Smart Contract Audit Report for Medley Finance

Testers

1. Or Duan
2. Avigdor Sason Cohen
3. Jakub Heba

Table of Contents

Management Summary

Medley Finance contacted Sayfer to perform a security audit on their Solana programs in 03/2025.

This report includes a review of the Medley Finance Solana smart contracts, developed using Rust and Anchor. The review covers architectural structure, authority models, CPI interactions, SPL token usage, and a list of implementation and security improvement recommendations. All findings and risk ratings are framed exclusively around Solana's PDA authority model, CPI patterns, rent-exempt account requirements, and compute-unit limits.

Over the research period of 4 weeks, we discovered 24 vulnerabilities. 3 of them are marked as critical, as deploying the contract as-is may lead to direct loss of funds and functionality.

Several fixes should be implemented following the report, to ensure the system's security posture is competent.

After a review by the Sayfer team, we certify that all the security issues mentioned in this report have been addressed by the Medley Finance team.

Risk Methodology

At Sayfer, we are committed to delivering high-quality smart contract audits tailored to the blockchain execution environment under review. For Solana programs, our risk model considers the architectural distinctions of Solana's runtime.

Our risk assessment is based on two key factors: **IMPACT** and **LIKELIHOOD**. Impact refers to the potential harm resulting from an issue (e.g., lamport loss, account corruption, program failure). Likelihood considers factors such as the complexity of the program, frequency of user interaction, and surface exposure via cross-program invocations (CPIs).

Given Solana's parallel transaction execution model, compute budget limits, and strict account ownership model, the following additional considerations influence risk:

- Rent-exemption and lamport management for persistent accounts.
- BPF compute budget exhaustion or CPI depth errors.
- Account deserialization failures and runtime constraints.
- Authority misuse or PDA derivation conflicts.

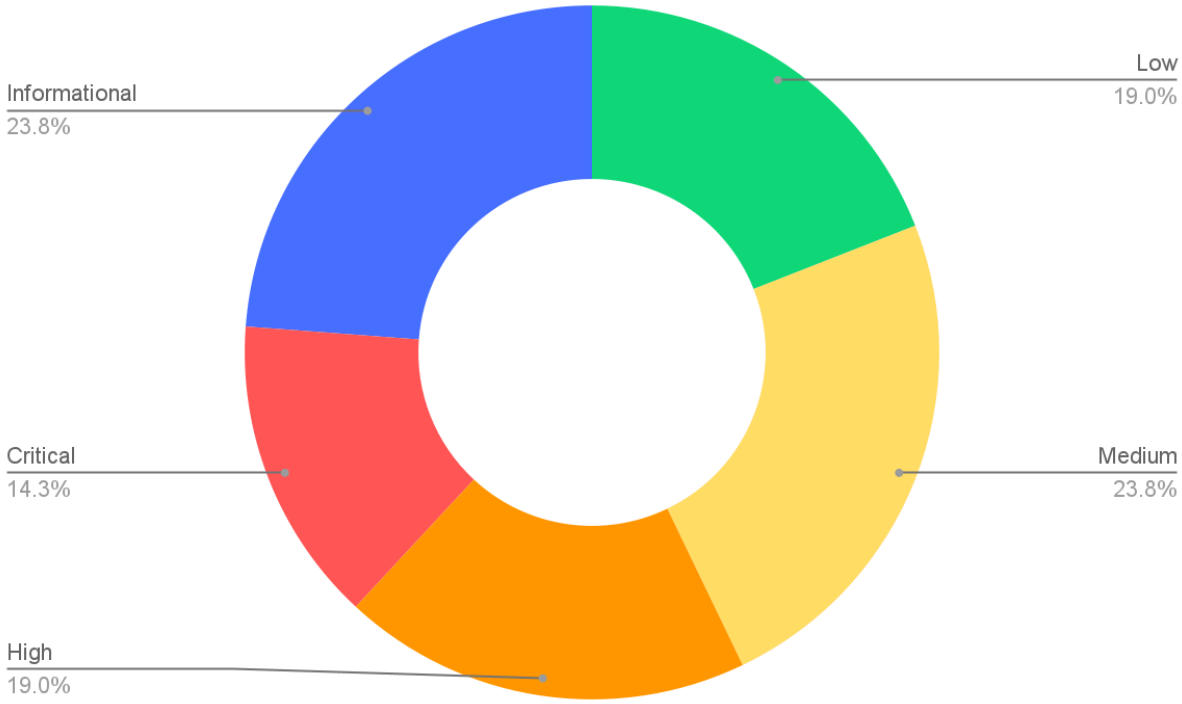
Risk is defined as follows:

Overall Risk Security

IMPACT >	LOW	MEDIUM	HIGH
HIGH	Medium	High	Critical
MEDIUM	Low	Medium	High
LOW	Informational	Low	Medium

LIKELIHOOD >

Vulnerabilities by Risk



Risk	Low	Medium	High	Critical	Informational
# of issues	4	6	4	5	5

Approach

Introduction

Medley Finance contacted Sayfer to perform a security audit on their solana programs.

This report documents the research carried out by Sayfer targeting the selected resources defined under the research scope. Particularly, this report displays the security posture review for the aforementioned contracts.

Scope Overview

Together with the client team we defined the following contract as the scope of the project.

