BlockApex

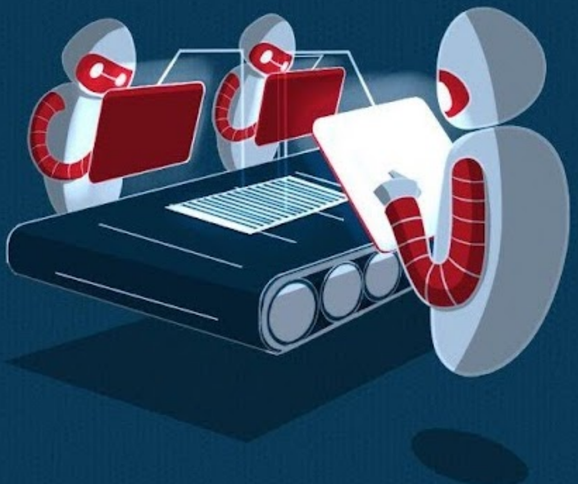
SMART CONTRACT SECURITY ANALYSIS REPORT

```
pragma solidity 0.7.0;
contract Contract {

    function hello() public returns (string) {
        return "Hello World!";
    }

    function findVulnerability() public returns (string) {
        return "Finding Vulnerability";
    }

    function solveVulnerability() public returns (string) {
        return "Solve Vulnerability";
    }
}
```



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.

1.1 Summary of Findings Identified

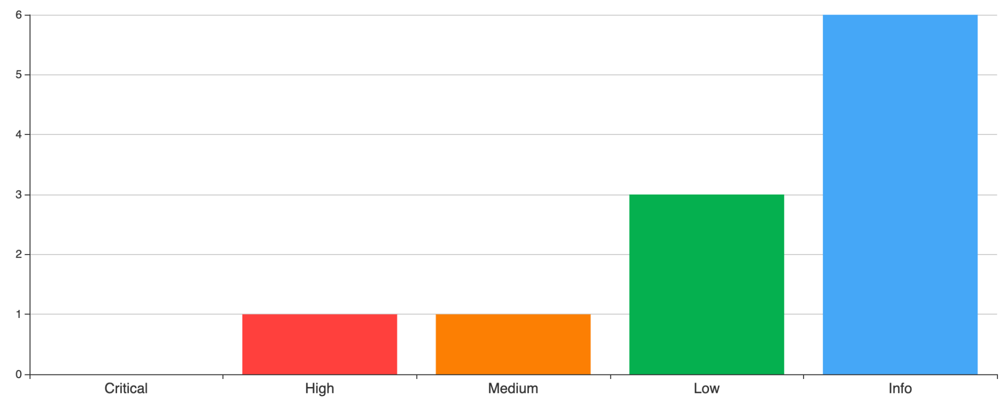


Figure 1: Executive Summary

1.2 Scope

1.2.1 In Scope

Perpetual Futures: Involves contracts that do not have an expiry date, allowing traders to hold positions indefinitely.

Binary Options: A type of options contract where the payout is either a fixed amount or nothing, depending on the outcome of a yes/no proposition related to the price of an underlying asset.

Liquidity Pools: These are pools of tokens locked in a smart contract that provide liquidity for the platform, facilitating trading and other financial activities.

Ambassador Program: A program aimed at promoting the platform and incentivizing user participation and engagement.

1.2.2 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.

1.3 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.

