FUN...FOMO...GOBBLE...GO

– Mad Shield

David P – *david@fomo3d.app david@madshield.xyz*
StaccOverflow – *stacc@fomo3d.app*
BlueWolf – *wolf@madshield.xyz*
JecikPo – *je@madshield.xyz*
OrcaKoan – *orcakoan@gmail.com*

Table of Contents

1. Introduction

This audit focuses on the F8M3 protocol, a Solana-based decentralized exchange (DEX) designed to revolutionize liquidity provisioning and trading through an innovative automated market maker (AMM) model. Gobbler addresses the challenges of capital efficiency, impermanent loss, and fee structures in decentralized finance (DeFi) by implementing a unique constant product curve with virtual liquidity and a flat-rate fee system.

The audit was conducted between September 20th, 2024 and October 31st, 2024. During this period, the Gobbler program was thoroughly analyzed with particular attention paid to the AMM structure, the deposit and withdrawal processes, and the swap mechanism. Several recommendations were made and communicated to the development team for future improvements.

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 examines the incentive structures created by the flat-rate fee system and the early liquidity provider rewards, ensuring they align with the project's goals of creating a fair, efficient, and potentially more profitable DEX experience for all participants.

1.1 Overview

Gobbler is a Decentralized Exchange (DEX) designed to differentiate itself through a constant fee structure and a unique liquidity provisioning mechanism aimed at incentivizing early liquidity deposits and late withdrawals. This DEX is built with notable modifications that make it stand out within the Solana ecosystem. The core innovation lies in its liquidity pools (LPs) and their integration with a bonding curve model. Gobbler introduces two types of LPs:

1. **Traditional LP Pools:** These operate similarly to conventiotnal DEXs, where liquidity providers earn fees based on swaps and can withdraw their initial deposits at any time.
2. **Bonding Curve LPs:** In this model, LP tokens are priced on a bonding curve, meaning early depositors are rewarded with higher potential returns, and those who withdraw later face increased incentives. This structure mimics a "game-theory" approach, attracting liquidity providers to deposit early and delay withdrawals for maximum rewards.

Additionally, the Gobbler DEX offers fixed-rate fees, which contrasts with the percentage-based fees seen on most DEXs. This makes Gobbler an attractive venue for liquidity providers as well as traders, promising a more cost-effective trading environment. The multi-tiered fee system, encompassing trading, protocol, and fund fees, creates a predictable environment for traders. This model has attracted attention from key Solana stakeholders like the Jupiter Aggregator, which now routes larger trades through Gobbler due to lower fees.

2. CFMM Implementation

At the foundation of Gobbler's mechanics is a **constant function market maker (CFMM)** model specifically tailored for the Solana Virtual Machine (SVM). For a given trading pair, say assets X and Y, the Gobbler AMM enforces a consistent equilibrium using the relation:

$$x * y = k^2$$

Where x and y represent the quantities of assets X and Y held in reserve pools managed by the protocol. Liquidity providers (LPs) fund these reserves to facilitate trades across the Gobbler network. Through this relationship, the AMM enables users to exchange an amount Δx of asset X for a corresponding amount Δy of asset Y, ensuring that:

$$(x + \Delta x) * (y - \Delta y) = x * y = k^2$$

This approach preserves the constant product formula and maintains stable liquidity conditions across the trading pair.

2.1 LP Bonding Curve

At the core of this design is a novel approach that builds upon the familiar $xy = k^2$ equation by introducing an additional curve :

$$k^2 * k'^2 = virtual\_\alpha$$

In this formula, k^2 represents the traditional constant product, k'^2 is a dynamic constant that evolves based on the timing of liquidity provision, and $virtual\_\alpha$ is a parameter governing the relationship between k and k'. This dual-curve system allows for more nuanced control over liquidity incentives.

The introduction of k'^2 creates a mechanism where the effective liquidity $L(t)$ at time t is influenced by when it was provided. Early liquidity providers receive more tokens for the same amount of liquidity, represented as $L(t\_early) > L(t\_late)$ for equal token inputs. This structure not only incentivizes early liquidity provisioning but also encourages providers to maintain their positions for longer periods. The dynamic nature of k'^2 extends to fee distribution as well, where fee allocation $f(t)$ is proportional to $L(t)$, allowing early liquidity providers to earn more fees over time.

