// Private Solana Program Security Assessment

02.06.2024 - 03.20.2024

Perpetuals

Flash Trade

HALBORN

Perpetuals - Flash Trade

Prepared by: HALBORN

Last Updated 05/13/2024

Date of Engagement by: February 6th, 2024 - March 20th, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
9	1	1	0	4	3

TABLE OF CONTENTS

1. Introduction

Flash Trade is a decentralized spot and perpetuals exchange on Solana that lets to trade on up to 100x leverage, with low fees and minimal price impact. The trading engine of Flash is powered by a unique multi-asset pool-to-peer oracle-based program, the first of its kind on Solana. Additionally, it uses an evolutionary NFT architecture to abstract accounts, unlocking a new level of gamification to incentivize high-volume trading on the protocol.

Halborn conducted a security assessment on their Solana programs, beginning on February 6th, 2024 and ending on March 20th, 2024. The security assessment was scoped to the programs provided in the [flash-contracts-closed](https://github.com/flash-trade/flash-contracts-closed/tree/main) GitHub repository. Commit hashes and further details can be found in the **Scope** section of this report.

2. Assessment Summary

The team at Halborn was provided 6.5 weeks for the engagement and assigned 1 full-time security engineer to review the security of the programs in scope. The security engineer is a blockchain and Solana Program security expert with advanced penetration testing and Solana Program hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to identify potential security issues within the programs.

In summary, Halborn identified some improvements to reduce the likelihood and impact of multiple risks, which has been partially addressed by the Flash Trade team. The main ones were the following:

- Protocol Fees inconsistency in WithdrawFees
- Failure to remove Markets despite empty position values
- Risk of lost unclaimed rewards due to if Flp Stake misconfiguration
- Check missing during reward vault initialization

3. Test Approach And Methodology

Halborn performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the platform.

- Manual program source code review to identify business logic issues.

- Mapping out possible attack vectors

- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.

- Scanning dependencies for known vulnerabilities (`cargo audit`).

- Local runtime testing (`solana-test-framework`)

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (m_e)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILIY METRIC (m_e)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (m_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (m_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: flash-contracts-closed

(b) Assessed Commit ID: 7dcf940

(c) Items in scope:

- 1. perpetuals (`programs/perpetuals/*`)
- 2. fbnft-rewards (`programs/fbnft-rewards/*`)
- 3. perp-composability (`programs/perp-composability/*`)

Out-of-Scope: - third-party libraries and dependencies, - financial-related attacks

REMEDIATION COMMIT ID:

- 06a5eb406a5eb4
- 1e44c351e44c35
- 60b325860b3258

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	1	0	4	3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-06 - PROTOCOL FEES INCONSISTNECY IN WITHDRAWFEES	Critical	SOLVED - 03/08/2024
HAL-10 - FAILURE TO REMOVE MARKETS DESPITE EMPTY POSITION VALUES	High	SOLVED - 03/08/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-05 - MULTIPLE RISKS ASSOCIATED WITH INVALID ORACLE CONFIGURATION	Low	PARTIALLY SOLVED - 04/22/2024
HAL-03 - CHECK MISSING DURING REWARD VAULT INITIALIZATION	Low	SOLVED - 04/22/2024
HAL-07 - RISK OF LOSS OF USERS BENEFITS AND REWARDS DUE TO PERPETUALS AND POOL MISCONFIGURATION	Low	PARTIALLY SOLVED - 04/22/2024
HAL-04 - RISK OF LOST UNCLAIMED REWARDS DUE TO IF FLP STAKE MISCONFIGURATION	Low	SOLVED - 04/28/2024
HAL-09 - MISSING VALIDATION FOR ORACLE AUTHORITY ADDRESS	Informational	ACKNOWLEDGED
HAL-02 - REDUNDANT PERMISSIONS	Informational	ACKNOWLEDGED
HAL-08 - LACK OF ERROR HANDLING IF DIFFERENT COLLATERAL CUSTODY IN PERP COMPOSABILITY	Informational	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 (HAL-06) PROTOCOL FEES INCONSISTNECY IN WITHDRAWFEES

// CRITICAL

Description

The WithdrawFees instruction empowers administrators to transfer custody protocol fees, which are dynamically updated with each call to CollectStakeReward, to a designated receiving account.
programs/perpetuals/src/instructions/collect_stake_reward.rs

```
    let user_reward = math::checked_as_u64(math::checked_div(
        math::checked_mul(flp_stake.unclaimed_rewards as u128,
            flp_stake.fee_share_bps as u128
        )?,
        Perpetuals::BPS_POWER
    )?)?;

    // transfer tokens to user
    msg!("Transfer flp tokens");
    perpetuals.transfer_tokens(
        ctx.accounts.fee_custody_token_account.to_account_info(),
        ctx.accounts.receiving_token_account.to_account_info(),
        ctx.accounts.transfer_authority.to_account_info(),
        ctx.accounts.token_program.to_account_info(),
        user_reward,
    )?;

    fee_custody.fees_stats.paid = math::checked_add(fee_custody.fees_stats.paid,
user_reward as u128)?;
    fee_custody.fees_stats.protocol_fee = math::checked_add(
        fee_custody.fees_stats.protocol_fee,
        math::checked_sub(flp_stake.unclaimed_rewards, user_reward)?
    )?;
```

Consequently, the custody fees paid (**custody.fees_stats.paid**) are adjusted based on the withdrawn amount.
However, a critical oversight This perpetuates a scenario where protocol fees consistently accumulate with each execution of CollectStakeRewards, resulting in subsequent calls to WithdrawFees transferring larger amounts than warranted. This situation jeopardizes custody funds, risking partial depletion with each instruction call and leaving the system in an inconsistent state.persists: the protocol fees themselves remain static.
programs/perpetuals/src/instructions/withdraw_fees.rs

```
   let custody = ctx.accounts.custody.as_mut();
    let fee_amount = custody.fees_stats.protocol_fee;
    msg!("Withdraw token fees: {}", fee_amount);

    ctx.accounts.perpetuals.transfer_tokens(
        ctx.accounts.custody_token_account.to_account_info(),
        ctx.accounts.receiving_token_account.to_account_info(),
        ctx.accounts.transfer_authority.to_account_info(),
        ctx.accounts.token_program.to_account_info(),
        fee_amount,
    )?;

    custody.fees_stats.paid = math::checked_add(custody.fees_stats.paid, fee_amount
as u128)?;

    Ok(0)
```

