MONETHIC

Magic Beans OTC

Smart Contract Audit Report

Prepared for:
Magic Beans

Date:
19.06.2024

Version:
Final, for public release

Table of Contents

About Monethic 2

About Client 2

Scoping Details 3

Scope 3

Timeframe 3

Vulnerability Classification 4

Vulnerabilities summary 5

Technical summary 6

1. Improper order type validation leading to potential funds lock in escrow 6
2. Ensure sufficient escrow balance for rent invariant in partially filled orders 7
3. Incorrect receiver of lamports during CancelOffer instruction 7
4. Missing validation of user account in CancelOffer and CancelBid instructions 8
5. Inability to utilize existing wrapped SOL (WSOL) on OpenBid 9
6. Incompatibility with non-associated token accounts on offer creation 10
7. Order parameters validation missing on order creation 11
8. Redundant account closing logic in CloseOffer and CloseBid instructions 11
9. Missing events emissions for critical actions 12
10. Using vector for holding orders 13
11. Add optional argument for minimum amount in lift_offer instruction 14
12. Centralize account settings using singleton pattern for improved configurability 15
13. Use WSOL instead of SOL for improved flexibility and consistency 16
14. Implement emergency pause functionality for protocol safety 17

About Monethic

Monethic is a young and thriving cybersecurity company with extensive experience in various fields, including Smart Contracts, Blockchain protocols (layer 0/1/2), wallets and off-chain components audits, as well as traditional security research, starting from penetration testing services, ending at Red Team campaigns. Our team of cybersecurity experts includes experienced blockchain auditors, penetration testers, and security researchers with a deep understanding of the security risks and challenges in the rapidly evolving IT landscape. We work with a wide range of clients, including fintechs, blockchain startups, decentralized finance (DeFi) platforms, and established enterprises, to provide comprehensive security assessments that help mitigate the risks of cyberattacks, data breaches, and financial losses.

At **Monethic**, we take a collaborative approach to security assessments, working closely with our clients to understand their specific needs and tailor our assessments accordingly. Our goal is to provide actionable recommendations and insights that help our clients make informed decisions about their security posture, while minimizing the risk of security incidents and financial losses.

About Client

Magic Beans is an innovative on-chain escrow system designed to facilitate over-the-counter (OTC) trading of Solana tokens. This protocol supports secure, atomic, peer-to-peer transactions of SPL tokens by allowing users to initiate a trade by depositing their tokens into an on-chain escrow account, effectively "making a market." Counterparties can then complete the trade by depositing their corresponding tokens into the escrow account, ensuring a secure and seamless transaction.

Disclaimer

This report reflects a rigorous security assessment conducted on the specified product, utilizing industry-leading methodologies. While the service was carried out with the utmost care and proficiency, it is essential to recognize that no security verification can guarantee 100% immunity from vulnerabilities or risks.

Security is a dynamic and ever-evolving field. Even with substantial expertise, it is impossible to predict or uncover all future vulnerabilities. Regular and varied security assessments should be performed throughout the code development lifecycle, and engaging different auditors is advisable to obtain a more robust security posture.

This assessment is limited to the defined scope and does not encompass parts of the system or third-party components not explicitly included. It does not provide legal assurance of compliance with regulations or standards, and the client remains responsible for implementing recommendations and continuous security practices.

Scoping Details

The purpose of the assessment was to conduct Smart Contract Audit against Magic Beans Smart Contracts, available through the Github platform.

Scope

The scope covered by the security assessment specifies that the Magic Beans contracts will be audited, the code of which has been shared on the GitHub platform with the **1336d44c1126ffac8e7a1e92322ed8979e194cd1** commit SHA hash.

Additionally, during the assessment, all changes in the code were audited on an ongoing basis, as well as the solutions and suggestions proposed by Monethic. The final SHA commit hash included in the audit was **b5eeade186c047c5fdf9795b42e1077f55e38980.**

No additional internal documentation was provided.

Timeframe

On June 05th Monethic was chosen for the security audit of Magic Beans. Work began immediately.

On June 19th, the report from the Smart Contract security assessment was delivered to the customer.

Vulnerability Classification

All vulnerabilities described in the report were thoroughly classified in terms of the risk they generate in relation to the security of the contract implementation. Depending on where they occur, their rating can be estimated on the basis of different methodologies.

In most cases, the estimation is done by summarizing the impact of the vulnerability and its likelihood of occurrence. The table below presents a simplified risk determination model for individual calculations.

		Impact		
	Severity	**High**	**Medium**	**Low**
Likelihood	**High**	**Critical**	**High**	**Medium**
	Medium	**High**	**Medium**	**Low**
	Low	**Medium**	**Low**	**Low**

Vulnerabilities that do not have a direct security impact, but may affect overall code quality, as well as open doors for other potential vulnerabilities, are classified as **INFORMATIVE**.