In the classic AMM design where the *Δx* and *Δy* are calculated from the requested amount of LP tokens (*ΔL*) the following equations are used:

$$\Delta x = \Delta L * Rx / L$$
$$\Delta y = \Delta L * Ry / L$$

Where *Rx* and *Ry* represent the total token reserves and the *L* represents the total supply of the LP token. As in Uniswap v2 [1] the LP token supply is equivalent to the reserve product we can transform the bonding curve into the following equation:

$$L * L' = virtual\_\alpha$$

Where *L'* is the virtual reserves, hence:

$$L' = virtual\_\alpha / L$$

And in order to satisfy the *L(t_early) > L(t_late)* requirement the following formulas shall be used by Gobbler to calculate the amounts of reserve deltas:

$$\Delta x = \Delta L * Rx / L'$$
$$\Delta y = \Delta L * Ry / L'$$

Hence:

$$\Delta x = \Delta L * Rx / virtual\_\alpha / L$$
$$\Delta y = \Delta L * Ry / virtual\_\alpha / L$$

By incentivizing early and sustained liquidity provision, Gobbler aims to achieve higher capital efficiency compared to traditional AMMs. The dynamic k'^2 factor allows the protocol to adjust incentives based on the current state of the liquidity pool.
At the core of this design is a novel approach that builds upon traditional AMM mechanics through two key equations:

$$X * Y = K^2$$
$$K^2 * K'^2 = \alpha$$

Where X and Y are the token supplies, K represents the real supply of LP tokens, K' represents the virtual liquidity (with an applied multiplier), and α is a constant parameter governing their relationship.
When users provide liquidity, they specify what percentage of the pool they want to own, following the traditional AMM principle:

$$\delta X/X = \delta Y/Y = \delta K/K$$

However, Gobbler introduces an additional step: after calculating δK based on the desired pool share, the protocol applies a buy function to determine δK' (the actual amount of LP tokens to mint) while maintaining the α constant. This creates a mechanism where early liquidity providers receive more LP tokens (K') for the same amount of real liquidity (K).

For withdrawals, users specify K' (the virtual tokens they own), and the protocol:

1. Applies the sell function to calculate δK (the real liquidity share)
2. Uses δK to determine the proportional amounts of X and Y to return (respecting X * Y = K^2)
3. Burns the requested δK' amount of LP tokens

This structure not only incentivizes early liquidity provisioning but also encourages providers to maintain their positions for longer periods. The dynamic nature of K' extends to fee distribution as well, where fee allocation is proportional to the virtual liquidity held, allowing early liquidity providers to earn more fees over time.
By incentivizing early and sustained liquidity provision through this dual constant product mechanism, Gobbler aims to achieve higher capital efficiency compared to traditional AMMs. The relationship between K and K' allows the protocol to adjust incentives based on the current state of the liquidity pool.

2.2 Fee Structure

Gobbler introduces an innovative flat-rate fee structure, deviating from conventional percentage-based models in decentralized exchanges. This approach inherently favors larger trades, reducing proportional costs for high-volume transactions and potentially mitigating MEV impact.
Gobbler's fee structure significantly influences pool balance dynamics and liquidity provider behavior. It aims to create a more equitable trading environment, aligning with the interests of both large-scale traders and long-term liquidity providers.

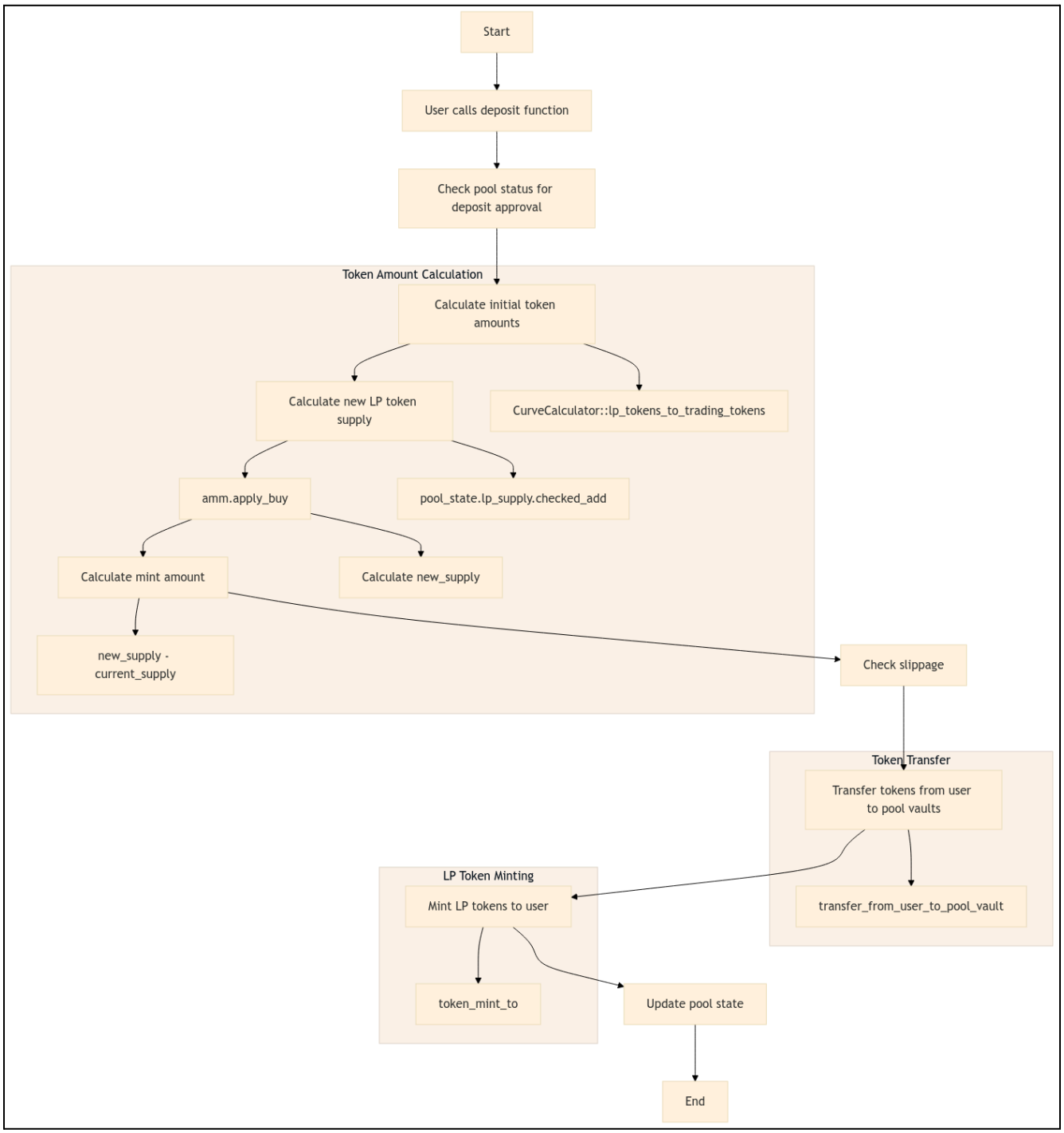
By offering an attractive environment for large trades and long-term liquidity provision, Gobbler positions itself to potentially reshape DeFi liquidity dynamics.