Contract	SHA-256
programs/dmac_contracts/src/lib.rs	`7a3d9865045c550ca4f073be9cc8ddd55c397608389db2de01880ac7ef87185a`
programs/dmac_contracts/src/event.rs	`8fdc3efd8f7ad9032b4fb1910f3adec4e41e677e5d5a7a778af0642eb3c470a1`
programs/dmac_contracts/src/jidl.rs	`12821671c28103ba3d44e09222b21105cc7c956680ae856b4567586606a8db64`

Our tests were performed from 26/03/2025 to 16/04/2025.

Scope Validation

We began by ensuring that the scope defined to us by the client was technically logical. Deciding what scope is right for a given system is part of the initial discussion.

Threat Model

The primary threat to the Medley Finance protocol is the unauthorized access or manipulation of program-controlled accounts resulting in the theft of lamports or SPL tokens.

Solana-specific threats include:

- **CPI Injection**: Malicious input to cross-program invocations leading to unauthorized instruction execution.

- **PDA Mismanagement**: Unsafe or static seeds leading to predictable or hijackable program-derived addresses.
- **Missing Rent-Exemption**: Accounts deleted due to insufficient lamports, leading to loss of critical program state.
- **Stale Oracle Data**: Use of outdated price feeds resulting in unfair swaps or index valuation manipulation.
- **Token-2022 Compatibility Failures**: Incorrect assumptions around token program interfaces may result in failed transfers or broken accounting.
- **Lack of Slippage Controls**: Unchecked swap output allows sandwich or front-running exploitation.
- **Race-condition risk** due to parallel execution on identical mutable accounts.
- **Loss of program state** if any required account falls below rent-exempt balance.
- **Compute-budget exhaustion** leading to transaction failure or denial-of-service.

Additionally, Solana's parallelized transaction model introduces concurrency considerations. If two users trigger operations that depend on mutable shared accounts (e.g., index state), a lack of atomicity or state locking could lead to inconsistent program behavior or race conditions.

Protocol Overview

Protocol Introduction

Medley is a decentralized finance (DeFi) protocol designed to provide users with advanced financial tools and services within the blockchain ecosystem. It offers features such as decentralized lending, borrowing, and yield optimization strategies, enabling users to maximize their crypto assets' potential. By leveraging smart contracts, Medley ensures transparency, security, and efficiency in its operations, aiming to democratize access to financial services and foster a more inclusive financial system.

Security Evaluation

The following test cases were the guideline while auditing the system. This checklist is a modified version of the SCSVS v1.2, with improved grammar, clarity, conciseness, and additional criteria. Where there is a gap in the numbering, an original criterion was removed. Criteria that are marked with an asterisk were added by us.

Architecture, Design and Threat Modeling	Test Name
G1.2	Every introduced design change is preceded by threat modeling.
G1.3	The documentation clearly and precisely defines all trust boundaries in the contract (trusted relations with other contracts and significant data flows).
G1.4	The SCSVS, security requirements or policy is available to all developers and testers.
G1.5	The events for the (state changing/crucial for business) operations are defined.
G1.6	The project includes a mechanism that can temporarily stop sensitive functionalities in case of an attack. This mechanism should not block users' access to their assets (e.g. tokens).
G1.7	The amount of unused cryptocurrencies kept on the contract is controlled and at the minimum acceptable level so as not to become a potential target of an attack.
G1.8	If any 'catch-all' CPI handler (e.g., Anchor's default route) is publicly reachable, it is included in the threat model.
G1.9	Business logic is consistent. Important changes in the logic should be applied in all contracts.
G1.10	Automatic code analysis tools are employed to detect vulnerabilities.
G1.11	The program is compiled with the latest stable Solana SDK version (anchor + solana-program)
G1.12	When using an external implementation of a contract, the most recent version is used.
G1.13	When functions are overridden to extend functionality, the super keyword is used to maintain previous functionality.
G1.14	The order of inheritance is carefully specified.
G1.15	There is a component that monitors contract activity using events.
G1.16	The threat model includes whale transactions.
G1.17	The leakage of one private key does not compromise the security of the entire project.

Policies and Procedures	Test Name
G2.2	The system's security is under constant monitoring (e.g. the expected level of funds).
G2.3	There is a policy to track new security vulnerabilities and to update libraries to the latest secure version.
G2.4	The security department can be publicly contacted and that the procedure for handling reported bugs (e.g., thorough bug bounty) is well-defined.
G2.5	The process of adding new components to the system is well defined.
G2.6	The process of major system changes involves threat modeling by an external company.
G2.7	The process of adding and updating components to the system includes a security audit by an external company.
G2.8	In the event of a hack, there's a clear and well known mitigation procedure in place.
G2.9	The procedure in the event of a hack clearly defines which persons are to execute the required actions.
G2.10	The procedure includes alarming other projects about the hack through trusted channels.
G2.11	A private key leak mitigation procedure is defined.

Upgradability	Test Name
G3.2	Before upgrading, an emulation is made in a fork of the main network and everything works as expected on the local copy.
G3.3	The upgrade process is executed by a multisig contract where more than one person must approve the operation.
G3.4	Timelocks are used for important operations so that the users have time to observe upcoming changes (please note that removing potential vulnerabilities in this case may be more difficult).
G3.5	*initialize()* can only be called once.
G3.6	*initialize()* can only be called by an authorized role through Anchor access-control macros (e.g., *#[access_control(admin_only)]*).
G3.7	The update process is done in a single transaction so that no one can front-run it.
G3.8	Upgradeable contracts have reserved gap on slots to prevent overwriting.
G3.9	The number of reserved (as a gap) slots has been reduced appropriately if new variables have been added.
G3.10	There are no changes in the order in which the contract state variables are declared, nor their types.
G3.11	New values returned by the functions are the same as in previous versions of the contract (e.g. *owner()*, *balanceOf(address)*).

