BEOSIN
Web3 Security & Compliance

Manic Trade

Smart Contract Security Audit

No. 202605221825

May 22nd, 2026

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 1 High-risk, 1 Medium-risk, 2 Low-risks and 1 Info items were identified in the Manic Trade project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

- **Project Description:**

The project under audit is a Solana-based on-chain vault protocol developed using the Anchor framework. It consists of a single core program — the Vault Program — responsible for managing user deposits, redemptions, and the settlement and distribution of multiple fee types.

The Vault Program is structured around discrete settlement cycles called Batches. Each batch follows a complete lifecycle from open to close. During an active batch, users may submit deposits or withdrawal requests; once the batch is closed, the operator processes each user's settlement individually, and the conclude_batch instruction finalizes the batch-level fund aggregation and settlement confirmation. All accounting is denominated in shares: deposits are converted to shares at the NAV recorded at the opening of the next batch, while redemptions are priced at the NAV recorded at the opening of the current batch. The spread between the two reflects NAV changes generated by the strategy over the course of the batch.

On the fund management side, the protocol separates capital flows into two independent channels: deposit funds and withdrawal liquidity reserves. User deposits are injected into the vault via the deposit instruction and settled on-chain at the batch level. Withdrawal liquidity is pre-funded by an authorized supply provider through the `add_withdraw_supply` instruction and returned after batch settlement via `withdraw_supply`. When aggregate redemption demand exceeds the available deposit surplus, the market maker covers the shortfall through the `transfer_temp` instruction. If aggregate demand exceeds both the deposit surplus and market-maker-provided liquidity, settlement cannot proceed, exposing the protocol to liquidity shortfall risk.

The protocol implements four fee mechanisms: deposit fee, withdrawal fee, performance fee, and revenue fee. Each fee type is governed by a dedicated FeeDetailAccount that stores the list of fee recipients and their respective allocation ratios. The operator specifies the `performance_fee` at the time a withdrawal request is submitted, while the revenue_fee is calculated automatically during settlement based on per-batch NAV appreciation. The revenue fee is funded by the market maker: during `process_user_batch`, the computed `revenue_fee` is folded into `withdraw_delta_amount`, so users receive their full `requested_withdraw_amount` with no per-claim deduction. Final fee disbursement is executed through four independent collect fee instructions, which distribute accumulated fees to designated recipients in proportion to their configured allocation ratios.

1 Overview

1.1 Project Overview

Project Name	Manic Trade
Project Language	Rust
Platform	Solana
Github	https://github.com/mirrorworld-universe/vault-program-library/tree/version_3 (origin) https://github.com/mirrorworld-universe/vault-program-library (fixed)
Commit	f3998723ed11b92471757adc96761f62b656dfd4 (origin) de09a10976c8a746b22f803cb517334b9bdaef02 (fixed)

1.2 Audit Overview

Audit work duration: Apr 10, 2026 - May 26, 2026

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a function of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
Manic-01	Precision Mismatch in transfer_temp	High	Fixed
Manic-02	Missing Validation	Medium	Fixed
Manic-03	Cross-Batch Shares Double-Spend Vulnerability	Low	Acknowledged
Manic-04	Inconsistent Revenue Fee Calculation	Low	Acknowledged
Manic-05	Redundant Validations	Info	Acknowledged

Finding Details:

[Manic-01] Precision Mismatch in transfer_temp

Severity Level	High
Type	Business Security
Location	transfer_temp.rs#L150-154

Description	In the `transfer_temp` instruction, the CPI transfer call uses `adjust_amount` to convert the amount based on the difference between `DEFAULT_MINT_DECIMALS` and the actual `token_mint_account_decimals` before execution. However, during the actual adjustment, the precision of `transfer_amount` is incorrectly converted in the reverse direction, causing what should be the target precision to be treated as the source precision instead. When the vault's bound token mint decimals differ from `DEFAULT_MINT_DECIMALS`, the actual on-chain token transfer amount does not match the internal bookkeeping amount, resulting in a systematic divergence between the vault's recorded balances and its actual treasury assets. When the token decimals are greater than `DEFAULT_MINT_DECIMALS`, the actual tokens transferred out significantly exceed the bookkeeping deduction, and repeatedly triggering this code path can gradually drain the treasury. Conversely, when token decimals are smaller, the bookkeeping becomes inflated, causing incorrect calculations for subsequent user redemptions.
	`let transfer_amount_adjusted: u64 = adjust_amount(`
	`transfer_amount,`
	`DEFAULT_MINT_DECIMALS,`
	`token_mint_account_decimals,`
	`);`