2.3 LP Token Metadata

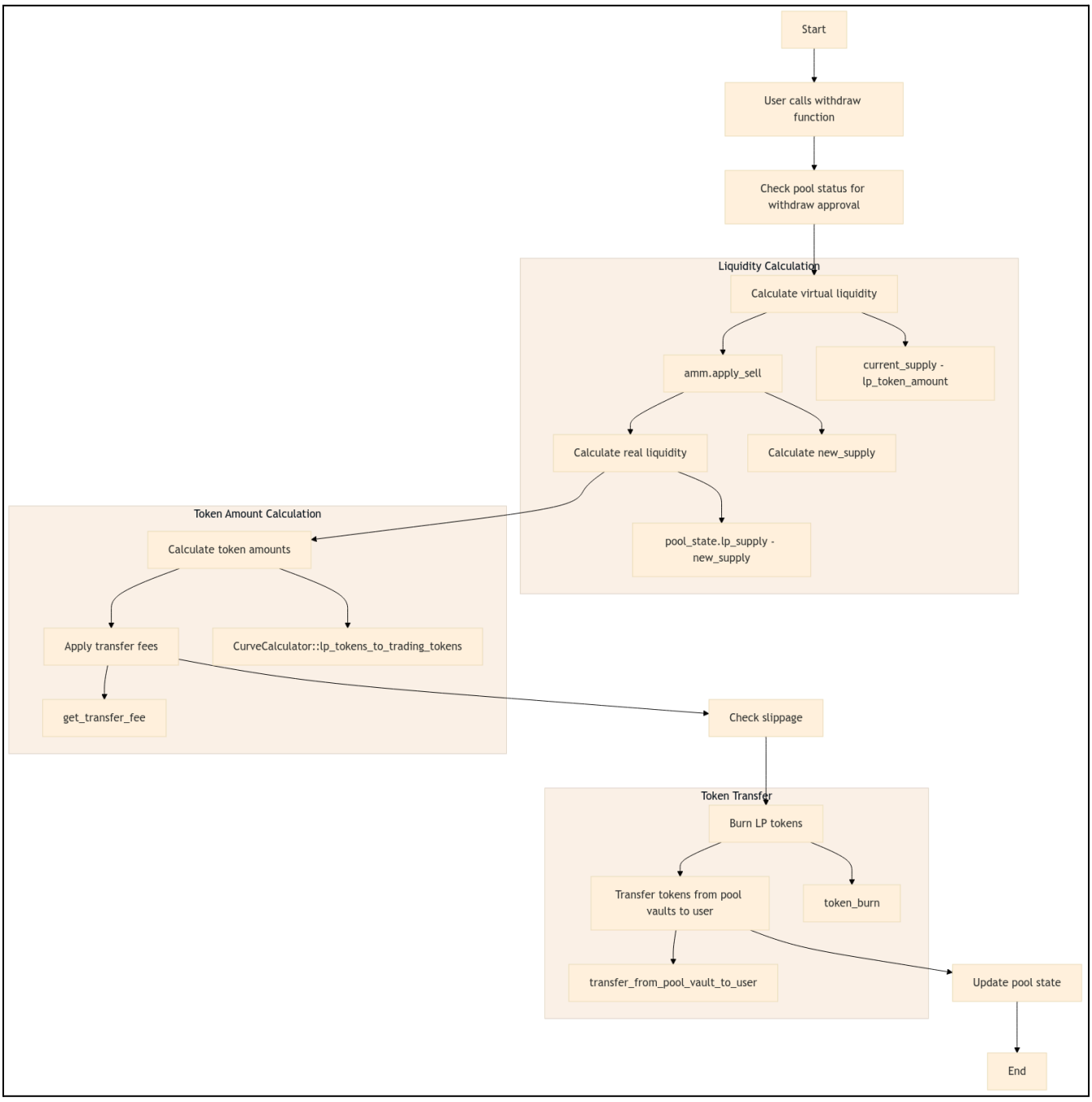
Gobbler introduces a sophisticated approach to Liquidity Provider (LP) token metadata, enhancing the functionality and flexibility of these tokens within the DeFi ecosystem. This feature allows for the customization of metadata associated with LP tokens, extending beyond basic identifiers to include rich, position-specific information. This customizable metadata structure potentially enables more nuanced representation of liquidity positions, facilitating the development of advanced financial products and strategies built upon these LP tokens. As the ecosystem evolves, striking a balance between customization and adherence to emerging standards will be crucial for maximizing the utility and integration potential of Gobbler's LP tokens across various DeFi protocols and applications.