Status Description

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 High Temporary funds lock due to Inefficient collateral check

# 2 Medium Potential DoS Due to Low Liquidity Limitations

# 3 Low Unintended Increase of 200x Leverage Due to Rounding Error

# 4 Low Unambigous liquidity unlock mechanism

# 5 Low Excessive Use of unwrap() in Error Handling

# 6 Info Insufficient Checks for Arithmetic Operations

# 7 Info Risk of Stale Prices

# 8 Info Missing Slippage Check

# 9 Info Misleading output messages

# 10 Info Implementation of Multisig

# 11 Info Unused Variable

2 Findings and Risk Analysis

2.1 Temporary funds lock due to Inefficient collateral check

Severity: High

Status: Partially Fixed

Location programs/popfi/src/lib.rs

Description On weekends, liquidity providers have the option to deposit and withdraw liquidity. Trading is also enabled during this period. It is necessary to implement a check within the **withdraw_from_liquidity_pool** function to ensure that a minimum amount of liquidity is maintained to sustain existing trades. This precaution aims to prevent the complete withdrawal of the entire liquidity pool, which could disrupt active trades. When calculating payouts for a trade, if the payout is greater than zero, the transfer of the payout is made from the **pda_house_acc** to the **player_acc**. If in the future the liquidity amount increases beyond 200 SOL, and liquidity providers other than the protocol contribute liquidity, there may be a potential scenario where funds are temporarily locked.

Recommendation This is an edge case, regarding collateral management we have following recommendations:

1. For the liquidity Providers there should be a minimum lockup period time i.e 1 Month/2 weeks so that this type of scenario doesn't happen.
2. For the trades being executed on Weekends there should be enough underlaying colletral to sustain those trades.

Auditor Response

The fix applied on lockup time has inherent limitation that if a user misses the epoch to withdraw then he has to wait 2 epochs more to withdraw , this is acknowledged by the developer and its the intended design. Still there is no collateral check applied on withdrawal to cater the needs of opened trades. It should be considered too.Collateral management itself is a complex financial task. What we suggest that should be enough under laying collateral to sustain trades. Regarding how much ,it should be vetted.

Dev Response

Upon discussion with developer on discord, Dev proposes to put "(1 - 2 * total collaterals / total deposits in lp) * user deposits" on withdrawal.

2.2 Potential DoS Due to Low Liquidity Limitations

Severity: Medium

Status: Acknowledged

Location programs/popfi/src/lib.rs#L1633,2633

Description In the current implementation of PopFi a maximum cap of 200 SOL is applied which can be deposited into the liquiditypool.When making a future call,its checked that the total collateral in the "ratio_acc" should be less that the "lp_acc.total_deposit / 4". This implies that if "ratio.acc.total_collateral" increases more than 50 SOL then no more trades can be placed.

```
    if ratio_acc.total_collateral &gt; (lp_acc.total_deposits / 4) {
                return Err(ProgramError::InvalidArgument);
            }
```

Impact An attacker can carryout DDOS against the protocol by using 1x Leverage and putting multiple orders each having a size of 1 SOL. The cost to the attack will be 51 SOL considering the current amount of liquidity in the Pool.

Recommendation It is recommended to increase the liquidity in the pool. As the liquidity will increase so will the cost of attack. The check against collateral is good for protocol too but it can be abused as indicated above.

Dev Response

We are planning to Increase liquidity, 200 Is just for small scale at the moment

2.3 Unintended Increase of 200x Leverage Due to Rounding Error

Severity: Low

Status: Fixed

Location programs/popfi/src/lib.rs#L42-68

Description Popfi allows a maximum leverage of up to 200x. While making a trade, a 'get_dynamic_leverage' function is called to determine the maximum leverage a user can utilize. Under the condition where the long_short_ratio is 0.6 for long positions and 0.4 for short positions, the maximum leverage returned is inadvertently 201x due to rounding errors. As a result, a user can actually take leverage of 201x.

```
    fn get_dynamic_leverage(long_short_ratio: u64, price_direction: u8) -&gt; u64 {
        let long_short_f64 = long_short_ratio as f64;
        let long_short_ratio_f64 = long_short_f64 / SCALE as f64; // Assuming the ratio was
            provided as 0-10 (integers), converting it to 0.0-1.0 (float)
        assert!(
            long_short_ratio_f64 &gt;= 0.0 &amp;&amp; long_short_ratio_f64 &lt;= 1.0,
            &quot;Ratio must be between 0 and 1&quot;
        );
        let leverage: f64;
        match price_direction {
            0 =&gt; {
                // 0 for Increase (long)
                leverage = MAX_LEVERAGE * (-K * (long_short_ratio_f64 - G_17).powi(2)).exp();}
            1 =&gt; {
                // 1 for Decrease (short)
                leverage = MAX_LEVERAGE * (-K * (G_15 - long_short_ratio_f64).powi(2)).exp();}
            _ =&gt; {
                panic!(&quot;Invalid price direction&quot;);}
        }
        return SCALE * (leverage + 1.0) as u64; // Rounding to nearest whole number and then
            converting to u64
    }
```