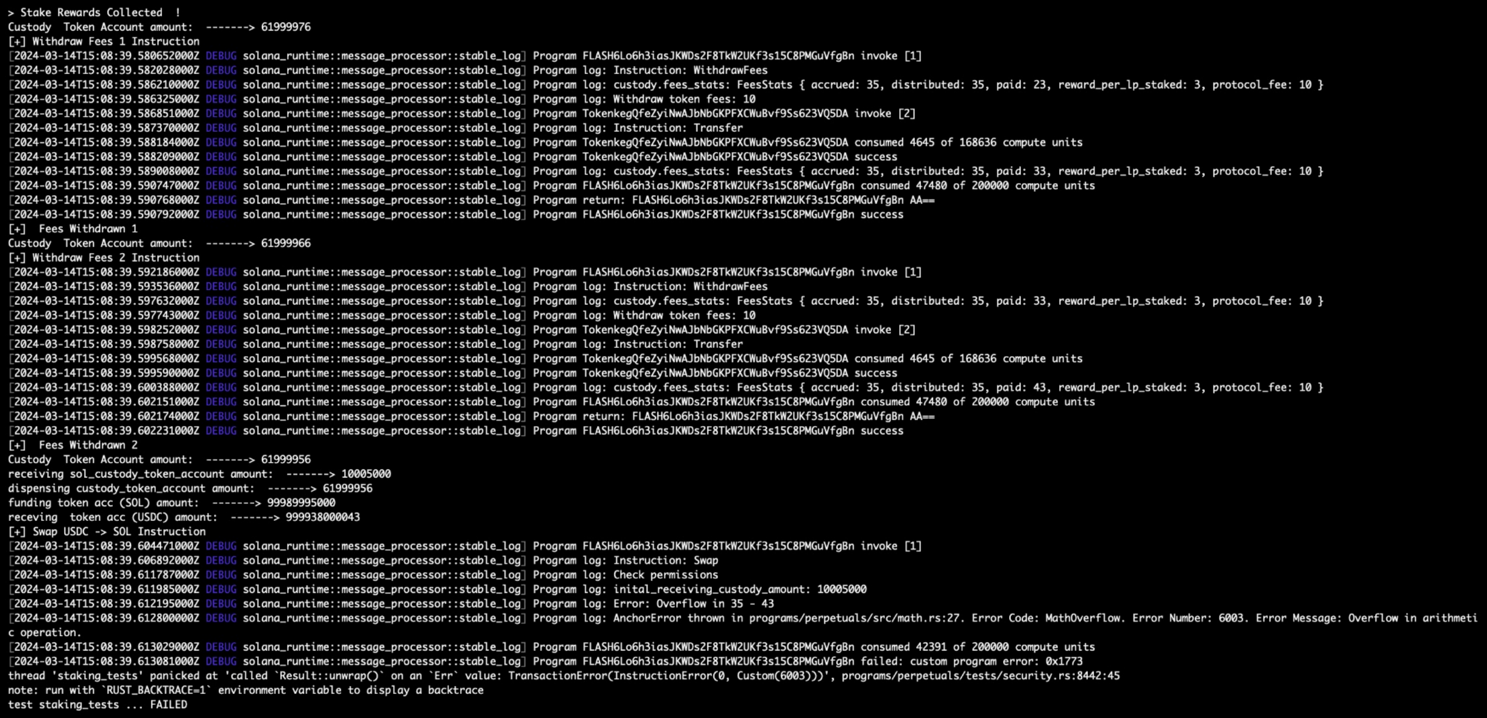
This discrepancy not only compromises the accuracy of protocol fee withdrawal transactions but also affects other transaction calculations. As protocol fees inflate without adjustment, the corresponding paid fees are updated accordingly. This leads to potential overflow in the calculation of the initial custody amount in other instructions, as the fees paid may exceed those accrued on the custody from which the protocol fees are derived.

```
   let initial_custody_amount = math::checked_add(
        math::checked_add(collateral_custody.assets.owned,
collateral_custody.assets.collateral)? as u128,
        math::checked_sub(collateral_custody.fees_stats.accrued,
collateral_custody.fees_stats.paid)?
    )?;
```

Proof of Concept

1. Init Perpetual
2. Add a collection
3. Add a pool
4. Set permissions
5. Set Pool Config (setting
6. Add diferent Custodies (USDC, SOL, ETH, BTC)
7. Set Custodies (USDC, SOL, ETH, BTC)
8. Bob creates trading Account
9. Bob creates Referral Account
10. Bob updates its Trading Account
11. Alice creates Referral Account
12. Add Market SOL -> SOL (LONG)
13. Add Market SOL -> USDC (SHORT)
14. Init Staking setting Custody USDC as pool.reward_custody