2.4 Program Charts

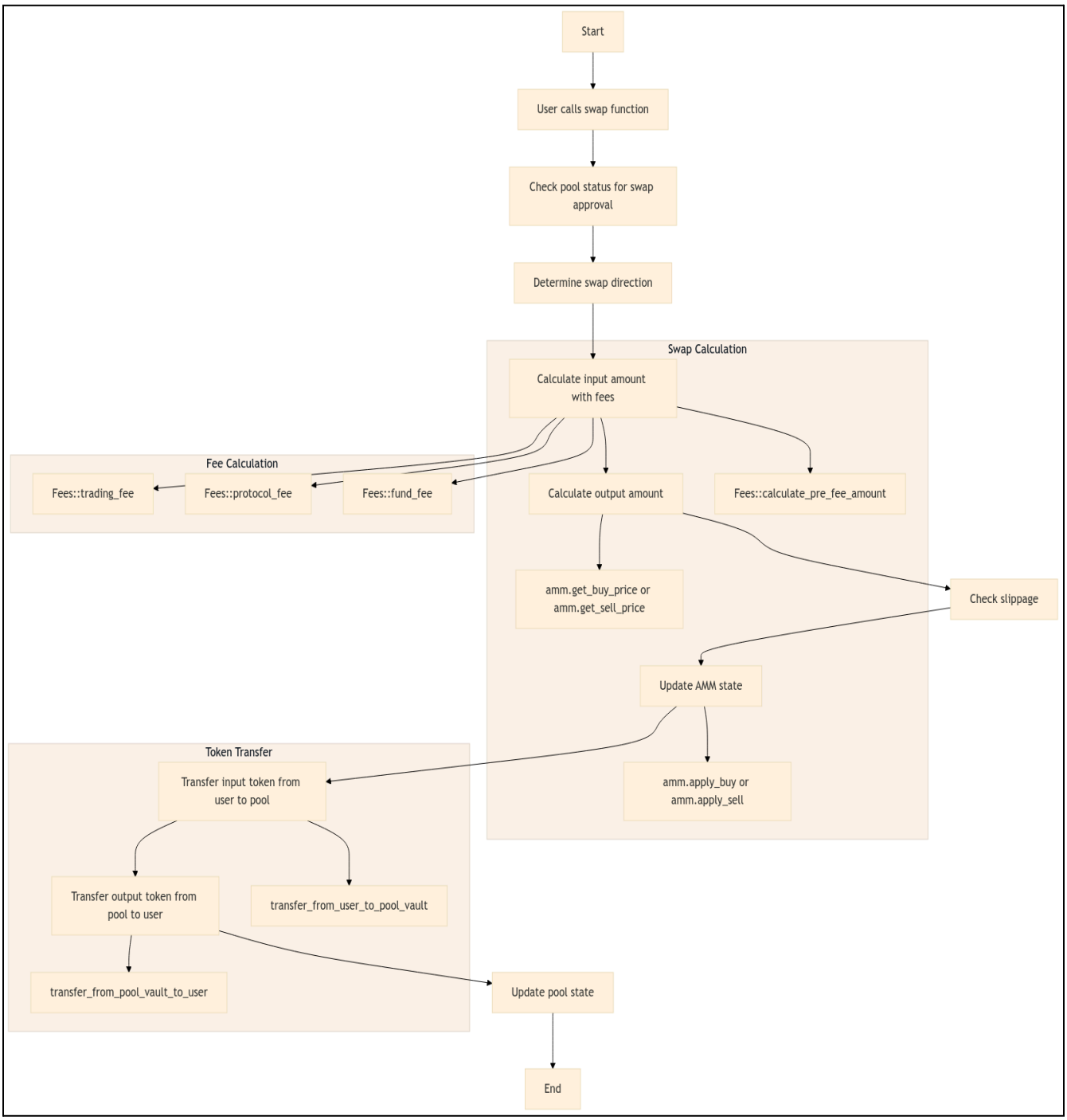
1. Deposit Process



2. Withdraw Process



3. Swap Process



3. Scope and Objectives

The primary objectives of the audit are defined as:

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

- 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.

Gobbler has delivered the program to Mad Shield at the following Github repositories.

Repository URL	https://github.com/staccDOTsol/raydium-cp-swap/tree/master
Commit (start of audit)	a1b39110a246cfb22108e4ade9186f172abb41da
Commit (end of audit)	a1b39110a246cfb22108e4ade9186f172abb41da

Tab1. Audit Marks

The project is scheduled for migration to a public open-source repository, which will enhance transparency for stakeholders. Upon completion, the repository will be available at https://github.com/FOMO-GOBBLE/mono-fomo.

4. Methodology

Given that the program is a fork of Raydium CP-Swap, we employed a comprehensive and systematic methodology to test the program's constraints and behavior. Our primary approach involved fetching historical transaction data from the CP-Swap program and leveraging it to simulate real-time scenarios.

This methodology allowed us to observe how the program would operate in real-world conditions, especially within high-volume environments like meme markets, which have shown significant throughput in constant product swaps. This data-driven approach was essential for evaluating the program's stability and responsiveness under stress.

Steps involved:

1. **Historical Data Analysis**: We gathered historical transaction logs from the Raydium CP-Swap contract. This included a diverse range of swap behaviors across different market conditions. 1271870 transactions including swaps, LP changes (deposit/withdraw)
2. **Client Update**: Using the collected transaction data, we updated the client to simulate on-chain behavior. This enabled us to create precise conditions for testing, such as sudden liquidity shifts, high-frequency transactions, and the impact of volatile meme markets.
3. **Simulation of Real-World Conditions**: The updated client was then used to simulate real-world operations, especially throughput in high-volume markets. This helped identify potential bottlenecks, inefficiencies, or vulnerabilities that may emerge when the program is deployed on-chain.