Proof of Concept We simply called the `max_dynamic_leverage` with the values: 60000,0 => Long 40000,1 => Short

Recommendation It is recommended to return 200 if the value becomes more than 200. Since its the maximum leverage which is allowed.

2.4 Unambigous liquidity unlock mechanism

Severity: Low

Status: Fixed

Location programs/popfi/src/lib.rs#L1118

Description On weekends liquidity pools are unlocked and users can deposit and withdraw liquidity/fees. Currently "update_lp_acc_fee" sets the "lp_acc.locked = false". When update_lp_acc_fee is called, pools are opened and after weekend "lock_lp_acc" is called to lock the pools. For clarity and its advised to put the unlock functionality with the "lock_lp_acc" similar to "halt_lp_acc". Its a matter of design choice.

```
    lp_acc.locked = false;
```

Recommendation It is recommended to put the same mechanism as **halt_lp_acc** on the **lock_lp_acc** too.

```
    if lp_acc.is_halted {
        lp_acc.is_halted = false;
    } else if !lp_acc.is_halted {
        lp_acc.is_halted = true;
    }
```

2.5 Excessive Use of unwrap() in Error Handling

Severity: Low

Status: Fixed

Location programs/popfi/src/lib.rs Total 80 Such instance were found where unwrap() is being used.

Description The Solana program extensively utilizes unwrap() for handling Option and Result types. This method, while simplifying code, leads to abrupt program termination if it encounters None or Err values, without proper error handling.

Impact Using unwrap() can cause unexpected program panics and transaction failures, especially problematic in a blockchain context. This not only affects the reliability of the program but also risks financial implications for users due to failed transactions.

Recommendation Refactor Error Handling: Gradually replace unwrap() with explicit error handling patterns using match or if let. This ensures more controlled and reliable program behavior. Intermediate Steps: Where immediate comprehensive refactoring is not feasible, use expect() as a temporary measure to provide more context during panics.

2.6 Insufficient Checks for Arithmetic Operations

Severity: Info

Status: Fixed

Description Popfi incorporates multiple complex mathematical calculations within its smart contract operations. These calculations are crucial to the platform's functionality and require precision and accuracy. In a language like Rust, arithmetic operations carry the risk of overflows and underflows scenarios where calculations exceed the maximum or minimum limits of the data type being used, leading to potentially catastrophic errors.

Recommendation To mitigate these risks, it's recommended to employ 'checked arithmetic' in Rust. Checked arithmetic operations are designed to safely handle scenarios that could otherwise result in overflows or underflows. By using these operations, any arithmetic that goes beyond the bounds of what the data types can handle will result in a predictable, manageable error rather than an unpredictable and potentially harmful behavior.

2.7 Risk of Stale Prices

Severity: Info

Status: Acknowledged

Description The Pyth oracle operates by collecting price data from various reliable sources, which transmit this data to the blockchain at regular intervals. Despite its design focusing on high availability and reliability, there are instances where network delays, compromised data providers, or other issues can slow down the aggregation process. This delay can lead to the oracle displaying outdated or 'stale' prices.