G3.12	The implementation is initialized.
G3.13	The implementation can't be destroyed.

Business Logic	Test Name
G4.2	The contract logic and protocol parameters implementation corresponds to the documentation.
G4.3	The business logic proceeds in a sequential step order and it is not possible to skip steps or to do it in a different order than designed.
G4.4	The contract has correctly enforced business limits.
G4.5	The business logic does not rely on the values retrieved from untrusted contracts (especially when there are multiple calls to the same contract in a single flow).
G4.6	The business logic does not rely on the contract's balance (e.g., *balance == 0*).
G4.7	Sensitive operations do not depend on block data (e.g., *block hash*, *timestamp*).
G4.8	The contract uses mechanisms that mitigate transaction-ordering (front-running) attacks (e.g. pre-commit schemes).
G4.9	The contract does not send funds automatically, but lets users withdraw funds in separate transactions instead.

Access Control	Test Name
G5.2	The principle of the least privilege is upheld. Other contracts should only be able to access functions and data for which they possess specific authorization.
G5.3	New contracts with access to the audited contract adhere to the principle of minimum rights by default. Contracts should have a minimal or no permissions until access to the new features is explicitly granted.
G5.4	The creator of the contract complies with the principle of the least privilege and their rights strictly follow those outlined in the documentation.
G5.5	The contract enforces the access control rules specified in a trusted contract, especially if the dApp client-side access control is present and could be bypassed.
G5.6	Calls to external contracts are only allowed if necessary.
G5.7	Modifier code is clear and simple. The logic should not contain external calls to untrusted contracts.
G5.8	All user and data attributes used by access controls are kept in trusted contracts and cannot be manipulated by other contracts unless specifically authorized.
G5.9	the access controls fail securely, including when a revert occurs.
G5.10	If the input (function parameters) is validated, the positive validation approach (whitelisting) is used where possible.

Communication	Test Name

G6.2	Libraries that are not part of the application (but the smart contract relies on to operate) are identified.
G6.3	Cross-program invocations (CPIs) to untrusted programs are prohibited unless account constraints are strictly validated.
G6.4	Third-party programs do not override error handling or message passing in a way that obscures on-chain logs.
G6.5	CPIs are validated against expected program IDs and account constraints before invocation.
G6.6	The result of each CPI (including returned data) is checked and errors are bubbled up.
G6.7	Program verifies the signer / writable account metas supplied to each instruction and never relies on sysvar::instructions order alone.

Arithmetic	Test Name
G7.2	All arithmetic respects Rust's checked/unwrapped semantics, and explicit panics are avoided.
G7.3	Any unchecked {} arithmetic blocks in Rust do not introduce wrapping or panic conditions.
G7.4	Extreme values (e.g. maximum and minimum values of the variable type) are considered and do not change the logic flow of the contract.
G7.5	Non-strict inequality is used for balance equality.
G7.6	Correct orders of magnitude are used in the calculations.
G7.7	In calculations, multiplication is performed before division for accuracy.
G7.8	The contract does not assume fixed-point precision and uses a multiplier or store both the numerator and denominator.

Denial of Service	Test Name
G8.2	The contract does not iterate over unbound loops.
G8.3	The business logic isn't blocked if an actor (e.g. contract, account, oracle) is absent.
G8.4	The business logic does not disincentivize users to use contracts (e.g. the cost of transaction is higher than the profit).
G8.5	Expressions of functions assert or require have a passing variant.
G8.6	There are no costly operations in a loop.
G8.7	There are no calls to untrusted contracts in a loop.
G8.8	If there is a possibility of suspending the operation of the contract, it is also possible to resume it.
G8.9	If whitelists and blacklists are used, they do not interfere with normal operation of the system.
G8.10	No DoS via compute unit exhaustion, account size overflow, or account lock

	contention.

Blockchain Data	Test Name
G9.2	Any saved data in contracts is not considered secure or private (even private variables).
G9.3	No confidential data is stored in the blockchain (passwords, personal data, token etc.).
G9.4	Contracts do not use string literals as keys for mappings. Global constants are used instead to prevent Homoglyph attack.
G9.5	Contract does not trivially generate pseudorandom numbers based on the information from blockchain (e.g. seeding with the block number).

Compute-Unit Usage and Limitations	Test Name
G10.2	Compute-unit usage is anticipated, defined and has clear limitations that cannot be exceeded.
G10.3	Program logic does not depend on hard-coded compute budgets or lamport fee assumptions.