15. Refresh 1
16. Bob adds Liquidity to different Custodies
17. Bob Deposits Stake
18. Set the different Custom Oracles Price and Refresh Stake day 2
19. Alice and Bob Open Positions Market SOL -> USDC (SHORT)
20. Refresh day 3
21. Bob Unstake instant
22. Bob Collect Stake Rewards (so the protocol fees are updated)
23. Admin Withdraw Fees once
24. Admin Withdraw Fees twice
25. Atemp to swap



```
> Stake Rewards Collected  !
Custody  Token Account amount:  -------> 61999976
[+] Withdraw Fees 1 Instruction
[2024-03-14T15:08:39.580652000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn invoke [1]
[2024-03-14T15:08:39.582028000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: WithdrawFees
[2024-03-14T15:08:39.586210000Z DEBUG solana_runtime::message_processor::stable_log] Program log: custody.fees_stats: FeesStats { accrued: 35, distributed: 35, paid: 23, reward_per_lp_staked: 3, protocol_fee: 10 }
[2024-03-14T15:08:39.586325000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Withdraw token fees: 10
[2024-03-14T15:08:39.586851000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2024-03-14T15:08:39.587370000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2024-03-14T15:08:39.588184000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 168636 compute units
[2024-03-14T15:08:39.588209000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2024-03-14T15:08:39.589008000Z DEBUG solana_runtime::message_processor::stable_log] Program log: custody.fees_stats: FeesStats { accrued: 35, distributed: 35, paid: 33, reward_per_lp_staked: 3, protocol_fee: 10 }
[2024-03-14T15:08:39.590747000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn consumed 47480 of 200000 compute units
[2024-03-14T15:08:39.590768000Z DEBUG solana_runtime::message_processor::stable_log] Program return: FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn AA==
[2024-03-14T15:08:39.590792000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn success
[+]  Fees Withdrawn 1
Custody  Token Account amount:  -------> 61999966
[+] Withdraw Fees 2 Instruction
[2024-03-14T15:08:39.592186000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn invoke [1]
[2024-03-14T15:08:39.593536000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: WithdrawFees
[2024-03-14T15:08:39.597632000Z DEBUG solana_runtime::message_processor::stable_log] Program log: custody.fees_stats: FeesStats { accrued: 35, distributed: 35, paid: 33, reward_per_lp_staked: 3, protocol_fee: 10 }
[2024-03-14T15:08:39.597743000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Withdraw token fees: 10
[2024-03-14T15:08:39.598252000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2024-03-14T15:08:39.598758000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2024-03-14T15:08:39.599568000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 168636 compute units
[2024-03-14T15:08:39.599590000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2024-03-14T15:08:39.600388000Z DEBUG solana_runtime::message_processor::stable_log] Program log: custody.fees_stats: FeesStats { accrued: 35, distributed: 35, paid: 43, reward_per_lp_staked: 3, protocol_fee: 10 }
[2024-03-14T15:08:39.602151000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn consumed 47480 of 200000 compute units
[2024-03-14T15:08:39.602174000Z DEBUG solana_runtime::message_processor::stable_log] Program return: FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn AA==
[2024-03-14T15:08:39.602231000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn success
[+]  Fees Withdrawn 2
Custody  Token Account amount:  -------> 61999956
receiving sol_custody_token_account amount:  -------> 10005000
dispensing custody_token_account amount:  -------> 61999956
funding token acc (SOL) amount:  -------> 99989995000
receving  token acc (USDC) amount:  -------> 999938000043
[+] Swap USDC -> SOL Instruction
[2024-03-14T15:08:39.604471000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn invoke [1]
[2024-03-14T15:08:39.606892000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Swap
[2024-03-14T15:08:39.611787000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Check permissions
[2024-03-14T15:08:39.611985000Z DEBUG solana_runtime::message_processor::stable_log] Program log: inital_receiving_custody_amount: 10005000
[2024-03-14T15:08:39.612195000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Error: Overflow in 35 - 43
[2024-03-14T15:08:39.612800000Z DEBUG solana_runtime::message_processor::stable_log] Program log: AnchorError thrown in programs/perpetuals/src/math.rs:27. Error Code: MathOverflow. Error Number: 6003. Error Message: Overflow in arithmeti
c operation.
[2024-03-14T15:08:39.613029000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn consumed 42391 of 200000 compute units
[2024-03-14T15:08:39.613081000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn failed: custom program error: 0x1773
thread 'staking_tests' panicked at 'called `Result::unwrap()` on an `Err` value: TransactionError(InstructionError(0, Custom(6003)))', programs/perpetuals/tests/security.rs:8442:45
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
test staking_tests ... FAILED
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:H/D:H/Y:M/R:N/S:U (10.0)

Recommendation

To address the observed inconsistency, it's pivotal to implement a mechanism within the withdraw_fees handler instruction to dynamically update the protocol fees with each call. Specifically, resetting the protocol fees to zero once they've been withdrawn ensures accuracy and consistency.

Remediation Plan

SOLVED: To rectify the identified inconsistency, the **Flash Trade team** has been made a modification to the **withdraw_fees** handler instruction. Now, upon fee withdrawal, the custody's fee state protocol fee is dynamically reset to zero. This adjustment ensures the ongoing accuracy and consistency of the protocol's fee management.

Remediation Hash

https://github.com/flash-trade/flash-contracts-closed/pull/42/commits/06a5eb4ba74bbc617ddc3cc8423a639c57f6b04a

7.2 (HAL-10) FAILURE TO REMOVE MARKETS DESPITE EMPTY POSITION VALUES

// HIGH

Description

The RemoveMarket instruction is restricted solely to administrator signatories equipped with multisig capabilities. Its purpose is to eliminate a market when its collective position values become empty. Despite this initial validation, the instruction handler fails to proceed with the necessary actions. As a result, the market remains intact, leaving the pool's market vector unaltered. This oversight permits trading activities to continue even after the market's supposed removal, thereby allowing actions like OpenPosition to proceed. Moreover, this loophole could potentially generate commissions from a market that should have been eradicated.

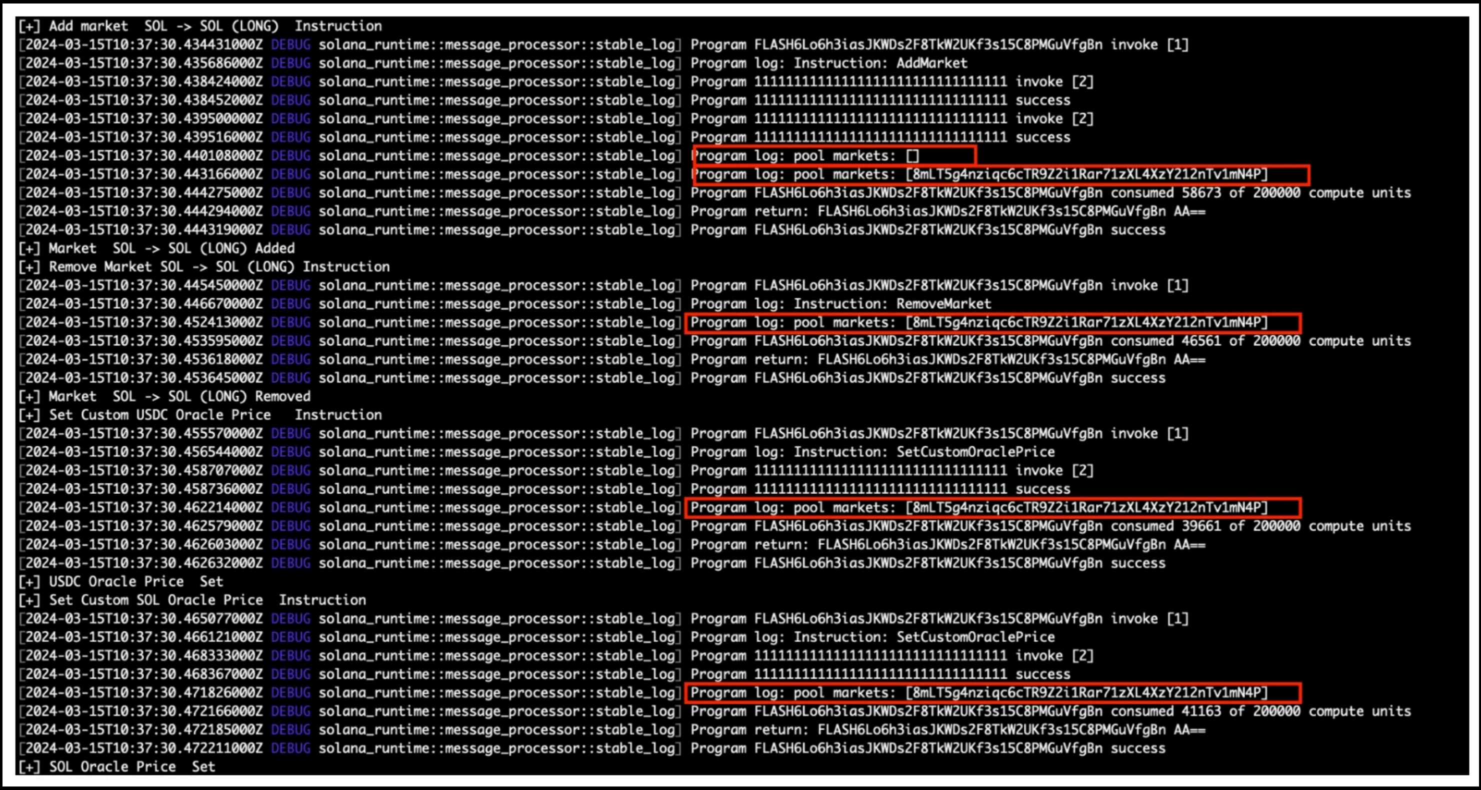
programs/perpetuals/src/instructions/remove_market.rs

```
pub fn remove_market<'info>(
    ctx: Context<'_, '_, '_, 'info, RemoveMarket<'info>>,
    params: &RemoveMarketParams,
) -> Result<u8> {