Vulnerabilities summary

No.	Severity	Name	Status
1	High	Improper order type validation leading to potential funds lock in escrow	Resolved
2	Low	Ensure sufficient escrow balance for rent invariant in partially filled orders	Resolved
3	Low	Incorrect receiver of lamports during `CancelOffer` instruction	Resolved
4	Low	Missing validation of user account in `CancelOffer` and `CancelBid` instructions	Resolved
5	Informative	Inability to utilize existing wrapped SOL (WSOL) on `OpenBid`	Resolved
6	Informative	Incompatibility with non-associated token accounts on offer creation	Acknowledged
7	Informative	Order parameters validation missing on order creation	Resolved
8	Informative	Redundant account closing logic in `CloseOffer` and `CloseBid` instructions	Resolved
9	Informative	Missing events emissions for critical actions	Resolved
10	Informative	Using vector for holding orders	Resolved
11	Informative	Add optional argument for minimum amount in `lift_offer` instruction	Acknowledged
12	Informative	Centralize account settings using singleton pattern for improved configurability	Resolved
13	Informative	Use WSOL instead of SOL for improved flexibility and consistency	Resolved
14	Informative	Implement emergency pause functionality for protocol safety	Resolved

Technical summary

1. Improper order type validation leading to potential funds lock in escrow

Severity

HIGH

Description

It has been observed that the Magic Beans program does not validate the order type during the closing of orders. This lack of validation allows a `Bid` order to be improperly closed using the `CancelOffer` instruction, leading to user funds allocated in the escrow of the `Bid` order being locked forever. Once the order account is closed by the `CancelOffer` instruction, it becomes impossible to close the associated escrow account, resulting in permanent loss of access to the funds.

Vulnerable Scenarios:

1. **Expired Bid Order Exploited by Malicious Actor**

A malicious actor can exploit the expired `Bid` order by closing it with the `CancelOffer` instruction, causing the funds in the escrow account to be locked indefinitely.

2. **Frontend Error or Phishing Attack:**

An error in the frontend or a phishing attack can trick the user into incorrectly closing a `Bid` order with the `CancelOffer` instruction, leading to the same issue of locked funds in the escrow.

Remediation

To address this vulnerability, the program should implement proper order type validation during the closing of orders. Specifically, the `CancelOffer` instruction should check that the order being closed is of the correct type (i.e., an `Offer` order). If the order type is invalid for the given instruction, the program should reject the operation. What is more, order type validation should also be performed in other operations - `HitBid` and `LiftOffer` instructions.

Status: Resolved

2. Ensure sufficient escrow balance for rent invariant in partially filled orders

Severity

LOW

Description

In the Magic Beans program, there exists a potential edge case during partial filling of orders where the escrow may not have enough balance to cover the rent. The code comments acknowledge this possibility and currently handle it by causing the transaction to fail.

Remediation

To address this issue, it is necessary to implement an invariant check to ensure that the escrow has enough balance to cover the rent before processing partially filled orders. This can be achieved by utilizing the `Rent` struct's `minimum_balance` method, which calculates the minimum balance required to keep an account alive based on its size and rent parameters. By performing this check, the protocol can prevent transaction failures due to insufficient escrow balance and handle the edge case more gracefully.

Status: Resolved

---

3. Incorrect receiver of lamports during `CancelOffer` instruction

Severity

LOW

Description

In the Magic Beans program, it has been observed that during the `CancelOffer` instruction, the receiver of lamports for closing the account is the maker's associated token account (`maker_ata`). However, this approach poses an issue as users cannot retrieve these lamports until they close the associated token account.

Instead, the lamports should be directed directly to the maker's account, not the maker's associated token account. Failing to address this issue can lead to user inconvenience and confusion, as they may encounter difficulties retrieving their lamports.

```
let cpi_accounts_fee = CloseSplAccount {
     account: self.escrow.to_account_info().clone(),
     destination: self.maker_ata.to_account_info().clone(),
     authority: self.order.to_account_info().clone(),
 };

 token::close_account(CpiContext::new_with_signer(
     self.token_program.to_account_info(),
     cpi_accounts_fee,
     &[&seeds],
 ))?;
```

Remediation

To resolve this issue, it is necessary to update the `CancelOffer` instruction to direct the receiver of lamports to the maker's account instead of the maker's associated token account. By making this adjustment, users can retrieve their lamports directly without the need to close the associated token account.

This remediation enhances user experience and streamlines the process of canceling offers, ensuring that users can access their funds more efficiently.