Clarity and Readability	Test Name
G11.2	The logic is clear and modularized in multiple simple contracts and functions.
G11.3	Each contract has a short 1-2 sentence comment that explains its purpose and functionality.
G11.4	Off-the-shelf implementations are used, this is made clear in comment. If these implementations have been modified, the modifications are noted throughout the contract.
G11.5	The inheritance order is taken into account in contracts that use multiple inheritance and shadow functions.
G11.6	Where possible, contracts use existing tested code (e.g. token contracts or mechanisms like *ownable*) instead of implementing their own.
G11.7	Consistent naming patterns are followed throughout the project.
G11.8	Variables have distinctive names.
G11.9	All storage variables are initialized.
G11.10	Functions with specified return type return a value of that type.
G11.11	All functions and variables are used.
G11.12	*require* is used instead of *revert* in *if* statements.
G11.13	The *assert* function is used to test for internal errors and the *require* function is

	used to ensure a valid condition in input from users and external contracts.
G11.14	Assembly code is only used if necessary.

Test Coverage	Test Name
G12.2	Abuse narratives detailed in the threat model are covered by unit tests.
G12.3	Sensitive functions in verified contracts are covered with tests in the development phase.
G12.4	Implementation of verified contracts has been checked for security vulnerabilities using both static and dynamic analysis.
G12.5	Contract specification has been formally verified.
G12.6	The specification and results of the formal verification is included in the documentation.

Decentralized Finance	Test Name
G14.1	The lender's contract does not assume its balance (used to confirm loan repayment) to be changed only with its own functions.
G14.2	Functions that move lender balances are protected against CPI chaining that could manipulate balances during flash-loan-style operations on Solana.
G14.3	Flash loan functions can only call predefined functions on the receiving contract. If it is possible, define a trusted subset of contracts to be called. Usually, the sending (borrowing) contract is the one to be called back.
G14.4	If it includes potentially dangerous operations (e.g. sending back more SOL/SPL tokens than borrowed), the receiver's function that handles borrowed SOL or tokens can be called only by the pool and within a process initiated by the receiving contract’s owner or another trusted source (e.g. multisig).
G14.5	Calculations of liquidity pool share are performed with the highest possible precision (e.g. if the contribution is calculated for SOL it should be done with 9 digit precision - for lamports). The dividend must be multiplied by the 10 to the power of the number of decimal digits (e.g. dividend * 10^9 / divisor).
G14.6	Rewards cannot be calculated and distributed within the same function call that deposits tokens (it should also be defined as non-re-entrant). This protects from momentary fluctuations in shares.
G14.7	Governance contracts are protected from flash loan attacks. One possible mitigation technique is to require the process of depositing governance tokens and proposing a change to be executed in different transactions included in different blocks.
G14.8	When using on-chain oracles, contracts are able to pause operations based on the oracles’ result (in case of a compromised oracle).

G14.9	External contracts (even trusted ones) that are allowed to change the attributes of a project contract (e.g. token price) have the following limitations implemented: thresholds for the change (e.g. no more/less than 5%) and a limit of updates (e.g. one update per day).
G14.10	Contract attributes that can be updated by the external contracts (even trusted ones) are monitored (e.g. using events) and an incident response procedure is implemented (e.g. during an ongoing attack).
G14.11	Complex math operations that consist of both multiplication and division operations first perform multiplications and then division.
G14.12	When calculating swap prices (e.g. SOL ↔ SPL token) the numerator and denominator are multiplied by reserves, as done in constant-product AMMs on Solana (e.g., Orca).

Security Assessment Findings

Inverted Token Validation

ID	SAY-01
Status	Fixed
Risk	Critical
Business Impact	The inverted require statement will reject legitimate token swaps while accepting incorrect ones, completely breaking the core token swapping functionality and allowing attackers to swap arbitrary tokens not included in the index.
Location	- lib.rs - swap_to_tkn(Context<SwapToTkn>, Vec<u8>) - swap_to_sol(Context<SwapToSol>, u64, Vec<u8>)
Description	The code implements an inverted comparison operator when validating token mints during swap operations. Instead of checking that the provided token matches the expected one in the index, it requires them to be different. • lib.rs:412-415, 662-665 `require!( index_info.index_tokens[token_index].mint ≠ expected_token_mint.key(), ErrorCode::IncorrectTokenMint );`
Mitigation	Invert the comparison operator from `!=` to `==` to correctly validate that the token being swapped matches the expected token.

Decimal Precision Risk in Swaps

ID	SAY-02
Status	Fixed
Risk	Critical
Business Impact	The failure to account for token decimal precision will cause calculation issues when handling tokens with non-standard decimal places, leading to direct and substantial fund loss for users and the protocol.
Location	- `lib.rs; swap_to_sol(Context<SwapToSol>, u64, Vec<u8>)`
Description	`swap_to_sol(...)` accepts an external `token_amount_in_decimals` parameter and uses it directly in token transfers without any validation or adjustment based on the token's decimal precision. This is problematic because Solana tokens have widely varying decimal precisions - SOL uses 9 decimals, USDC uses 6, and many bridged tokens use 18 decimals. When processing a token with 18 decimals as if it had 9, amounts would be off by a factor of 10^9 - a billion times too large or too small. This mathematical miscalculation is deterministic and guaranteed to occur whenever tokens with non-standard decimals are processed.
Mitigation	Make sure that the protocol accounts for token decimal precision by modifying the `IndexToken` struct to store decimal information and implementing appropriate normalization when calculating transfer amounts. Retrieve this information when tokens are added to the index and incorporate decimal-aware calculations throughout all token transfer operations.