4. **Observations and Adjustments**: Throughout the simulations, we monitored performance metrics and stability indicators to fine-tune the program and adjust its constraints, ensuring optimal functionality.

This approach allowed us to evaluate how the program behaves in a live environment and ensured that it can efficiently handle real-time constraints.

5. 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 varia(nce, 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
GOBBLER_01 [RESOLVED]	Incorrect Bonding Curve Calculation Causing Pool Liquidity Shortfalls and Trade Failures	*Critical*
GOBBLER_02 [RESOLVED]	Misappropriation Of Fees Between The Involved Parties	*Critical*
GOBBLER_GR_01 [ACKNOWLEDGED]	Increase Flat Rate Swap Fees For Higher Protocol Revenue	*Informational*

1. GOBBLER_01 - Incorrect Bonding Curve Calculation Causing Pool Liquidity Shortfalls and Trade Failures [Critical]

Description

The bonding curve in the Raydium protocol is currently miscalculating the number of tokens to be deposited or withdrawn, returning fewer tokens than expected. Based on our analysis, two primary issues arise from this miscalculation:

1. **Reduced Pool Liquidity**: Large trades face significant slippage, leading to less efficient trades and reduced protocol functionality due to inadequate liquidity.
2. **Frequent Trade Failures for Smaller Transactions**: The protocol experiences insufficient liquidity for small trades, as deposits of equivalent liquidity amounts are divided by a `cost_ratio` to reward early liquidity providers (LPs) as per the bonding curve. Consequently, smaller trades fail, impacting the trading experience and liquidity provisioning.

Recommended Fix

To address this, the bonding curve calculation should be modified so that the curve's effect is applied directly to the quantity of LP tokens rather than altering the trading token quantities. This approach retains the correct deposit and withdrawal token amounts, adjusting the issuance or burning of LP tokens to reflect the bonding curve's virtual effect, as detailed in section 2.