    // validate signatures
    let mut multisig = ctx.accounts.multisig.load_mut()?;

    let signatures_left = multisig.sign_multisig(
        &ctx.accounts.admin,
        &Multisig::get_account_infos(&ctx)[1..],
        &Multisig::get_instruction_data(AdminInstruction::RemoveMarket, params)?,
    )?;
    if signatures_left > 0 {
        msg!(
            "Instruction has been signed but more signatures are required: {}",
            signatures_left
        );
        return Ok(signatures_left);
    }

    let market = ctx.accounts.market.as_mut();
    require!(
        market.collective_position.collateral_amount == 0
            && market.collective_position.size_amount == 0
            && market.collective_position.locked_amount == 0
            && market.collective_position.open_positions == 0,
        PerpetualsError::InvalidMarketState
    );
```

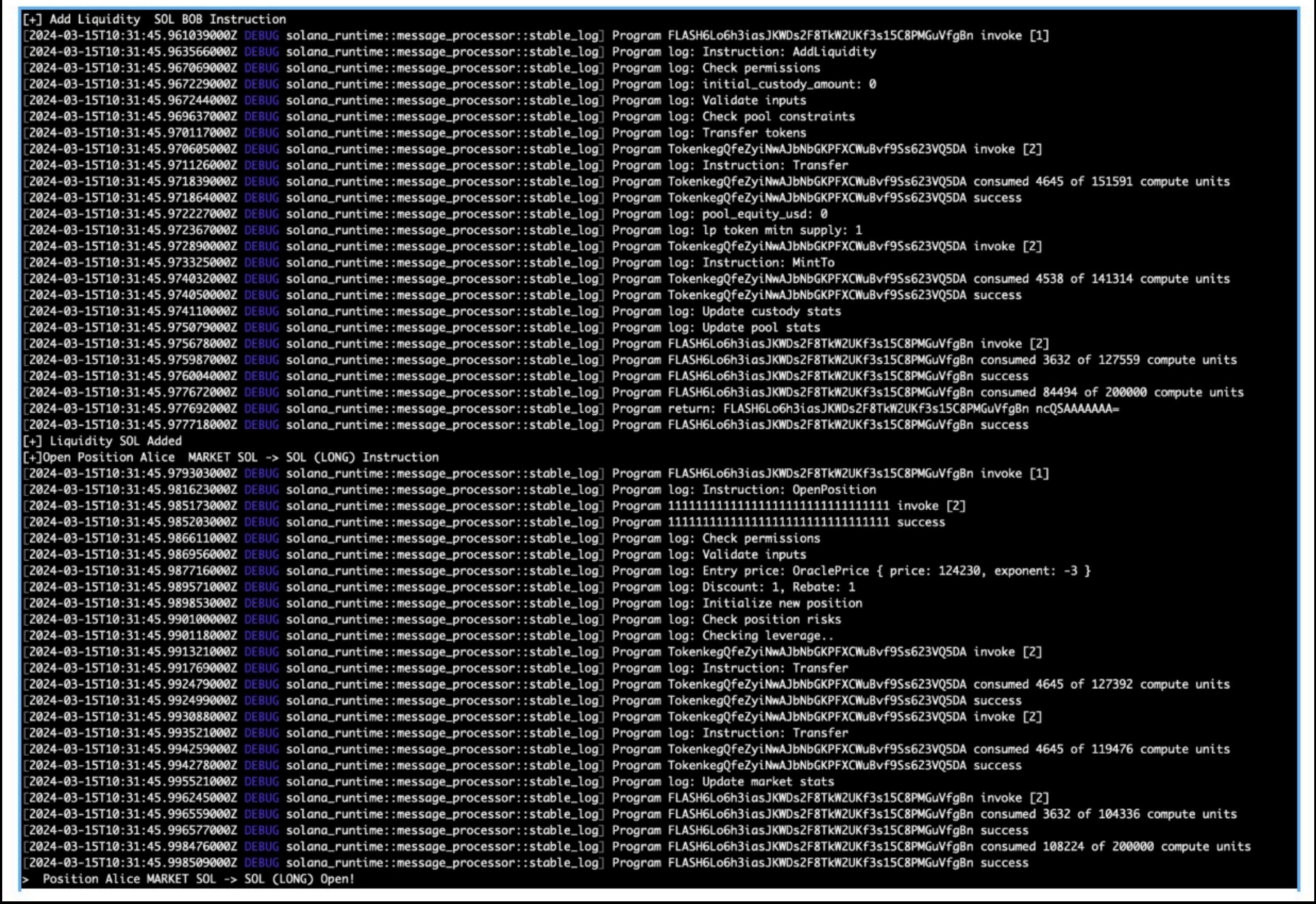
```
Ok(0)
```

Proof of Concept

1. Init Perpetual
2. Add a collection
3. Add a pool
4. Add diferent Custodies (USDC, SOL)
5. Bob creates trading Account
6. Bob creates Referral Account
7. Bob updates its Trading Account
8. Alice creates Referral Account
9. Add Market SOL -> SOL (LONG)
10. Remove Market SOL -> SOL (LONG)
11. Set Custom Oracles Pirce
12. Bob adds Liquidity (SOL)
13. Alice and Bob Open Positions Market SOL -> SOL (LONG)
14.



```
[+] Add market  SOL -> SOL (LONG)  Instruction
[2024-03-15T10:37:30.434431000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn invoke [1]
[2024-03-15T10:37:30.435686000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: AddMarket
[2024-03-15T10:37:30.438424000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 invoke [2]
[2024-03-15T10:37:30.438452000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 success
[2024-03-15T10:37:30.439500000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 invoke [2]
[2024-03-15T10:37:30.439516000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 success
[2024-03-15T10:37:30.440108000Z DEBUG solana_runtime::message_processor::stable_log] Program log: pool markets: []
[2024-03-15T10:37:30.443166000Z DEBUG solana_runtime::message_processor::stable_log] Program log: pool markets: [8mLT5g4nziqc6cTR9Z2i1Rar71zXL4XzY212nTv1mN4P]
[2024-03-15T10:37:30.444275000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn consumed 58673 of 200000 compute units
[2024-03-15T10:37:30.444294000Z DEBUG solana_runtime::message_processor::stable_log] Program return: FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn AA==
[2024-03-15T10:37:30.444319000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn success
[+] Market  SOL -> SOL (LONG) Added
[+] Remove Market SOL -> SOL (LONG) Instruction
[2024-03-15T10:37:30.445450000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn invoke [1]
[2024-03-15T10:37:30.446670000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: RemoveMarket
[2024-03-15T10:37:30.452413000Z DEBUG solana_runtime::message_processor::stable_log] Program log: pool markets: [8mLT5g4nziqc6cTR9Z2i1Rar71zXL4XzY212nTv1mN4P]
[2024-03-15T10:37:30.453595000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn consumed 46561 of 200000 compute units
[2024-03-15T10:37:30.453618000Z DEBUG solana_runtime::message_processor::stable_log] Program return: FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn AA==
[2024-03-15T10:37:30.453645000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn success
[+] Market  SOL -> SOL (LONG) Removed
[+] Set Custom USDC Oracle Price   Instruction
[2024-03-15T10:37:30.455570000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn invoke [1]
[2024-03-15T10:37:30.456544000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: SetCustomOraclePrice
[2024-03-15T10:37:30.458707000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 invoke [2]
[2024-03-15T10:37:30.458736000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 success
[2024-03-15T10:37:30.462214000Z DEBUG solana_runtime::message_processor::stable_log] Program log: pool markets: [8mLT5g4nziqc6cTR9Z2i1Rar71zXL4XzY212nTv1mN4P]
[2024-03-15T10:37:30.462579000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn consumed 39661 of 200000 compute units
[2024-03-15T10:37:30.462603000Z DEBUG solana_runtime::message_processor::stable_log] Program return: FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn AA==
[2024-03-15T10:37:30.462632000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn success
[+] USDC Oracle Price  Set
[+] Set Custom SOL Oracle Price  Instruction
[2024-03-15T10:37:30.465077000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn invoke [1]
[2024-03-15T10:37:30.466121000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: SetCustomOraclePrice
[2024-03-15T10:37:30.468333000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 invoke [2]
[2024-03-15T10:37:30.468367000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 success
[2024-03-15T10:37:30.471826000Z DEBUG solana_runtime::message_processor::stable_log] Program log: pool markets: [8mLT5g4nziqc6cTR9Z2i1Rar71zXL4XzY212nTv1mN4P]
[2024-03-15T10:37:30.472166000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn consumed 41163 of 200000 compute units
[2024-03-15T10:37:30.472185000Z DEBUG solana_runtime::message_processor::stable_log] Program return: FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn AA==
[2024-03-15T10:37:30.472211000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn success
[+] SOL Oracle Price  Set
```



