involves a token swap, which is likely to be a significant portion of all key creations. The problem is inherent in the current design and will persist as long as the closed-source Meteora integration remains in place.

Recommendation

To address this critical issue, we recommend one of the following approaches:

1. Remove Meteora integration entirely:
Modify the `create_key()` function to only accept WSOL (Wrapped SOL) as payment. This simplifies the process and removes the need for token swaps.

2. Implement an open-source swap solution:
Replace Meteora with an open-source AMM like Raydium. This allows for full code verification while maintaining the flexibility of multiple payment tokens. However, it's worth noting that `LP_MINT` tokens are already burnt, which may complicate this solution.

3. If Meteora integration must be maintained:
Clearly document the risks associated with using a closed-source component in the protocol. Implement additional monitoring and safety checks around the Meteora integration to mitigate potential risks as much as possible.

It's important to note that the development team seems inclined to maintain with the current Meteora implementation due to the fact that the LP tokens of the currently deployed liquidity is burnt as such locked in the closed-source Meteora protocol. This results in the status quo to remain as a persistent unverifiable issue and open security risk.

Additional Notes

The root cause of this issue lies with Jupiter's decision to keep Meteora closed-source. This choice by a major player in the Solana ecosystem creates challenges for projects like FOMO that rely on their services.

We strongly recommend advocating for Jupiter to open-source their code, as this would significantly enhance the security and transparency of not just FOMO, but many other projects in the ecosystem.

Regardless of the chosen solution, it's crucial to prioritize transparency and verifiability in the protocol's design. The current reliance on closed-source components introduces unnecessary risks and should be addressed to ensure the long-term security and reliability of the FOMO protocol. The final implementation decision rests with the FOMO team, but we strongly advise considering the security implications of each option.

5. Conclusion

The FOMO program demonstrates an innovative approach to NFT-based gaming with its dynamic pricing mechanism and round-based structure. However, the two critical vulnerabilities identified—the incorrect royalty configuration and the unverifiable swap function—need to be addressed to ensure the program can operate securely and efficiently within the Solana ecosystem. Our recommendations aim to assist the development team in enhancing the program's security posture and operational integrity.

The protocol's design enables engaging gameplay dynamics and novel tokenomics through its multi-pool structure and time-based rounds. The audit represents a collaborative effort between the FOMO team and Mad Shield to ensure the security and desired behavior of the program as it continues to evolve and potentially integrate with other DeFi protocols. We encourage ongoing vigilance and regular security assessments to maintain the robustness of this promising project in the face of the ever-changing blockchain landscape.

References

[1] FOMO details : link