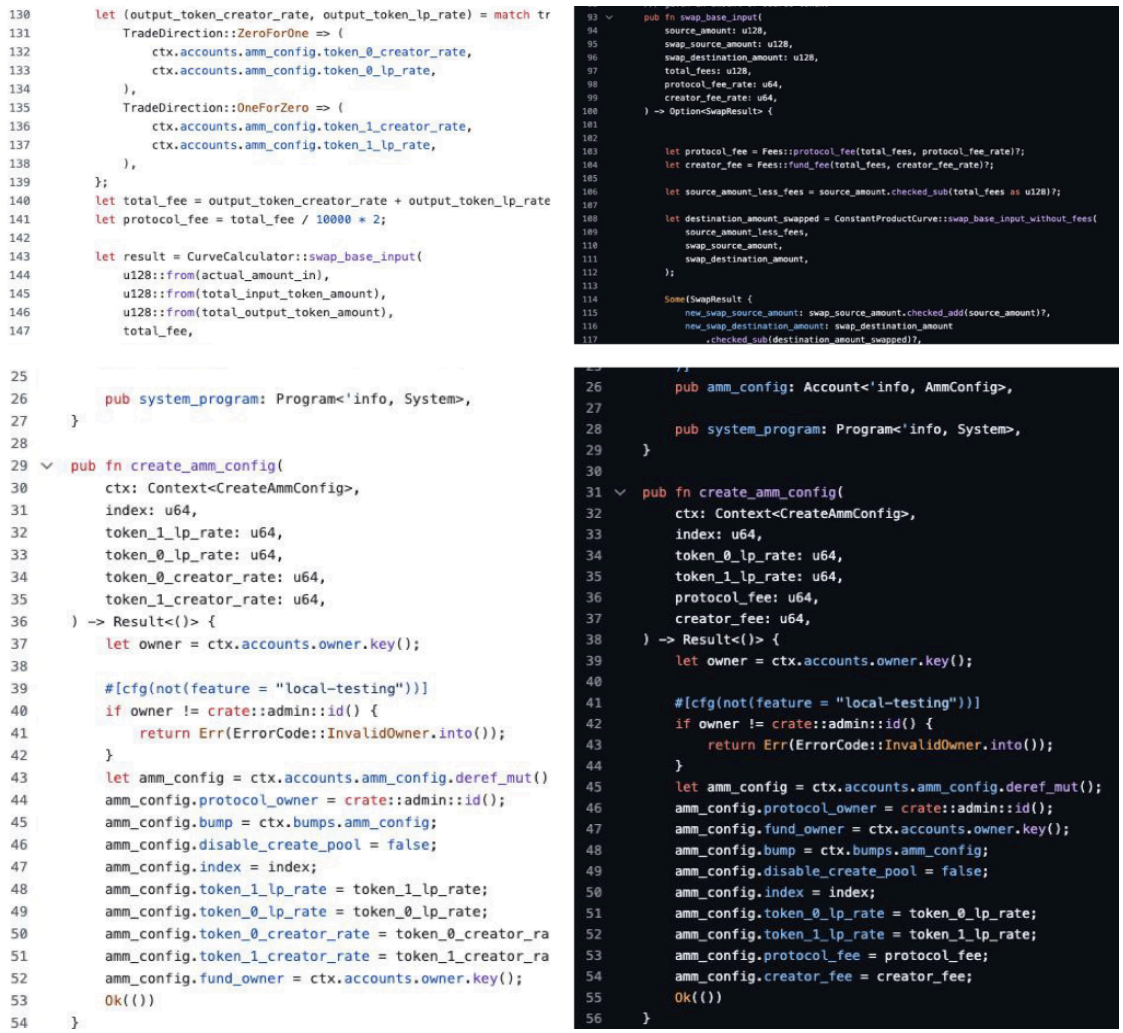
Patch Commit – 639cfe20e9b9027ac2b5714855185d4ad30b1586

2. GOBBLER_02 - Misappropriation Of Fees Between The Involved Parties [Critical]

Description

In the original Raydium protocol, fees are distributed based on trade value, divided among liquidity providers (LPs), the protocol, and a fund, with specific ratios assigned to each. However, the fee structure does not accurately inherit this distribution logic in the Gobbler protocol. Gobbler currently applies a flat fee rate to all trades regardless of size, resulting in:

1. **Misrouted Fees to an Incorrect Fund Account**: Fees were incorrectly routed to an inherited fund account not intended for Gobbler, affecting the intended allocation.
2. **Miscalculated Protocol and LP Fees**: This misallocation resulted in reduced returns for liquidity providers and lower revenue for the protocol itself.



```
let (output_token_creator_rate, output_token_lp_rate) = match tr
    TradeDirection::ZeroForOne => (
        ctx.accounts.amm_config.token_0_creator_rate,
        ctx.accounts.amm_config.token_0_lp_rate,
    ),
    TradeDirection::OneForZero => (
        ctx.accounts.amm_config.token_1_creator_rate,
        ctx.accounts.amm_config.token_1_lp_rate,
    ),
};
let total_fee = output_token_creator_rate + output_token_lp_rate
let protocol_fee = total_fee / 10000 * 2;

let result = CurveCalculator::swap_base_input(
    u128::from(actual_amount_in),
    u128::from(total_input_token_amount),
    u128::from(total_output_token_amount),
    total_fee,
```

```
pub fn swap_base_input(
    source_amount: u128,
    swap_source_amount: u128,
    swap_destination_amount: u128,
    total_fees: u128,
    protocol_fee_rate: u64,
    creator_fee_rate: u64,
) -> Option<SwapResult> {


    let protocol_fee = Fees::protocol_fee(total_fees, protocol_fee_rate)?;
    let creator_fee = Fees::fund_fee(total_fees, creator_fee_rate)?;

    let source_amount_less_fees = source_amount.checked_sub(total_fees as u128)?;

    let destination_amount_swapped = ConstantProductCurve::swap_base_input_without_fees(
        source_amount_less_fees,
        swap_source_amount,
        swap_destination_amount,
    );

    Some(SwapResult {
        new_swap_source_amount: swap_source_amount.checked_add(source_amount)?,
        new_swap_destination_amount: swap_destination_amount
            .checked_sub(destination_amount_swapped)?,
```

```
    pub system_program: Program<'info, System>,
}

pub fn create_amm_config(
    ctx: Context<CreateAmmConfig>,
    index: u64,
    token_1_lp_rate: u64,
    token_0_lp_rate: u64,
    token_0_creator_rate: u64,
    token_1_creator_rate: u64,
) -> Result<()> {
    let owner = ctx.accounts.owner.key();

    #[cfg(not(feature = "local-testing"))]
    if owner != crate::admin::id() {
        return Err(ErrorCode::InvalidOwner.into());
    }
    let amm_config = ctx.accounts.amm_config.deref_mut()
    amm_config.protocol_owner = crate::admin::id();
    amm_config.bump = ctx.bumps.amm_config;
    amm_config.disable_create_pool = false;
    amm_config.index = index;
    amm_config.token_1_lp_rate = token_1_lp_rate;
    amm_config.token_0_lp_rate = token_0_lp_rate;
    amm_config.token_0_creator_rate = token_0_creator_ra
    amm_config.token_1_creator_rate = token_1_creator_ra
    amm_config.fund_owner = ctx.accounts.owner.key();
    Ok(())
}
```

```
    pub amm_config: Account<'info, AmmConfig>,

    pub system_program: Program<'info, System>,
}

pub fn create_amm_config(
    ctx: Context<CreateAmmConfig>,
    index: u64,
    token_0_lp_rate: u64,
    token_1_lp_rate: u64,
    protocol_fee: u64,
    creator_fee: u64,
) -> Result<()> {
    let owner = ctx.accounts.owner.key();

    #[cfg(not(feature = "local-testing"))]
    if owner != crate::admin::id() {
        return Err(ErrorCode::InvalidOwner.into());
    }
    let amm_config = ctx.accounts.amm_config.deref_mut();
    amm_config.protocol_owner = crate::admin::id();
    amm_config.fund_owner = ctx.accounts.owner.key();
    amm_config.bump = ctx.bumps.amm_config;
    amm_config.disable_create_pool = false;
    amm_config.index = index;
    amm_config.token_0_lp_rate = token_0_lp_rate;
    amm_config.token_1_lp_rate = token_1_lp_rate;
    amm_config.protocol_fee = protocol_fee;
    amm_config.creator_fee = creator_fee;
    Ok(())
}
```

Fig 1 – Fee Calculation and Config Creation in Raydium (Left) vs Gobbler (Right)

Recommended Fix

We recommend creating three distinct fee destinations within Gobbler's architecture to align Gobbler's fee distribution with the intended model. Each should share a predefined constant fee (`lp_rate`) with specific ratios assigned as follows:

- **Liquidity Providers**: Allocate a percentage of the trade fee, redeemable through LP tokens, whose value appreciates over time as fees are accumulated.
- **Protocol Fees**: To incentivize users and promote platform usage, assign a configurable percentage of the trade fee for the protocol.
- **Creator Fees**: Set another configurable percentage of the trade fee to support ongoing development and protocol improvements.

These fee destinations should be configurable constants, allowing the protocol to balance incentives between LPs, developers, and protocol maintenance.

Patch Commit – [639cfe20e9b9027ac2b5714855185d4ad30b1586]

6. General Recommendations

As part of the audit process, we identified a non-critical issue, which is summarized below. While this finding does not represent vulnerabilities, addressing them would improve the program's robustness and efficiency. Each issue is assigned a general recommendation (GR) code and detailed accordingly.