When prices become stale, substantial arbitrage opportunities often emerge, enticing traders to capitalize on these disparities. Furthermore, when the oracle resumes normal operation and updates its price feeds, it can create additional significant arbitrage possibilities.

Extended periods of stale pricing or sporadic oracle functionality can result in a series of large-scale arbitrages. These can adversely impact liquidity providers, potentially leading to financial repercussions within the trading ecosystem.

Recommendation To mitigate the risk of stale prices from a single oracle, it is advisable to adopt a multi-oracle strategy. Sole reliance on one price source can result in problems. Chainlink, which also offers an oracle service for Solana, presents a viable alternative. Incorporating Chainlink alongside Pyth within the protocol is recommended to ensure more accurate pricing and to prevent issues associated with stale price data from a single oracle source.

2.8 Missing Slippage Check

Severity: Info

Status: Fixed

Location programs/popfi/src/lib.rs#L1742,2744

Description While submitting an order slippage can be provided by user. There is no check if 0 is provided as slippage price. It will cause a division error while calculating the difference.

```
    let difference = SCALE as i64 * (initial_price - slippage_price) / slippage_price
```

```
    let difference = SCALE as i64 * (pyth_price - slippage_price) / slippage_price;
```

Recommendation It is recommended to put check for the slippage price , it should be > 0.

2.9 Misleading output messages

Severity: Info

Status: Fixed

Location programs/popfi/src/lib.rs#L1816,1827,1834

Description In the popfi Smart Contract,multiple output messages are being logged. These can be for debugging process or for logging purpose. Some messages are missleading and should be corrected so accurate discription and messages are logged.

```
    msg!(&quot;bet_amount: {}&quot;, fees_payout);
```

Recommendation Correct description and message should be logged to avoid confusion.

2.10 Implementation of Multisig

Severity: Info

Status: Acknowledged

Description Futures and options trading is a complex financial activity which requires robust security measures. Currently, transactions and operational controls are managed through single-signature wallets, which, while efficient, pose significant security risks. It is advised to use MultiSig wallet which will ensure enhanced security.

2.11 Unused Variable

Severity: Info

Status: Acknowledged

Location programs/popfi/src/lib.rs

Description Two variables "_number: u64" were indentified which are not being used as per our observation.

```
    impl&lt;&#x27;info&gt; BinaryOption {
        pub fn create_bin_opt(
            ctx: Context&lt;CreateBinaryOption&gt;,
            **_number: u64,**
            affiliate_code: [u8; 8],
            bet_amount: u64,
            expiration: u64,
            price_direction: u8,
            symbol: u8,
            slippage_price: i64,
            slippage: i64,
        )
```

```
    impl&lt;&#x27;info&gt; FuturesContract {
        pub fn create_fut_cont(
            ctx: Context&lt;CreateFuturesContract&gt;,
            **_number: u64,**
            affiliate_code: [u8; 8],
            bet_amount: u64,
            leverage: u64,
            price_direction: u8,
            symbol: u8,
            sl_price: i64,
            tp_price: i64,
            slippage_price: i64,
            slippage: i64,
        )
```

Impact There is no such impact of the issue but for code clearity unused variable should be removed.

Disclaimer:

The smart contracts provided by the client with the purpose of security review have been thoroughly analyzed in compliance with the industrial best practices till date w.r.t. Smart Contract Weakness Classification (SWC) and Cybersecurity Vulnerabilities in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token (if any), its sale, or any other aspect of the project that contributes to the protocol's public marketing.

Crypto assets/ tokens are the results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the reports in any way, including to make any decisions to buy or sell any token, product, service, or asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the programmable code and only the programmable code, we note, as being within the scope of our review within this report. The smart contract programming language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond the programming language's compiler scope that could present security risks.

This security review cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While BlockApex has done their best in conducting the analysis and producing this report, it is important to note that one should not rely on this report only - we recommend proceeding with several independent code security reviews and a public bug bounty program to ensure the security of smart contracts.