No Recovery Path for Failed Multi-Step Operations

ID	SAY-03
Status	Fixed
Risk	Critical
Business Impact	Funds can become permanently locked in intermediate PDAs if swap operations fail midway through execution, with no mechanism to recover these funds in case of network disruptions or other failures.
Location	- lib.rs - swap_to_tkn(Context<SwapToTkn>, Vec<u8>) - rebalance_index(Context<RebalanceIndex>, u64, Vec<u8>)
Description	The protocol implements complex multi-step operations that span multiple transactions, but provides no recovery mechanism if these operations fail part way through. For example, in `swap_to_tkn( ... )`, funds are transferred to temporary accounts before swaps are executed, but if these swaps fail, there's no way to recover the transferred funds. • lib.rs:430-447

```
invoke_signed(
   &transfer(
       &ctx.accounts.program_authority_pda.key(),
       &ctx.accounts.wsol_token_account.key(),
       // swap_to_tkn_info.sol_to_swap,
       sol_to_swap,
   ),
   &[

ctx.accounts.program_authority_pda.to_account_info().clone(),
       ctx.accounts.wsol_token_account.to_account_info().clone(),
       ctx.accounts.system_program.to_account_info().clone(),
   ],
   &[&[
       PROGRAM_AUTHORITY_SEED,
       ctx.accounts.index_mint.key().as_ref(),
       &[bump],
   ]], // Sign with PDA's seeds
```

	``)?;``` If any step fails after this initial transfer, the funds remain in intermediate accounts with no recovery path.
Mitigation	Implement emergency recovery functions that allow administrators to rescue funds from intermediate states in case of multi-step operation failures. Add timeouts to operations and automatic cleanup mechanics if certain operations aren't completed within expected timeframes.

Zero Rent-Exempt Lamports for Account Creation

ID	SAY-04
Status	Fixed
Risk	High
Business Impact	Accounts created without rent-exemption will be deleted after two epochs, potentially causing transaction failures at unpredictable times and permanently locking user funds when they attempt operations during account deletion.
Location	- `lib.rs; create_index(Context<CreateIndex>, String, String, String, Vec<IndexToken>, Vec<FeeCollector>, Option<u64>, Option<u64>)`
Description	The code explicitly sets `rent_exempt_lamports` to zero when creating program authority PDAs, ignoring the Solana requirement for rent exemption. This approach violates Solana's account model where accounts must maintain a minimum balance to avoid deletion • `lib.rs:175-188` `let space = 0; // No extra data, just a pure System Account` `let rent_exempt_lamports = 0;` `let binding = ctx.accounts.index_mint.key();` `let seeds: &[&[u8]] = &[PROGRAM_AUTHORITY_SEED, binding.as_ref()];` `let (pda_pubkey, bump) = Pubkey::find_program_address(seeds, ctx.program_id);` `let create_account_ix = create_account(` `  &payer.key(),` `  &pda_pubkey,` `  rent_exempt_lamports,` `  space as u64,` `  &system_program.key(), // Set owner as System Program` `);` Without proper rent exemption, these accounts will be purged by the runtime, causing critical protocol functions to fail unexpectedly.

Mitigation	Calculate the proper rent-exempt lamports amount based on the account size and provide this value when creating accounts. Replace the hardcoded zero with the calculated value.

Infrequent Index Valuation Updates

ID	SAY-05
Status	Fixed
Risk	High
Business Impact	The index's value becomes increasingly inaccurate as underlying asset prices change, creating systematic arbitrage opportunities where users can profit by buying undervalued or selling overvalued index tokens at the expense of other users.
Location	- `lib.rs; buy_index(Context<BuyIndex>, u64)`
Description	The protocol tracks index value in the `total_value` field, but this value is only updated during user-initiated transactions and not based on current market prices of the underlying assets. When market prices change, the stored index value becomes outdated, creating price discrepancies. • `lib.rs:359-351` `// Update state` `index_info.total_value += deposited_sol_in_usd;` `index_info.total_supply += tokens_to_mint;` Without regular updates based on current asset prices, the index value can significantly deviate from the true market value of its underlying assets
Mitigation	Implement a mechanism to update the index's total value based on current market prices of underlying assets, either through a dedicated update function that can be called regularly or by calculating real-time values at the point of user operations using price oracles.

Token standard mismatch in swap operations

ID	SAY-06
Status	Fixed
Risk	High
Business Impact	The protocol may fail when interacting with tokens that use different standards (Token2022 vs standard SPL), preventing certain tokens from being included in indices and potentially causing fund loss during swap operations.
Location	- lib.rs - swap_to_tkn(Context<SwapToTkn>, Vec<u8>) - swap_to_sol(Context<SwapToSol>, u64, Vec<u8>)
Description	The code inconsistently handles different token standards, hardcoding assumptions about which standard to use for different operations. The index token uses Token2022 but swap operations use the standard token program, creating potential incompatibilities. • lib.rs:1157-1159; struct CreateIndex `pub price_update: Account<'info, PriceUpdateV2>, // Pyth price feed account` `pub token_program: Program<'info, Token2022>, // SPL Token program` `pub system_program: Program<'info, System>, // System program` • lib.rs:1311; struct SwapToTkn `pub token_program: Interface<'info, TokenInterface>` This separation assumes external tokens always use the standard token program, which will fail when interacting with Token2022 tokens, an increasingly common standard on Solana.
Mitigation	Implement a flexible approach that can handle both token standards which standard a token uses at runtime. Use the `TokenInterface` approach to accept any token program that implements the standard interface, and adjust operations based on the detected token program.