Recommendation	It is recommended to fix the reversed precision conversion in `adjust_amount` by swapping the source and target precision parameters, ensuring that `DEFAULT_MINT_DECIMALS` is correctly used as the source precision and `token_mint_account_decimals` as the target precision.
Status	**Fixed.** The project team corrected the parameter order of adjust_amount.

[Manic-02] Missing Validation

Severity Level	Medium
Type	Business Security
Location	add_withdraw_supply.rs#L130-134
Description	In the `add_withdraw_supply` instruction, `supply_provider` is declared only as `Signer<'info>` without verifying whether it is an authorized supply provider, allowing any signing account to invoke this instruction. More critically, `token_program` is passed in via `remaining_accounts` without any identity verification. An attacker can substitute a self-deployed malicious program in place of the real SPL Token Program. Since `check_token_mint_account` only validates the correctness of the mint address, a fake `token_program` will completely ignore the mint, from, to, and authority parameters in the `TransferChecked` struct, directly returning Ok without executing any transfer. The attacker can pass in real mint, treasure, and token account accounts to bypass other validations, then use the fake `token_program` to make `transfer_checked` execute as a no-op. As a result, `current_withdraw_treasure_balance += params.supply` succeeds in bookkeeping while the treasury actually receives no tokens. The inflated bookkeeping may be exploited by subsequent `withdraw_supply` or `process_user_batch` operations for excess withdrawals or incorrect settlements, potentially leading to real fund losses when combined with other permission issues. `let transfer_supply_provider_to_vault_detail_cpi_ctx: CpiContext<TransferChecked> = CpiContext::new( token_program_account_info.clone(), transfer_supply_provider_to_vault_detail_cpi_accounts, );`
Recommendation	It is recommended to enforce that `token_program` is the official SPL Token Program in the `add_withdraw_supply` Accounts struct, and add identity verification for `supply_provider`.
Status	**Fixed.** The project team added a global check_program_seed validation to verify PDA legitimacy.

[Manic-03] Cross-Batch Shares Double-Spend Vulnerability

Severity Level	Low
Type	Business Security
Location	request_withdraw.rs
Description	In the request_withdraw instruction, the program only validates the user's share balance via check_shares but does not deduct or lock user_detail.current_shares. The actual deduction is deferred to process_user_batch for unified settlement. Since user_batch_detail account PDAs are seeded with batch_index, each batch corresponds to an independent account with complete cross-batch isolation. When the operator has not yet processed batch N's process_user_batch while batch N+1 has already opened, an attacker can exploit this window to register duplicate redemptions: the user_batch_detail for batch N+1 is a fresh account with performance_fee, total_cancel_requested_withdraw_amount, and other fields all at zero. Consequently, the total_withdraw_and_performance_fee accumulation starts from zero, while check_shares still validates against the original undeducted user_detail.current_shares. The attacker can repeatedly submit withdrawal requests across multiple unprocessed batches using the same share balance, causing total registered redemptions to exceed actually held shares. When process_user_batch eventually settles, the checked_sub operation on user_detail.current_shares underflows, causing the transaction to permanently panic. The affected user_batch_detail is forever stuck in Pending status, vault_batch_detail.processed_batch_user can never equal batch_users, the check_can_conclude_batch precondition in conclude_batch can never pass, and the entire vault's batch processing flow is completely blocked, trapping all other users' funds in the vault.
Recommendation	It is recommended to enforce that all users' user_batch_detail accounts from the previous batch are in Processed status before allowing open_new_batch.
Status	**Acknowledged.**