Status: Resolved

4. Missing validation of user account in CancelOffer and CancelBid instructions

Severity

LOW

Description

In the Magic Beans program, it has been observed that the `CancelOffer` and `CancelBid` instructions do not validate the `user_account` parameter. Consequently, it is possible to cancel both bid and offer orders with a different user account, leading to discrepancies in the protocol. Additionally, the `remove_order` helper function lacks validation to check if the removed order exists, resulting in potential inconsistencies between the total amount of orders by the user and the actual orders present in the account. This discrepancy poses a risk to the integrity and accuracy of the protocol's order management system.

Remediation

To address this issue, it is imperative to implement proper validation of the `user_account` parameter in the `CancelOffer` and `CancelBid` instructions. Additionally, validate the existence of the removed order in the `remove_order` helper function to ensure consistency between the total orders by the user and the actual orders present in the account.

By enforcing these validations, the protocol can maintain the integrity of its order management system and prevent discrepancies that could lead to incorrect protocol behavior.

Status: Resolved

---

5. Inability to utilize existing wrapped SOL (WSOL) on `OpenBid`

Severity

INFORMATIVE

Description

The current implementation of the open bid instructions includes an in-program sync native operation. This prevents users from utilizing their existing Wrapped SOL (WSOL) balances. As a result, users are unable to use their pre-existing WSOL, leading to unnecessary token transfers and potential user inconvenience.

```
token::sync_native(CpiContext::new(
    self.token_program.to_account_info(),
    SyncNative {
        account: self.escrow.to_account_info(),
    },
))?;
```

Remediation

Revise the sync native operation in the open bid instructions to accommodate users' existing WSOL balances. The recommended steps are:

1. Check the user's current WSOL balance.
2. If the user does not have enough WSOL, transfer only the required difference to meet the bid requirement.

This approach will optimize the use of existing WSOL, reducing unnecessary transfers and improving the user experience.

Status: Resolved

---

6. Incompatibility with non-associated token accounts on offer creation

Severity

INFORMATIVE

Description

During the offer creation process, the system currently only supports associated token accounts. However, a seller's token may reside in a standard (non-associated) token account, which can have any address. Associated token accounts are created using specific seeds, whereas standard token accounts do not follow this restriction.

This limitation causes incompatibility issues, preventing users who hold tokens in standard accounts from creating offers. This can be the case if users are interacting with a program using SDK, as in such users can specify sender mint account directly.

```
// programs/magicbeans/src/contexts/offer.rs:31
#[account(
    mut,
    associated_token::mint = mint,
    associated_token::authority = seller,
)]
pub seller_ata: Account<'info, TokenAccount>,
```

Remediation

To ensure compatibility with all token accounts, modify the offer creation process to support standard token accounts in addition to associated token accounts. This will allow tokens to be recognized and utilized from any address, thus enhancing the flexibility and usability of the platform for all users.

Status: Acknowledged

---

7. Order parameters validation missing on order creation

Severity

INFORMATIVE

Description

Currently, there is no validation mechanism in place for the rate and amount parameters during the order creation process. This lack of validation allows orders with zero values for these parameters to be created, which can lead to incorrect order entries and potential disruptions in the system's functionality.

Remediation

Implement validation checks for the rate and amount parameters during the order creation process. Specifically, ensure that these parameters are verified against zero values. Orders with a rate or amount of zero should be rejected with an appropriate error message to maintain data integrity and system reliability.

Status: Resolved

---

8. Redundant account closing logic in `CloseOffer` and `CloseBid` instructions

Severity

INFORMATIVE

Description

In the Solana program, redundant logic for closing accounts is observed in the `CloseOffer` and `CloseBid` instructions. The use of the Anchor close macro already handles the closing logic efficiently. However, additional account closing logic is implemented within these instructions, resulting in unnecessary redundancy.

```
#[account(mut, close = signer)]
pub order: Account<'info, Order>,

let order_lamports = self.order.to_account_info().lamports();
self.signer.add_lamports(order_lamports)?;
**self.order.to_account_info().lamports.borrow_mut() = 0;
```

Remediation

To address this issue, remove the redundant account closing logic from the `CloseOffer` and `CloseBid` instructions and rely solely on the Anchor close macro for account closure. By eliminating redundant code, the program becomes more concise, easier to maintain, and less prone to errors.

Status: Resolved

---

9. Missing events emissions for critical actions

Severity

INFORMATIVE

Description

The current Solana program lacks event emissions for critical actions such as order creation, cancellation, and execution. This absence of event logging can lead to difficulties in tracking and auditing operations, making it challenging to maintain a clear and transparent record of state changes and user actions. Moreover, without event emissions, monitoring tools and dApp frontends cannot efficiently respond to changes in the program state, potentially leading to a suboptimal user experience.

Remediation

To address this issue, it is recommended to integrate event emissions into all critical actions within the program. By using Anchor's event macros, events can be emitted whenever significant state changes occur, providing a reliable and consistent way to track and audit actions.