Missing Slippage Protection in Jupiter Swaps

ID	SAY-07
Status	Fixed
Risk	High
Business Impact	Swap transactions are vulnerable to front-running and sandwich attacks, allowing malicious actors to extract value from users' trades by manipulating market conditions between transaction submission and execution.
Location	- `lib.rs;` - `swap_on_jupiter<'_>(&[AccountInfo], Program<'info, Jupiter>, Vec<u8>, &Pubkey)` - `rebalance_index(Context<RebalanceIndex>, u64, Vec<u8>)`
Description	The protocol integrates with Jupiter for token swaps but implements no slippage protection or minimum output validation. When calling the Jupiter swap function, there's no validation of the output amount received against any minimum threshold. Without slippage protection, users can receive significantly less value than expected if prices move before their transaction executes, or if their transaction is front-run. The same issue can be found in `rebalance_index( ... )`.
Mitigation	Implement minimum output validation after swaps by comparing the expected output amount to the actual received amount. Either use Jupiter's built-in slippage protection parameters in the swap instruction data or add a post-swap verification.

Excessive Staleness Window

ID	SAY-08
Status	Fixed
Risk	Medium
Business Impact	The 10-minute staleness window for price feeds enables potential oracle manipulation attacks during significant market movements, allowing attackers to execute trades with outdated prices at the expense of the protocol.
Location	- `lib.rs; get_token_price(&Account<PriceUpdateV2>, &str)`
Description	`get_token_price(...)` accepts price feed data up to 10 minutes old (600,000 milliseconds), which is substantially longer than standard practice in DeFi protocols. • lib.rs:73 `let price = price_update.get_price_no_older_than(&Clock::get()?, 600000, &feed_id_bytes)?;` This extended window increases vulnerability to price manipulation, especially in volatile markets. Even a moderate 5% price movement within this window could create significant arbitrage opportunities for attackers who can time their transactions accordingly.
Mitigation	• Reduce the maximum staleness period to 1-2 minutes (60,000-120,000 milliseconds) to ensure prices more accurately reflect current market conditions. • Add additional checks for extreme price movements, especially for highly volatile assets.

No Admin Rotation

ID	SAY-09
Status	Fixed
Risk	Medium
Business Impact	If the admin key is compromised, the entire protocol is permanently compromised with no recovery path, creating a single point of failure for the system's security and governance.
Location	- Structural
Description	The protocol lacks the ability to change the admin address once it is set during initialization. The hardcoded admin key combined with its storage in state lead to an overly rigid structure, which prevents protocol governance evolution, as ownership cannot be transferred to multisig or DAO structures as the protocol matures. • lib.rs:106-117
Mitigation	Implement an admin rotation functionality that allows the current admin to transfer their privileges to a new address.

```rust
const ADMIN: Pubkey =
pubkey!("2LYa8F6T2iPd4uaxM7hu3ctKXXtHnBPgP5YzCETrFgiT");
const TOKEN_2022_PROGRAM_ID: Pubkey =
pubkey!("TokenzQdBNbLqP5VEhdkAS6EPFLC1PHnBqCXEpPxuEb");

pub fn initialize(ctx: Context<Initialize>, admin: Pubkey) -> Result<()>
{
    require!(ADMIN == *ctx.accounts.admin.key, ErrorCode::Unauthorized);

    let program_state = &mut ctx.accounts.program_state;
    program_state.admin = admin;
    program_state.bump = ctx.bumps.program_state;

    Ok(())
}
```

No Emergency Pause

ID	SAY-10
Status	Fixed
Risk	Medium
Business Impact	During critical security incidents or market emergencies, protocol operators cannot temporarily suspend operations, leaving users exposed to potential loss of funds.
Location	- `Structural`
Description	While the protocol includes an index status field that is checked in multiple operations, it is impossible to change the default active state (0). This creates a structural weakness in the protocol's emergency response capabilities.
Mitigation	• Implement a pause/unpause function accessible only to the admin or another governance mechanism that can change the index status value. • Add an appropriate status enum with clear states (Active, Paused) rather than using numeric values.

Missing Index Token Verification

ID	SAY-11
Status	Fixed
Risk	Medium
Business Impact	Without proper verification between burned tokens and the index account, an attacker might be able to burn tokens from one index to receive assets from another index under specific conditions.
Location	- `lib.rs; sell_index(Context<SellIndex>, u64)`
Description	`sell_index( ... )` does not verify that the index tokens being burned correspond to the provided `index_info` account. The function readily accepts the token burn and calculates redemption values without confirming this critical relationship.
Mitigation	• Add explicit verification that the burn matches the expected burn for the provided `index_info` account. • Create a direct connection between these accounts through appropriate seed derivation and validation.