[Manic-04] Inconsistent Revenue Fee Calculation

Severity Level	Low
Type	Business Security
Location	process_user_batch.rs#L278-301
Description	In the settlement logic of `process_user_batch`, the pricing basis for redemption and the basis for `revenue_fee` collection are inconsistent. The redemption share calculation uses `vault_batch_detail.share_price` (the price at the opening of Batch N), whereas the `revenue_fee` is calculated based on `next_vault_batch_detail.share_price - vault_batch_detail.share_price` (the per-unit NAV appreciation over Batch N).

```
            transfer_withdraw_from_deposit_amount =
user_batch_detail.total_deposit_amount;
            withdraw_delta_amount = request_withdraw_amount
                .checked_sub(user_batch_detail.total_deposit_amount)
                .unwrap();
            withdraw_shares = (withdraw_delta_amount as u128)
                .checked_mul(SCALE_9)
                .unwrap()
                .checked_div(vault_batch_detail.share_price as u128)
                .unwrap() as u64;
            if next_vault_batch_detail.share_price >
vault_batch_detail.share_price {
                let revenue_share_price: u64 = next_vault_batch_detail
                    .share_price
                    .checked_sub(vault_batch_detail.share_price)
                    .unwrap();
                revenue_fee = (withdraw_shares as u128)
                    .checked_mul(revenue_share_price as u128)
                    .unwrap()
                    .checked_div(SCALE_9)
                    .unwrap() as u64;
            };
```

This design produces two opposing fairness problems:

Over-taxation of redeeming users: A user exits at the stale opening price of Batch N and therefore does not benefit from any NAV appreciation generated

	by the strategy during Batch N. Nevertheless, they are charged a `revenue_fee` based on that same appreciation — effectively taxing gains they never received. Under-taxation of continuing holders: If a user accumulated substantial gains prior to Batch N but the NAV declines during Batch N, no `revenue_fee` is charged at all. All historical gains from prior batches fall entirely outside the fee's scope. The root cause is that `revenue_fee` is in essence a per-batch incremental gains tax. Its taxable base should be shares that held through Batch N and benefited from NAV appreciation, not shares that redeemed during Batch N at the opening price. The mismatch at batch boundaries results in an inequitable fee burden between redeeming users and continuing holders within the same batch.
Recommendation	It is recommended to revise the `revenue_fee` calculation logic to align the taxable event with actual realized gains.
Status	**Acknowledged.**

[Manic-05] Redundant Validations

Severity Level	Info
Type	Coding Conventions
Location	Multiple instructions
Description	The codebase contains multiple instances of redundant account passing and validation. For example, in `process_user_batch`, `vault_withdraw_treasure_account_info` is passed in via `remaining_accounts` only to compute its PDA and verify via `check_withdraw_treasure` that its address equals `vault_withdraw_treasure_pda`, then serve as the authority reference for validating `vault_withdraw_treasure_token_account` — but this PDA has already been directly computed in the code via `find_program_address`, making it entirely unnecessary to pass in the account for an additional verification step. Similar redundancies exist in `add_withdraw_supply`, where `check_program_id` is called after `Account::try_from` despite `try_from` already validating the owner internally, among other instances.
Recommendation	It is recommended to optimize the redundant validations to reduce gas costs.
Status	**Acknowledged.**

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.3 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.4 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	SPL Token Standards
		Visibility Specifiers
		Lamport Check
		Account Check
		Signer Check
		Program Id Check
		Deprecated Items
		Redundant Code
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		Returned Value Security
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is a leading blockchain security and compliance technology company established in 2018. Being focused on blockchain ecosystem security and compliance, it has developed a product matrix including Beosin KYT, Beosin Trace, and Stablecoin Monitor, which have obtained international certifications such as ISO 27001 and SOC 2. Beosin's core products have been applied for over 70 intellectual property rights, and the company has participated in the development of multiple international standards related to blockchain security. It was among the first batch of enterprises selected for the Cyberport Incubation Programme. Its business covers professional code security audit services for blockchain ecosystems, anti-money laundering compliance technology services for exchanges, financial institutions, and payment institutions, and virtual asset tracing and investigation services for law enforcement and regulatory authorities.

As one of the earliest companies to apply formal verification to blockchain security, Beosin offers professional blockchain and smart contract security audit services. Beosin has audited over 4,500 smart contracts and blockchain projects and has become the official security partner for several renowned blockchains, including BNB Chain, TON, Soneium, Manta Network, Sonic SVM, and SOON Network.

BEOSIN
Web3 Security & Compliance

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

X
https://x.com/Beosin_com

Email
service@beosin.com

LinkedIn
https://www.linkedin.com/company/beosin