Status: Resolved

---

10. Using vector for holding orders

Severity

INFORMATIVE

Description

The current implementation of the Solana program uses a vector to hold orders, which presents several significant limitations and inefficiencies:

1. **PDA Size Limitation:**

A Program Derived Address (PDA) can hold a maximum of 10KB of data, restricting the maximum number of orders per maker to around 300.

2. **Expensive Reallocation Operations:**

Reallocating memory for vectors is computationally expensive and can degrade performance.

3. **Inefficient Iteration:**

Iterating over large vectors, especially during order removal, can lead to exceeding the maximum Compute Units (CU) or gas limit, causing transaction reverts.

Remediation

A more scalable and CU/Gas-efficient approach is to use a separate PDA for each order. The seed for these PDAs can include the user address and a counter (or even be a random address). The account struct can then contain the user's public key (owner). This design allows for efficient order management and retrieval.

Status: Resolved

---

11. Add optional argument for minimum amount in `lift_offer` instruction

Severity

INFORMATIVE

Description

In the `lift_offer` instruction of the Solana program, it could be beneficial to add an optional argument for `minimum_amount`. While this is an edge case, there are scenarios where a small leftover amount might remain in an order after a partial swap. In such cases, the cost of another transaction to swap this leftover amount may not be worth it for the user.

By providing a `minimum_amount` argument, users can specify the minimum amount that should be left in the order after lifting, preventing unnecessary additional transactions for small leftover amounts.

Remediation

To address this issue, modify the `lift_offer` instruction to include an optional argument for minimum_amount. This argument allows users to specify the minimum amount that should remain in the order after lifting. If the leftover amount after lifting is less than the `minimum_amount`, the instruction should fail, ensuring that the user's specified conditions are met.

Status: Acknowledged

12. Centralize account settings using singleton pattern for improved configurability

Severity

INFORMATIVE

Description

In the current implementation of the Magic Beans program, account settings such as fee receivers and fee basis points (BPS) are hardcoded. This approach limits configurability and requires redeployment of the program whenever changes are needed, which is inefficient and time-consuming.

To address this issue, it is recommended to centralize account settings in a single account using the singleton pattern. By creating a `platform_config` singleton account, all configurations, including fee receivers and fee BPS, can be stored in one place. This allows for easier modification of settings without the need for program redeployment, enhancing efficiency and flexibility.

Remediation

To resolve this issue, follow these steps to implement the singleton pattern for centralizing account settings:

Singleton Account Setup

- Define Platform Config Struct: Create a struct to represent platform configurations, including fee receivers, fee BPS, and any other relevant settings.
- Create Platform Config Account: Initialize a singleton account to store platform configurations using the defined struct.
- Implement Read/Write Operations: Develop functions to read and write platform configurations from/to the singleton account.

Status: Resolved

---

13. Use WSOL instead of SOL for improved flexibility and consistency

Severity

INFORMATIVE

Description

In the Solana program, it is recommended to consider using Wrapped SOL (WSOL) instead of native SOL. WSOL, being a token on the Solana blockchain, behaves similarly to other tokens, and all decentralized exchanges (DEXes) support it. However, using native SOL introduces several issues, such as the requirement to maintain a minimum balance of lamports in accounts.

This minimum balance constraint can lead to transaction failures if the balance falls below the minimum threshold, essentially causing a self-denial of service (DoS) scenario for users.

Additionally, managing both SPL tokens and SOL separately in protocols adds complexity and requires separate logic, reducing flexibility.

Remediation

To address these issues, it is recommended to transition to using WSOL instead of native SOL wherever possible. WSOL provides consistency with other tokens on the Solana blockchain and eliminates the need to manage minimum lamport balances.

By standardizing on WSOL, protocols can simplify their logic and ensure consistent behavior across different token types.

Status: Resolved

14. Implement emergency pause functionality for protocol safety

Severity

INFORMATIVE

Description

It is recommended to introduce an emergency pause mechanism in the Magic Beans protocol to ensure safety and mitigate risks during unforeseen circumstances or emergencies. While implementing such a pause mechanism may impact decentralization, a hybrid approach can be adopted to balance decentralization with protocol safety.

For instance, when the protocol is paused, users should be allowed to close their orders, but no new swaps should be permitted. This approach provides users with the ability to manage their existing positions while preventing further activity that may exacerbate the emergency situation.

Remediation

To address this issue, implement an emergency pause functionality in the Solana protocol with the following characteristics:

Pause Functionality

- Order Closure Only: Allow users to close their existing orders during the pause period to manage their positions.
- Swap Prevention: Disable the ability to initiate new swaps or create new orders while the protocol is paused.
- Admin Access: Grant designated administrators the authority to initiate and lift the pause as necessary.
- Clear Communication: Communicate the activation and deactivation of the pause mechanism clearly to users and stakeholders.

Status: Resolved

END OF THE REPORT