No Weight Validation during Rebalancing

ID	SAY-12
Status	Fixed
Risk	Medium
Business Impact	If rebalancing weights don't sum to 100%, the index composition would be permanently imbalanced, creating accounting errors and incorrect valuations that affect all users.
Location	- `lib.rs; rebalance_index_start(Context<RebalanceIndexStart>, Vec<u64>)`
Description	`rebalance_index_start( ... )` doesn't verify that the new weights sum to 100%, unlike the initial validation performed during index creation. Without this, an admin could set weights that don't properly sum to 100%, permanently damaging the index's accounting.
Mitigation	Add the weight validation logic from `create_index( ... )` to `rebalance_index_start( ... )`. • `lib.rs:144-148` `let total_weight: u64 = index_tokens .iter() .map(|index_token| index_token.weight) .sum(); require!(total_weight == 100_00, ErrorCode::InvalidTokenWeight);`

Unbounded Fee

ID	SAY-13
Status	Fixed
Risk	Medium
Business Impact	A malicious or compromised admin could set fees to extreme levels (up to 100% or higher), effectively stealing all user deposits.
Location	- `lib.rs; create_index(Context<CreateIndex>, String, String, String, Vec<IndexToken>, Vec<FeeCollector>, Option<u64>, Option<u64>)`
Description	The platform fee parameter has no upper bound, allowing it to be set to arbitrary values. While the default is a reasonable 100 basis points (1%), an admin could set this to 10000 (100%) or even higher. • lib.rs:230-232 `index_info.platform_fee_bps = platform_fee_bps.unwrap_or(100);` `ctx.accounts.initialize_token_metadata(name, symbol, uri)?;`
Mitigation	Decide a maximum cap on platform fees (e.g., 500 basis points or 5%) and add logic to enforce this maximum.

Incorrect Token-2022 Implementation

ID	SAY-14
Status	Fixed
Risk	Low
Business Impact	Improper implementation of the Token-2022 metadata extension could cause integration issues with wallets, explorers, and other ecosystem tools, potentially affecting user experience and token functionality.
Location	- `lib.rs:1147`
Description	The code incorrectly configures the `metadata_pointer` extension by setting the metadata address to the mint itself, which violates the Token-2022 standard that requires metadata to be stored in a separate account. • `lib.rs:1147` `extensions::metadata_pointer::metadata_address = index_mint,`
Mitigation	• Create a separate account for token metadata and configure the metadata_pointer extension to point to this separate account, following the Token-2022 standard guidelines. • Alternatively, consider using the built-in metadata capability of Token-2022 without the pointer extension.

Fixed Fee Can Underflow

ID	SAY-15
Status	Fixed
Risk	Low
Business Impact	Transactions could unexpectedly fail or attackers could exploit edge cases where the amount after fees becomes negative, leading to underflow and potential loss of funds.
Location	- `lib.rs; buy_index(Context<BuyIndex>, u64)`
Description	The new reversion introduces a fixed swap fee but doesn't properly validate that the input amount is sufficient to cover both the fixed and percentage-based fees before calculation. While the code checks that `amount_after_fee ≠ 0`, it doesn't verify that `amount_in_lamports > fee_in_lamports + FIXED_SWAP_FEE` before performing the subtraction, which could lead to arithmetic underflow in the calculation if the sum of fees exceeds the deposit amount, reverting the transaction at the end.
Mitigation	We recommend adding an explicit check to ensure the input amount is greater than the combined fees before calculation.

Duplicate Event Emission

ID	SAY-16
Status	Fixed
Risk	Low
Business Impact	Duplicate events could confuse off-chain indexers and monitoring systems, causing transaction history inconsistencies and potentially incorrect analytics reporting.
Location	- `lib.rs; buy_index(Context<BuyIndex>, u64)`
Description	`buy_index( ... )` emits `DmacSwapToTokenStartEvent` twice with identical parameters. • `lib.rs:371-375, 384-388` `emit!(DmacSwapToTokenStartEvent {` `  index_mint: ctx.accounts.index_mint.key(),` `  sol_to_swap: swap_to_tkn_info.sol_to_swap,` `  tokens: swap_to_tkn_info.swapped_tokens.len() as u64,` `});` `...` `emit!(DmacSwapToTokenStartEvent {` `  index_mint: ctx.accounts.index_mint.key(),` `  sol_to_swap: swap_to_tkn_info.sol_to_swap,` `  tokens: swap_to_tkn_info.swapped_tokens.len() as u64,` `});`
Mitigation	Remove one of the duplicate event emissions to ensure each event is emitted exactly once per operation.

Leftover Function

ID	SAY-17
Status	Fixed
Risk	Informational
Business Impact	This finding is purely informational.
Location	- `lib.rs; check_is_transferring(&Context<TransferHook>)`
Description	While the Token-2022 transfer hook extensions have been removed, the supporting `check_is_transferring( ... )` still remains in the code.
Mitigation	• Remove `check_is_transferring( ... )` and any related code that supported the transfer hook implementation. • Remove the `TransferHook` account structure.

Misleading Comment

ID	SAY-18
Status	Fixed
Risk	Informational
Business Impact	This finding is purely informational.
Location	- `lib.rs:1219`
Description	The codebase contains comments regarding token program types, referring to Token2022 program declarations as "SPL Token program". • `lib.rs:1219` `pub token_program: Program<'info, Token2022>, // SPL Token program`
Mitigation	Correct the comment.

Unexplained ‘Magic’ Number

ID	SAY-19
Status	Fixed
Risk	Informational
Business Impact	This finding is purely informational.
Location	- `lib.rs; get_rebalance_info(&Vec<u64>, &Vec<IndexToken>)`
Description	`get_rebalance_info( ... )` uses the number `10001` as a default return value without any explanation of its significance or meaning. • `lib.rs:1982-1983` `// If neither negative nor positive diffs exist, return (usize::MAX, 10001)` `(usize::MAX, 10001)`
Mitigation	• Replace magic numbers with named constants that clearly express their meaning and purpose. • Alternatively, clearly explain the significance of the number in a comment.