1. GOBBLER_GR_01 - Increase Flat Rate Swap Fees For Higher Revenue [Informational]

Description

The Gobbler decentralized exchange (DEX) operates with a constant fee structure initially set by the architect without robust data-driven analysis. To assess optimal fee levels for revenue maximization, the audit team conducted a thorough analysis using a sample of pools on the Jupiter aggregator. Specifically, data was aggregated from:

- The top 12 Jupiter pools with established trading pairs (e.g., SOL-USDC, SOL-JLP) representing approximately 74% of Jupiter's total 7-day volume.
- The top 13 "Flavor of the Week" or volatile pairs (e.g., SOL-MOODEENG, SOL-ACT) totaling around 26% of Jupiter's 7-day volume.

		Fees in base unit								
		11.11k	22.22k	33.33k	55.55k	100k	300k	500k	1m	5m
	Avg. Gobbler swap fee (%)	0.001%	0.003%	0.004%	0.004%	0.008%	0.011%	0.023%	0.033%	0.148%
Established pair (SOL-USDC, SOL-JLP, JUP-USDC, etc.)	Pairs competitive (#)	12	11	10	8	8	3	3	1	1
	7D total potential volume ($b)	$3.6	$3.4	$3.1	$1.5	$0.8	$0.3	$0.3	$0.1	$0.1
	7D total potential Gobbler fees ($k)	$67	$115	$146	$84	$117	$21	$20	$17	$79
	Avg. Gobbler swap fee (%)	0.002%	0.003%	0.005%	0.005%	0.015%	0.006%	0.007%	0.015%	0.076%
Flavor of the week pair (SOL-MOODENG, SOL-ACT, SOL-GNON, etc.)	Pairs competitive (#)	13	13	13	13	13	13	13	12	9
	7D total potential volume ($b)	$1.3	$1.3	$1.3	$1.3	$1.3	$1.3	$1.3	$1.2	$1.1
	7D total potential Gobbler fees ($k)	$6	$13	$19	$31	$56	$164	$268	$430	$1,656
	Avg. Gobbler swap fee (%)	0.000%	0.001%	0.001%	0.002%	0.004%	0.013%	0.021%	0.035%	0.155%
Total	**Pairs competitive (#)**	**25**	**24**	**23**	**21**	**21**	**16**	**16**	**13**	**10**
	7D total potential volume ($b)	**$4.9**	**$4.7**	**$4.4**	**$2.8**	**$2.1**	**$1.6**	**$1.6**	**$1.3**	**$1.2**
	7D total potential Gobbler fees ($k)	**$73**	**$128**	**$165**	**$115**	**$173**	**$185**	**$288**	**$447**	**$1,734**

Fig 2 – Table showing the Gobbler revenue and fees simulation by fitting a sigmoid distribution to the data of the top 12 established pairs & top 13 meme pairs at various constant fee rates.

The audit team then modeled potential transaction volume and liquidity provision for Gobbler with a sigmoid distribution parameterized by the constant fee rate as a variable. The results as depicted in **Fig 2** suggest that:

1. *Established Pairs:* Optimal fee rates align with 30,000 to 40,000 token units, reflecting liquidity providers' lower impermanent loss expectations for these pairs.

2. *Volatile Pairs:* Higher fees (between 1 to 5 million token units) can be leveraged due to the higher impermanent loss associated with these riskier assets, which constitute Gobbler's primary target market.

Recommended Fix

1. ***Manual Fee Adjustment Lever:*** Upon launch, Gobbler should include a feature to adjust fees for established pairs to an optimal level manually. Keeping fees lower than competitor DEXs for these pairs could ensure favorable routing from aggregators, maximizing profitability.
2. ***High Default Fee for Volatile Pairs:*** Set default fees for volatile or memecoin pairs at significantly higher rates (1-5 million units) to encourage deeper liquidity and generate increased revenue through compensating for higher impermanent loss.
3. ***Tiered Fee Structure:*** For more nuanced optimization, consider a tiered fee system with distinct rates for high, medium, and low trade amounts. This structure would help capture more fees from larger trades while addressing fee concerns for established pairs, balancing both liquidity incentives and protocol revenue growth.

7. Conclusion

Mad Shield conducted an extensive audit of Gobbler, utilizing a hands-on methodology that prioritizes a detailed, immersive review of the program. Our team's approach is rooted in active collaboration, working closely with each unique project to identify potential security risks and mitigate vulnerabilities effectively.

Mad Shield's dedication to advancing auditing techniques is clear throughout our process. We consistently apply innovative strategies, allowing us to analyze the code at a granular level, simulate real-world scenarios, and uncover potential risks that traditional audits might miss.

No critical vulnerabilities were identified during the Gobbler audit, and all findings were promptly communicated to the development team. Our recommendations focus on strengthening the codebase to enhance long-term security and resilience. Mad Shield remains committed to setting new standards in smart contract auditing.

References

[1] Hayden Adams, Noah Zinsmeister, Dan Robinson. Uniswap v2 Core. Mar 2020. URL: https://app.uniswap.org/whitepaper.pdf