BLOCKAPEX

SMART CONTRACT SECURITY

V 1.0

DATE: 15th APR 2024

PREPARED FOR: POPFI FINANCE

SECURITY REPORT

BLOCKAPEX VERIFIED

About BlockApex

Founded in early 2021, is a security-first blockchain consulting firm. We offer services in a wide range of areas including Audits for Smart Contracts, Blockchain Protocols, Tokenomics along with Invariant development (i.e., test-suite) and Decentralized Application Penetration Testing. With a dedicated team of over 40+ experts dispersed globally, BlockApex has contributed to enhancing the security of essential software components utilized by many users worldwide, including vital systems and technologies.

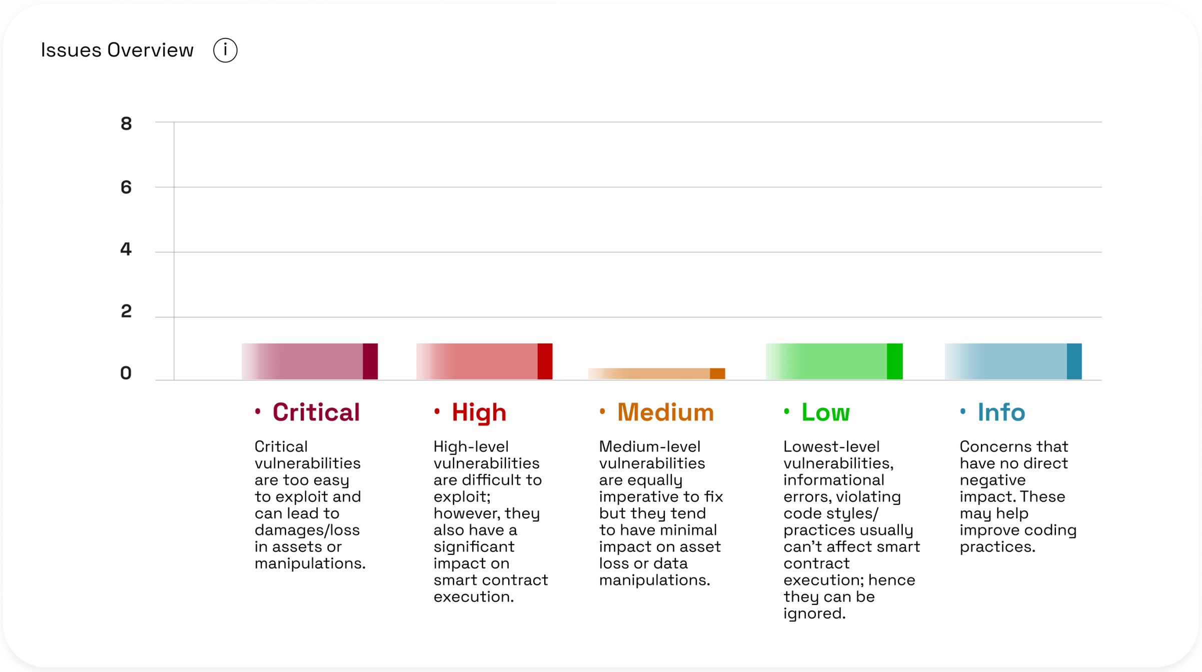
BlockApex has a focus on blockchain security, maintaining an expertise hub to navigate this dynamic field. We actively contribute to security research and openly share our findings with the community. Our work is available for review at our public repository, showcasing audit reports and insights into our innovative practices.

To stay informed about BlockApex's latest developments, breakthroughs, and services, we invite you to follow us on Twitter and explore our GitHub. For direct inquiries, partnership opportunities, or to learn more about how BlockApex can assist your organization in achieving its security objectives, please visit our Contact page at our website , or reach out to us via email at hello@blockapex.io.

Contents

1 Executive Summary

Our team performed a technique called Filtered Audit, where three individuals separately audited the PopFi Rust Program. A thorough and rigorous manual testing process involving line by line code review for bugs was carried out. All the raised flags were manually reviewed and re-tested to identify any false positives



• **Critical**
Critical vulnerabilities are too easy to exploit and can lead to damages/loss in assets or manipulations.

• **High**
High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution.

• **Medium**
Medium-level vulnerabilities are equally imperative to fix but they tend to have minimal impact on asset loss or data manipulations.

• **Low**
Lowest-level vulnerabilities, informational errors, violating code styles/ practices usually can't affect smart contract execution; hence they can be ignored.

• **Info**
Concerns that have no direct negative impact. These may help improve coding practices.