```
[+] Add Liquidity  SOL BOB Instruction
[2024-03-15T10:31:45.961039000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn invoke [1]
[2024-03-15T10:31:45.963566000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: AddLiquidity
[2024-03-15T10:31:45.967069000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Check permissions
[2024-03-15T10:31:45.967229000Z DEBUG solana_runtime::message_processor::stable_log] Program log: initial_custody_amount: 0
[2024-03-15T10:31:45.967244000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Validate inputs
[2024-03-15T10:31:45.969637000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Check pool constraints
[2024-03-15T10:31:45.970117000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Transfer tokens
[2024-03-15T10:31:45.970605000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2024-03-15T10:31:45.971126000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2024-03-15T10:31:45.971839000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 151591 compute units
[2024-03-15T10:31:45.971864000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2024-03-15T10:31:45.972227000Z DEBUG solana_runtime::message_processor::stable_log] Program log: pool_equity_usd: 0
[2024-03-15T10:31:45.972367000Z DEBUG solana_runtime::message_processor::stable_log] Program log: lp token mitn supply: 1
[2024-03-15T10:31:45.972890000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2024-03-15T10:31:45.973325000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: MintTo
[2024-03-15T10:31:45.974032000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4538 of 141314 compute units
[2024-03-15T10:31:45.974050000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2024-03-15T10:31:45.974110000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Update custody stats
[2024-03-15T10:31:45.975079000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Update pool stats
[2024-03-15T10:31:45.975678000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn invoke [2]
[2024-03-15T10:31:45.975987000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn consumed 3632 of 127559 compute units
[2024-03-15T10:31:45.976004000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn success
[2024-03-15T10:31:45.977672000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn consumed 84494 of 200000 compute units
[2024-03-15T10:31:45.977692000Z DEBUG solana_runtime::message_processor::stable_log] Program return: FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn ncQSAAAAAAA=
[2024-03-15T10:31:45.977718000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn success
[+] Liquidity SOL Added
[+]Open Position Alice  MARKET SOL -> SOL (LONG) Instruction
[2024-03-15T10:31:45.979303000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn invoke [1]
[2024-03-15T10:31:45.981623000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: OpenPosition
[2024-03-15T10:31:45.985173000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 invoke [2]
[2024-03-15T10:31:45.985203000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 success
[2024-03-15T10:31:45.986611000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Check permissions
[2024-03-15T10:31:45.986956000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Validate inputs
[2024-03-15T10:31:45.987716000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Entry price: OraclePrice { price: 124230, exponent: -3 }
[2024-03-15T10:31:45.989571000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Discount: 1, Rebate: 1
[2024-03-15T10:31:45.989853000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Initialize new position
[2024-03-15T10:31:45.990100000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Check position risks
[2024-03-15T10:31:45.990118000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Checking leverage..
[2024-03-15T10:31:45.991321000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2024-03-15T10:31:45.991769000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2024-03-15T10:31:45.992479000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 127392 compute units
[2024-03-15T10:31:45.992499000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2024-03-15T10:31:45.993088000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2024-03-15T10:31:45.993521000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2024-03-15T10:31:45.994259000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 119476 compute units
[2024-03-15T10:31:45.994278000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2024-03-15T10:31:45.995521000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Update market stats
[2024-03-15T10:31:45.996245000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn invoke [2]
[2024-03-15T10:31:45.996559000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn consumed 3632 of 104336 compute units
[2024-03-15T10:31:45.996577000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn success
[2024-03-15T10:31:45.998476000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn consumed 108224 of 200000 compute units
[2024-03-15T10:31:45.998509000Z DEBUG solana_runtime::message_processor::stable_log] Program FLASH6Lo6h3iasJKWDs2F8TkW2UKf3s15C8PMGuVfgBn success
>  Position Alice MARKET SOL -> SOL (LONG) Open!
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:H/R:N/S:U (8.8)

Recommendation

To address the identified issue, it is imperative to augment the `remove_market` instruction handler with a remediation process that not only facilitates the correct removal of the market account but also ensures the accurate update of the pool's markets vector to reflect the removal appropriately.

Remediation Plan

SOLVED: The **Flash Trade team** has implemented changes to complete the removal process within the remove_market instruction handler, comprising:
- Updating the pool's markets vector to precisely reflect the market's removal.
- Validation of the market account removal to guarantee its accuracy in the pool.

Remediation Hash

https://github.com/flash-trade/flash-contracts-closed/pull/42/commits/06a5eb4ba74bbc617ddc3cc8423a639c57f6b04a
]

7.3 (HAL-05) MULTIPLE RISKS ASSOCIATED WITH INVALID ORACLE CONFIGURATION

// LOW

Description

The AddCustody instruction allows admin to integrate new custodies, with several parameters required, including the respective oracle values used by the custody. Notably, the **oracle_account**, **custom_oracle_account**, and **oracle_type** parameters play pivotal roles in this process. The **oracle_type** field encompasses three possible values: *None*, *Pyth*, and *Custom*.
Upon completion, the data utilized to add the custody undergoes validation.
programs/perpetuals/src/instructions/add_custody.rs

```
    // record custody data
     let custody = ctx.accounts.custody.as_mut();
     custody.pool = pool.key();
     custody.mint = ctx.accounts.custody_token_mint.key();
     custody.token_account = ctx.accounts.custody_token_account.key();
     custody.decimals = ctx.accounts.custody_token_mint.decimals;
     custody.is_stable = params.is_stable;
     custody.depeg_adjustment = params.depeg_adjustment;
     custody.is_virtual = params.is_virtual;
     custody.oracle = params.oracle;
     custody.pricing = params.pricing;
     custody.permissions = params.permissions;
     custody.fees = params.fees;
     custody.borrow_rate = params.borrow_rate;
     custody.borrow_rate_state.current_rate = params.borrow_rate.base_rate;
     custody.borrow_rate_state.last_update = ctx.accounts.perpetuals.get_time()?;
     custody.bump = *ctx.bumps.get("custody").ok_or(ProgramError::InvalidSeeds)?;
     custody.token_account_bump = *ctx
         .bumps
         .get("custody_token_account")
         .ok_or(ProgramError::InvalidSeeds)?;
    // msg!("custody token account : {:?}", custody.token_account);
     //msg!("custody mint : {:?}", custody.mint);

     if !custody.validate() {
         err!(PerpetualsError::InvalidCustodyConfig)
```

programs/perpetuals/src/state/custody.rs

```
impl Custody {
    pub const LEN: usize = 8 + std::mem::size_of::<Custody>();
    pub fn validate(&self) -> bool {
        (!self.is_virtual || !self.is_stable)
            && self.token_account != Pubkey::default()
            && self.mint != Pubkey::default()
            && self.oracle.validate()
            && self.pricing.validate()
            && self.fees.validate()
            && self.borrow_rate.validate()
```