Unused Program State

ID	SAY-20
Status	Fixed
Risk	Informational
Business Impact	This finding is purely informational.
Location	- `lib.rs` - `buy_index(Context<BuyIndex>, u64)` - `sell_index(Context<SellIndex>, u64)`
Description	`buy_index( ... )` and `sell_index( ... )` load the `program_state` account but don't use it for any logical operations.
Mitigation	• Remove `program_state` from the `Context` structure if not necessary for the function's logic • Alternatively, implement consistent validation against `program_state` values if the intent is to perform administrative checks.

Unused Parameter in `swap_on_jupiter<'_>( ... )`

ID	SAY-21
Status	Fixed
Risk	Informational
Business Impact	This finding is purely informational.
Location	- `lib.rs; swap_on_jupiter<'_>(&[AccountInfo], Program<'info, Jupiter>, Vec<u8>, &Pubkey)`
Description	`swap_on_jupiter<'_>( ... )` accepts a `program_id` parameter but never uses it for validation or any other purpose.
Mitigation	Either remove the unused parameter or properly validate that the program ID matches the expected Jupiter program ID before proceeding with the swap.

Deprecated Findings

The following findings do not appear to be relevant to the most recent reversion of the code sent to us, either because they were fixed, or the code had significantly changed, but were included in the report for completeness. These findings were not included in the vulnerability breakdown or cout given above.

Premature Token Minting

ID	SAY-22
Status	Fixed
Risk	Critical
Business Impact	Users can receive index tokens before the underlying assets are actually acquired, potentially creating unbacked tokens if subsequent swap operations fail, leading to inflation that devalues all existing index tokens.
Location	- `lib.rs; buy_index(Context<BuyIndex>, u64)`
Description	The protocol mints index tokens to users immediately during the `buy_index( ... )`, before the corresponding assets are purchased through swap operations. If the swap operations fail after minting, users would hold tokens not backed by assets. • `lib.rs:334-358`

```
// Mint tokens to the user
let tokens_to_mint: f64 = if index_info.total_value == 0.0 {
   deposited_sol_in_usd
} else {
   (deposited_sol_in_usd * index_info.total_supply) /
index_info.total_value
};

if tokens_to_mint > 0.0 {
   token_2022::mint_to(
       CpiContext::new(
           ctx.accounts.token_program.to_account_info(),
           token_2022::MintTo {
               mint: ctx.accounts.index_mint.to_account_info().clone(),
               to: ctx.accounts.user_token_account.to_account_info(),
               authority:
ctx.accounts.authority.to_account_info().clone(),
           },
```

```
        ),
        (tokens_to_mint * LAMPORTS_PER_SOL as f64) as u64,
    )?;
    msg!(
        "Minted {} tokens to user: {}",
        tokens_to_mint,
        ctx.accounts.user.key()
    );
}
```

Mitigation	Delay token minting until after successful asset acquisition. Implement a two-phase process where user deposits are first used to acquire underlying assets, and only after successful acquisition, mint the corresponding index tokens to the user.

Risk of Race Conditions in Swap Operations

ID	SAY-23
Status	Fixed
Risk	Critical
Business Impact	Multiple users' swap operations could interfere with each other due to shared account state, potentially allowing one user to complete another user's operation and steal their tokens in a race condition scenario.
Location	- `lib.rs; struct SwapToTknStart`
Description	The code uses shared PDAs without user-specific context for tracking swap progress, creating potential race conditions in concurrent operations. The swap tracking PDA is created with seeds that don't include the user's pubkey, making it shared across all users. • lib.rs:1219-1228 `#[account(` `  init,` `  payer = admin,` `  space = 8 + size_of::<SwapToTokenInfo>() + size_of::<bool>() + index_info.index_tokens.len(),` `  seeds = [` `  b"swap_to_tkn",` `  index_mint.key().as_ref()` `  ],` `  bump,` `)]`
Mitigation	Include the user's pubkey in the seed for swap-related PDAs to isolate operations between users.

Improper Error Handling in transfer hook

ID	SAY-24
Status	Fixed
Risk	Low
Business Impact	The use of panic-based error handling in the transfer hook could create unpredictable transaction execution effects, potentially exploitable in complex transaction batches where partial execution might occur.
Location	- `lib.rs; execute(Context<TransferHook>, u64)`
Description	The transfer hook implementation uses a panic to prevent unauthorized transfers instead of returning a proper error. This approach is problematic as it leaves transactions in an indeterminate state and doesn't provide meaningful debugging information. • `lib.rs:988-994` `#[interface(spl_transfer_hook_interface::execute)]` `pub fn execute(ctx: Context<TransferHook>, _amount: u64) -> Result<()> {` `  // Fail this instruction if it is not called from within a transfer hook` `  check_is_transferring(&ctx)?;` `  panic!("Transfer not allowed");` `}`
Mitigation	Replace the panic with a proper error return to provide clear feedback and ensure consistent transaction handling.

We are available at security@sayfer.io
If you want to encrypt your message please use our public PGP key:
https://sayfer.io/pgp.asc
Key ID: 9DC858229FC7DD38854AE2D88D81803C0EBFCD88

Website: https://sayfer.io
Public email: info@sayfer.io
Phone: +972-559139416