1.1 Scope

1.1.1 In Scope

Perpetual Futures PopFi introduces perpetual futures, a novel trading instrument that allows for indefinite holding periods, unlike traditional futures. This innovation enhances flexibility and market participation. Our audit will focus on the mechanisms enabling full trade autonomy, predefined contractual triggers, and the unique unified Solana vault system that consolidates liquidity, ensuring capital efficiency and decentralization. Key aspects include the platform's approach to synthetic leverage, collateralization with SOL, and profit and loss settlement based on real-time data from the Pyth Oracle.

Binary Options Our audit extends to PopFi's binary options, a simplified trading format where users speculate on asset price movements within short timeframes. The focus will be on the contract's design, particularly how time frames are managed, and the use of Pyth Oracle for price accuracy. We will examine the balance between user control and the platform's operational needs, ensuring that the predefined conditions for trade interactions are transparent and fair.

Liquidity Pools The liquidity pool mechanism on PopFi, structured around epochs, presents a unique model for liquidity provision. Our audit will assess the security and efficiency of deposit and withdrawal processes, the risk-sharing model between liquidity providers and the platform, and the transparency of profit and loss calculation and distribution. We'll also evaluate safeguards for liquidity providers, including the capping mechanisms for open interest and trade winning percentages, ensuring they align with the platform's commitment to provider safety.

Ambassador Program The Ambassador Program represents PopFi's initiative to foster deeper community engagement and platform growth through enhanced rewards for ambassadors. The audit will scrutinize the program's structure,the elevated commission rates, and the overall impact on the platform's ecosystem. We will ensure that the program's implementation aligns with PopFi's objectives of decentralization and community benefit, maintaining integrity and transparency NFTs and Revenue Sharing The introduction of NFTs by PopFi marks an expansion into combining digital art with financial incentives. Our audit will examine the economic implications of NFT ownership on the platform's fee distribution. Specifically, we will assess the revenue-earning potential for NFT holders and the impact of the new fee distribution model on liquidity providers and the project itself. The focus will be on ensuring that the NFT framework is secure, equitable, and enhances the PopFi ecosystem's value.

Program in Scope: `../programs/popfi/src/lib.rs`

Initial Commit Hash: `8cda93904c01c722badc80f916b858666b3476b5`

Final Commit Hash: `73152135c8264c84c0410fa3a3aed908566a5344`

1.1.2 Out of Scope

Server-Based Trade Resolution Mechanism: The audit does not include the examination or evaluation of the server-side mechanism used for resolving trades. This aspect falls outside the audit's scope. Any features or functionalities not explicitly mentioned in the "In Scope" section are also considered outside the scope of this audit.

1.2 Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review: Commits for the contracts were updated throughout the post-review. Some notable changes include the changes for fetching price from and oracle, complete shift towards checked arithmetic & removal of the options contract. The test-cases for all of the contracts were minimal with little to no coverage. It is highly recommended to thoroughly test every positive & negative test-cases and have a test-suite developed before the mainnet deployment.

1.3 Methodology

The codebase was audited using a filtered audit technique. A band of three (3) auditors scanned the codebase in an iterative process for a time spanning 2 Weeks. Starting with the recon phase, a basic understanding was developed, and the auditors worked on developing presumptions for the developed codebase and the relevant documentation/whitepaper. Furthermore, the audit moved on with the manual code reviews to find logical flaws in the codebase complemented with code optimizations,software, and security design patterns, code styles and best practices.

1.4 Status Descriptions

Acknowledged: The issue has been recognized and is under review. It indicates that the relevant team is aware of the problem and is actively considering the next steps or solutions.

Fixed: The issue has been addressed and resolved. Necessary actions or corrections have been implemented to eliminate the vulnerability or problem.

Closed: This status signifies that the issue has been thoroughly evaluated and acknowledged by the development team. While no immediate action is being taken.

1.5 Summary of Findings Identified

S.No	Severity	Findings	Status
#1	CRITICAL	Erroneous Withdrawal Logic Leading to Asset Mismanagement in SOL Withdrawals	FIXED
#2	HIGH	Lack of Enforcement Mechanism for Liquidity Pool Lock State	FIXED
#3	LOW	Inadequate Handling of Asset Price Staleness in Oracle Updates	FIXED
#4	INFO	Non-Descriptive Error Codes in Program Error Handling	FIXED

2 Findings and Risk Analysis

2.1 Erroneous Withdrawal Logic Leading to Asset Mismanagement in SOL Withdrawals

Severity: Critical

Status: Fixed

Location :

`../programs/popfi/src/lib.rs#L3751`

Description in the `withdraw_pol_from_lp` function of a given smart contract, there is a logical error that leads to incorrect manipulation of the internal accounting of deposited assets. Specifically, when withdrawing SOL (indicated by usdc == 0), the function mistakenly subtracts the withdrawal amount from `lp_acc.projects_deposited_usdc` (USDC deposits) instead of `lp_acc.projects_deposited_sol` (SOL deposits). This error could lead to inaccurate tracking of the assets within the liquidity pool, potentially causing financial discrepancies and impacting the integrity of the platform's asset management.