However, it's crucial to highlight that the only validation conducted for the oracle parameters is the following:
programs/perpetuals/src/state/oracle.rs

```
impl OracleParams {
    pub fn validate(&self) -> bool {
        msg!("validating oracles");
        self.oracle_type == OracleType::None || self.oracle_account !=
Pubkey::default()
    }
}
```

This oversight presents various potential scenarios:

- If the *Pyth* value is provided, it's plausible to input an invalid **oracle_account** that does not genuinely correspond to a Pyth oracle, as such accounts lack proper validation.
- In the case of providing the value *None*, it's even feasible to designate an **oracle_account** as a zero address.

This oversight permits the addition of custody whose **oracle_type** value and **oracle_account**, once created, remains unchangeable via `SetCustodyConfig`. Such a situation may lead to an invalid custody, necessitating its removal via `RemoveCustody`, which succeeds only when the custody token count is zero. Should an invalid custodian be inadvertently added, malicious users could exploit this by transferring a nominal amount to the custodian token count, impeding its removal and subsequently preventing removal from the pool. Consequently, instructions such as `AddLiquidity`, `Swap`, and `RemoveLiquidity`, which require all custodian oracles in the pool as remaining counts, would fail.
Conversely, in cases where the *Custom* value is designated for **oracle_type**, the update of each oracle would mandate a call to the `SetCustomOraclePrice` instruction. However, these updates lack proper validation, potentially yielding uncertain results compared to official oracles.
programs/perpetuals/src/instructions/set_custom_oracle_price.rs

```
pub fn set_custom_oracle_price<'info>(
    ctx: Context<'_, '_, '_, 'info, SetCustomOraclePrice<'info>>,
    params: &SetCustomOraclePriceParams,
) -> Result<u8> {
    // validate signatures
    let mut multisig = ctx.accounts.multisig.load_mut()?;
    let signatures_left = multisig.sign_multisig(
        &ctx.accounts.admin,
        &Multisig::get_account_infos(&ctx)[1..],
        &Multisig::get_instruction_data(AdminInstruction::SetCustomOraclePrice,
params)?,
    )?;
    if signatures_left > 0 {
        msg!(
            "Instruction has been signed but more signatures are required: {}",
            signatures_left
        );
        return Ok(signatures_left);
    }

    // update oracle data
    ctx.accounts.oracle_account.set(
        params.price,
        params.expo,
        params.conf,
        params.ema,
        params.publish_time,
    );
    Ok(0)
```

Furthermore, it's essential to note that the **custom_oracle_account**, despite its inclusion, lacks validation in `AddCustody` or `SetCustodyConfig` instructions and serves no functional purpose.

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:H/D:H/Y:C/R:N/S:U (2.8)

Recommendation

To mitigate the risks associated with invalid oracle configurations, several proactive steps can be taken:

- Refine OracleType Enumeration: Eliminate the *None* value from the OracleType enumeration to prevent inadvertent assignment errors.
- Restrict Custom Oracles to Testing Environments: Limit the usage of custom oracles exclusively to testing scenarios.
- Implement Rigorous Validation Checks: Enhance validation protocols within the `AddCustody` and `SetCustodyConfig` instructions to ensure the accuracy of oracle parameters. Specifically, validate the **oracle_account** to ensure its compatibility with the designated oracle type (*Pyth* or *Custom* for testing environments).
- Flexible Oracle Adjustment: Enable administrators to modify **oracle_account** via the `SetCustodyConfig` instruction, facilitating swift correction of any oracle misconfigurations.
- Streamline Unused Features: Consider removing the **custom_oracle_account** entirely if it no longer serves a practical purpose.

Remediation Plan

PARTIALLY SOLVED: The **Flash Trade team** has been implemented some changes following the recommendations to fix the issue:

- A verification has been added to prevent the **oracle_account** from being invalid during the custody validation when the custody is added or its configuration is set.
- Additionally, the **oracle_account** has been included as an argument in the `SetCustodyConfig` instruction, enabling its modification if required.
- Furthermore, the option to assign *None* as the OracleType to the oracle has been eliminated.

Remediation Hash

https://github.com/flash-trade/flash-contracts-closed/commit/1e44c35c0a5a0cb8927111ef40cf5284be6bca8b

7.4 (HAL-03) CHECK MISSING DURING REWARD VAULT INITIALIZATION

// LOW

Description

The `InitRewardVault` instruction, integral to the fbnft-reward helper program, streamlines the initialization of the reward vault alongside the allocation of a corresponding token account, specifically designed to store NFT rewards earmarked for distribution among NFT owners. In this process, the specification of the **nft_count** parameter plays a pivotal role, signaling the quantity of NFTs eligible for reward distribution. However, an oversight emerges as this parameter lacks validation, thereby permitting it to be erroneously set to zero without an implemented functionality for modification in subsequent stages. Since the **rewards_per_nft** value of reward vault relies on the nft_count' parameter for updates during the `DistributeReward` call, any attempt to execute this instruction subsequently will fail. Consequently, accessing rewards through the `CollectReward` call becomes unattainable.

programs/fbnft-rewards/src/lib.rs

```
pub fn init_reward_vault<'info>(
        ctx: Context<InitRewardVault>,
        params: InitRewardVaultParams,
    ) -> Result<()> {
        let reward_vault = ctx.accounts.reward_vault.as_mut();
        reward_vault.admin = *ctx.accounts.admin.key;
        reward_vault.collection = ctx.accounts.collection_mint.key();
        reward_vault.reward_mint = ctx.accounts.reward_mint.key();
        reward_vault.reward_token_account = ctx.accounts.reward_token_account.key();
        reward_vault.transfer_authority = ctx.accounts.transfer_authority.key();
        reward_vault.bump =
*ctx.bumps.get("reward_vault").ok_or(ProgramError::InvalidSeeds)?;
        reward_vault.token_account_bump =
*ctx.bumps.get("reward_token_account").ok_or(ProgramError::InvalidSeeds)?;
        reward_vault.transfer_authority_bump =
*ctx.bumps.get("transfer_authority").ok_or(ProgramError::InvalidSeeds)?;
        reward_vault.nft_count = params.nft_count;
        reward_vault.rewards_per_nft = 0;
        reward_vault.accrued_amount = 0;
        reward_vault.paid_amount = 0;

        Ok(())
    }
```