Proof of Concept The issue can be observed in the following code snippet:

```
if usdc == 0 {
msg!(&amp;quot;Withdrawing SOL&amp;quot;);
msg!(
&amp;quot;Actual Deposits: {} SOL&amp;quot;,
(lp_acc.projects_deposited_sol as f64 + lp_acc.total_deposits as f64)
/ 1_000_000_000.0
);
msg!(
&amp;quot;Withdrawing: {} SOL&amp;quot;,
withdraw_amount as f64 / 1_000_000_000.0
);
lp_acc.projects_deposited_usdc = lp_acc
.projects_deposited_usdc
.checked_sub(withdraw_amount)
.ok_or(ProgramError::Custom(0xABCDEF))?;
```

Here, despite intending to process a SOL withdrawal (usdc == 0), the contract adjusts the USDC deposited balance (projects_deposited_usdc), which does not reflect the user's action accurately.

Recommendation correct the withdrawal logic to ensure that the correct asset type's deposited balance is adjusted during a withdrawal. This involves using `lp_acc.projects_deposited_sol` for adjustments when usdc == 0.

2.2 Lack of Enforcement Mechanism for Liquidity Pool Lock State

Severity: High

Status: Fixed

Location :

`../programs/popfi/src/lib.rs`

Description The PopFi platform's documentation outlines a mechanism where liquidity pools are "locked" for deposits and withdrawals during weekdays and "unlocked" during weekends, as part of its epoch structure. This mechanism is crucial for maintaining the stability and security of the liquidity pool, preventing malicious activities or exploitation through unexpected deposits and withdrawals. Upon review, it's observed that while there is a provision (`lp_acc.locked`) to indicate the pool's lock state, there is no implemented functionality to toggle this state between locked and unlocked. Specifically, the liquidity pool account initialization sets `lp_acc.locked` to false, with no subsequent code or function found in the provided snippets to change this state based on the epoch system's requirements.

Impact Without the ability to lock the liquidity pool according to the epoch schedule:

1. The platform cannot enforce the documented liquidity management strategy, potentially leading to unplanned liquidity fluctuations.
2. Liquidity providers might be able to interact with the pool outside of the intended unlock periods, contradicting platform policies and expectations.
3. This misalignment between documentation and implementation may undermine user trust and the platform's operational integrity.

Proof of Concept :

1. In the `initialize_lp_acc` function, lp_acc.locked is set to false, indicating an unlocked state.
2. The `stake_and_mint_tokens` and `withdraw_from_liquidity_pool` functions correctly check `lp_acc.locked` to decide if operations should proceed.
3. Absence of a function or mechanism to set lp_acc.locked to true as per the documented epoch structure, preventing the platform from programmatically locking the pool during weekdays.

Recommendation Implement a function that toggles the `lp_acc.locked` state between true (locked) and false (unlocked), ideally linked to the platform's epoch system.

2.3 Inadequate Handling of Asset Price Staleness in Oracle Updates

Severity: Low

Status: Fixed

Location :

`../programs/popfi/src/lib.rs#L2151`

Description To update the prices of assets , `update_price` function is called by the backend/server. To cater the scenario of stale prices , price of sol is fetched and based on that the prices of all other assets is updated. Making staleness of other assets based on SOL is not a good approch as prices of each assets is based on different oracle and it can happen that the price of BTC is stale but of SOL it is not

Recommendation It is recommended to not make price of SOL as source of staleness for all assets rather each asset's oracle to be considered individually.

2.4 Non-Descriptive Error Codes in Program Error Handling

Severity: Info

Status: Fixed

Location :

```
../programs/popfi/src/lib.rs
```

Description There are instances in the program code where error handling utilizes a generic placeholder `Err(ProgramError::Custom(0xABCDEF4))`. For improved clarity and error management, it's recommended to employ more descriptive standard error types.

Proof of Concept :

Several code segments utilize a non-descriptive custom error pattern:

```
return Err(ProgramError::Custom(0xABCDEF4));
```

Recommendation It's advisable to replace generic custom error codes with predefined error types that accurately describe the nature of the error, such as `ProgramError::InvalidArgument`. This adjustment will enhance error specificity, making it easier for developers and auditors to understand the context and cause of errors, thus facilitating more effective debugging and contract maintenance