programs/fbnft-rewards/src/lib.rs

```
 pub fn distribute_rewards<'info>(
        ctx: Context<DistributeRewards>,
        params: DistributeRewardsParams,
    ) -> Result<()> {
        let reward_vault = ctx.accounts.reward_vault.as_mut();

        // transfer tokens from funding account to reward vault
        reward_vault.transfer_tokens_from(
            ctx.accounts.funding_account.to_account_info(),
            ctx.accounts.reward_token_account.to_account_info(),
            ctx.accounts.admin.to_account_info(),
            ctx.accounts.token_program.to_account_info(),
            params.reward_amount,
        )?;

        reward_vault.accrued_amount =
reward_vault.accrued_amount.checked_add(params.reward_amount as u128).unwrap();
        reward_vault.rewards_per_nft =
reward_vault.accrued_amount.checked_div(reward_vault.nft_count as u128).unwrap() as
u64;
```

programs/fbnft-rewards/src/lib.rs

```
 pub fn collect_reward<'info>(
        ctx: Context<CollectReward>,
    ) -> Result<()> {
        let reward_vault = ctx.accounts.reward_vault.as_mut();
        let reward_record = ctx.accounts.reward_record.as_mut();

        let metadata = &Metadata::try_from(&ctx.accounts.metadata_account).unwrap();
        let collection = metadata.collection.as_ref().unwrap();
        require!(collection.verified, FbnftRewardsError::InvalidCollection);
        require_keys_eq!(collection.key, reward_vault.collection);


        let reward_amount =
reward_vault.rewards_per_nft.checked_sub(reward_record.reward_debt).unwrap();

        // transfer tokens from reward vault to receiving account
        reward_vault.transfer_tokens(
            ctx.accounts.reward_token_account.to_account_info(),
            ctx.accounts.receiving_account.to_account_info(),
            ctx.accounts.transfer_authority.to_account_info(),
            ctx.accounts.token_program.to_account_info(),
            reward_amount,
        )?;
        reward_vault.paid_amount = reward_vault.paid_amount.checked_add(reward_amount
as u128).unwrap();
        reward_record.reward_debt = reward_vault.rewards_per_nft;
```

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:M/D:M/Y:C/R:N/S:U (2.5)

Recommendation

To address this issue effectively, it is recommended to implement validation checks within the `InitRewardVault` instruction to ensure that the **nft_count** parameter is not set to zero. This validation should occur at the outset of the instruction to prevent any erroneous initialization attempts.

Remediation Plan

SOLVED: The **Flash Trade team** has been fixed the identified issue, implementing a validation to ensure that the **nft_count** value assigned during reward vault initialization is non-zero.

Remediation Hash

https://github.com/flash-trade/flash-contracts-closed/commit/1e44c35c0a5a0cb8927111ef40cf5284be6bca8b

7.5 (HAL-07) RISK OF LOSS OF USERS BENEFITS AND REWARDS DUE TO PERPETUALS AND POOL MISCONFIGURATION

// LOW

Description

The Init and AddPool instructions empower administrators to initialize perpetuals accounts and add pools, necessitating various parameter values as follows:

programs/perpetuals/src/instructions/init.rs

```
pub struct InitParams {
    pub min_signatures: u8,
    pub permissions: Permissions,
}
```

programs/perpetuals/src/instructions/add_pool.rs

```


pub struct AddPoolParams {
    pub name: String,
    pub permissions: Permissions,
    pub max_aum_usd: u128,
    pub metadata_title: String,
    pub metadata_symbol: String,
    pub metadata_uri: String,
}
```

However, certain fields of these accounts, such as `staking_fee_share_bps` and `vp_volume_factor` in the pool; and `voltage_multiplier` `trading_discount`, `referral_rebate`, `referral_discount`in perpetuals, are initialized to zero.

programs/perpetuals/src/instructions/init.rs

```
pub fn init(ctx: Context<Init>, params: &InitParams) -> Result<()> {
    // initialize multisig, this will fail if account is already initialized
    let mut multisig = ctx.accounts.multisig.load_init()?;

    multisig.set_signers(ctx.remaining_accounts, params.min_signatures)?;

    // record multisig PDA bump
    multisig.bump = *ctx
        .bumps
        .get("multisig")
        .ok_or(ProgramError::InvalidSeeds)?;
    // record perpetuals
    let perpetuals: &mut Account<'_, Perpetuals> = ctx.accounts.perpetuals.as_mut();
    perpetuals.permissions = params.permissions;
    perpetuals.transfer_authority_bump = *ctx
        .bumps
        .get("transfer_authority")
        .ok_or(ProgramError::InvalidSeeds)?;
    perpetuals.perpetuals_bump = *ctx
        .bumps
        .get("perpetuals")
        .ok_or(ProgramError::InvalidSeeds)?;
    perpetuals.inception_time = perpetuals.get_time()?;

    if !perpetuals.validate() {
        return err!(PerpetualsError::InvalidPerpetualsConfig);
    }
    msg!("perpetuals: {:?}", perpetuals);
    Ok(())
```

programs/perpetuals/src/instructions/add_pool.rs

```
pub fn add_pool<'info>(
    ctx: Context<'_, '_, '_, 'info, AddPool<'info>>,
    params: &AddPoolParams,
) -> Result<u8> {
    // validate inputs
    if params.name.is_empty() || params.name.len() > 64 {
        return Err(ProgramError::InvalidArgument.into());
    }

    // validate signatures
    let mut multisig = ctx.accounts.multisig.load_mut()?;

    let signatures_left = multisig.sign_multisig(
        &ctx.accounts.admin,
        &Multisig::get_account_infos(&ctx)[1..],
        &Multisig::get_instruction_data(AdminInstruction::AddPool, params)?,
    )?;
    if signatures_left > 0 {
        msg!(
            "Instruction has been signed but more signatures are required: {}",
            signatures_left
        );
        return Ok(signatures_left);
    }
    // record pool data
    let perpetuals = ctx.accounts.perpetuals.as_mut();
    let pool = ctx.accounts.pool.as_mut();



    if perpetuals.get_pool_id(&pool.key()).is_ok() {
        // return error if custody is already initialized
        return Err(ProgramError::AccountAlreadyInitialized.into());
    }

    if pool.inception_time != 0 {
        // return error if pool is already initialized
        return Err(ProgramError::AccountAlreadyInitialized.into());
    }


    msg!("Record pool: {}", params.name);
    pool.name = params.name.clone();
    pool.permissions = params.permissions;
    pool.inception_time = perpetuals.get_time()?;
    pool.flp_mint = ctx.accounts.lp_token_mint.key();
    pool.oracle_authority = ctx.accounts.oracle_authority.key();
    pool.max_aum_usd = params.max_aum_usd;
    pool.bump = *ctx.bumps.get("pool").ok_or(ProgramError::InvalidSeeds)?;
    pool.flp_mint_bump = *ctx
        .bumps
        .get("lp_token_mint")
        .ok_or(ProgramError::InvalidSeeds)?;
```

Until administrators invoke SetPoolConfig and SetPerpetualsConfig instructions to modify these values, several consequences may arise:

- Trading account benefits will be nonexistent, impeding level-up opportunities since these values form the basis for such calculations.
- Users initiating a 'DepositStake' call for the first time will have their 'fee_share_bps' initialized to zero in their flp_stake_account, leading to the loss of unclaimed rewards upon collection from the stake since all of them will be directed to protocol_fees. To mitigate this issue, administrators must precede these calls with SetFlpSTakeConfig, assigning appropriate values to each initialized flp account. Additionally, as mentioned in other finding in this report this instruction lacks validation to prevent the assignment of any value other than the corresponding one.
- Furthermore, SetPoolConfig and SetPerpetualsConfig instructions lack validation to prevent zero assignment to the mentioned values. Therefore, despite being called, they may still lead to the same scenario as during initialization.

programs/perpetuals/src/instructions/set_perpetuals_config.rs

```
pub fn set_perpetuals_config<'info>(
    ctx: Context<'_, '_, '_, 'info, SetPerpetualsConfig<'info>>,
    params: &SetPerpetualsConfigParams,
) -> Result<u8> {
    // validate signatures
    let mut multisig = ctx.accounts.multisig.load_mut()?;

    let signatures_left = multisig.sign_multisig(
        &ctx.accounts.admin,
        &Multisig::get_account_infos(&ctx)[1..],
        &Multisig::get_instruction_data(AdminInstruction::SetPerpetualsConfig,
params)?,
    )?;
    if signatures_left > 0 {
        msg!(
            "Instruction has been signed but more signatures are required: {}",
            signatures_left
        );
        return Ok(signatures_left);
    }

    let perpetuals = ctx.accounts.perpetuals.as_mut();

    perpetuals.permissions.allow_ungated_trading = params.allow_ungated_trading;

    perpetuals.voltage_multiplier = params.voltage_multiplier;
    perpetuals.trading_discount = params.trading_discount;
    perpetuals.referral_rebate = params.referral_rebate;
    perpetuals.referral_discount = params.referral_discount;

    if !perpetuals.validate() {
        return err!(PerpetualsError::InvalidPerpetualsConfig);
    }
```

programs/perpetuals/src/state/perpetuals.rs

```
    pub fn validate(&self) -> bool {
        for i in 0..6 {
            if self.trading_discount[i] as u128 > Self::RATE_POWER
                || self.referral_rebate[i] as u128 > Self::RATE_POWER
                || self.referral_discount as u128 > Self::RATE_POWER {
                return false;
            }
        }
        true
    }
```

programs/perpetuals/src/instructions/set_pool_config.rs

```
pub fn set_pool_config<'info>(
    ctx: Context<'_, '_, '_, 'info, SetPoolConfig<'info>>,
    params: &SetPoolConfigParams,
) -> Result<u8> {
    // validate signatures
    let mut multisig = ctx.accounts.multisig.load_mut()?;

    let signatures_left = multisig.sign_multisig(
        &ctx.accounts.admin,
        &Multisig::get_account_infos(&ctx)[1..],
        &Multisig::get_instruction_data(AdminInstruction::SetPoolConfig, params)?,
    )?;
    if signatures_left > 0 {
        msg!(
            "Instruction has been signed but more signatures are required: {}",
            signatures_left
        );
        return Ok(signatures_left);
    }

    let pool = ctx.accounts.pool.as_mut();

    pool.permissions = params.permissions;
    pool.oracle_authority = params.oracle_authority;
    pool.max_aum_usd = params.max_aum_usd;
    pool.staking_fee_share_bps = params.staking_fee_share_bps;
    pool.vp_volume_factor = params.vp_volume_factor;

    if !pool.validate() {
        return err!(PerpetualsError::InvalidPoolConfig);
    }
```

programs/perpetuals/src/state/pool.rs

```
 pub fn validate(&self) -> bool {
        for ratio in &self.ratios {
            if !ratio.validate() {
                return false;
            }
        }
        // check target ratios add up to 1
        if !self.ratios.is_empty()
            && self
                .ratios
                .iter()
                .map(|&x| (x.target as u128))
                .sum::<u128>()
                != Perpetuals::BPS_POWER
        {
            return false;
        }

        // check custodies are unique
        for i in 1..self.custodies.len() {
            if self.custodies[i..].contains(&self.custodies[i - 1]) {
                return false;
            }
        }

        // check markets are unique
        for i in 1..self.markets.len() {
            if self.markets[i..].contains(&self.markets[i - 1]) {
                return false;
            }
        }

        !self.name.is_empty() && self.name.len() <= 64 && self.custodies.len() ==
self.ratios.len()
    }
```

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:L/D:M/Y:C/R:N/S:U (2.4)

Recommendation

To address this issue, it is recommended to take the following steps:
1. Initialize the values of these fields during account creation, setting them to valid and expected values. This ensures that the account starts with appropriate configurations, minimizing the risk of erroneous behavior.
2. Add a check to the instructions responsible for modifying these values, to verify that they are not set to zero.

By implementing these measures, the system can ensure that perpetuals and pools are properly configured from the outset and that any modifications to their parameters are validated to prevent potential issues and loss of user benefits and rewards.

Remediation Plan

PARTIALLY SOLVED: The **Flash Trade team** has been implemented several key adjustments. Firstly, the values `staking_fee_share_bps` and `vp_volume_factor` have been included as arguments in the `AddPool` instruction. This ensures they are explicitly set during pool creation, preventing them from being initialized to zero by default as previously. Additionally, a validation has been implemented to ensure `staking_fee_share_bps` cannot be set to zero.
Furthermore, to prevent previous default initialization for `voltage_multiplier`, `trading_discount`, `referral_rebate`, and `referral_discount`, these values have been added as arguments in the `Init` instruction.
These modifications empower administrators to define these values upon initialization. However, apart from `staking_fee_share_bps`, the other added values lack validation checks and could still inadvertently be set to zero.

Remediation Hash

https://github.com/flash-trade/flash-contracts-closed/commit/1e44c35c0a5a0cb8927111ef40cf5284be6bca8b

7.6 (HAL-04) RISK OF LOST UNCLAIMED REWARDS DUE TO IF FLP STAKE MISCONFIGURATION

// LOW

Description

A user's Flp stake account encompasses various fields, with the **fee_share_bps** parameter being of utmost importance. Upon initialization triggered by the `DepositStake` call, this parameter mirrors the pool's **staking_fee_share_bps**, aligning the user's stake with the pool's fee structure. Furthermore, **fee_share_bps** plays a pivotal role in determining the user's rewards, slated for collection in the `CollectStakeRewards` instruction.
The `SetFlpStakeConfig` instruction, administered by an admin, allows for the adjustment of the **fee_share_bps** value within a user's Flp stake account, typically set to 70%. However, a critical oversight emerges: the new value undergoes insufficient validation, potentially permitting it to be erroneously set, even to zero, without adequate checks.
This oversight could lead to unintended consequences if the instruction is not invoked again to rectify the value. In such a scenario where it is set to zero, when the user requests a reward collection via `CollectStakeReward`, their rewards are calculated as zero, resulting in the user receiving no rewards. Consequently, any unclaimed rewards are forfeited, diverted entirely towards protocol fees.
programs/perpetuals/src/instructions/deposit_stake.rs

```
if flp_stake.is_initialized == false {
        flp_stake.fee_share_bps = pool.staking_fee_share_bps;
        flp_stake.bump =
*ctx.bumps.get("flp_stake_account").ok_or(ProgramError::InvalidSeeds)?;
        flp_stake.is_initialized = true;
    }
    flp_stake.owner = *ctx.accounts.owner.key;
```

programs/perpetuals/src/instructions/set_flp_stake_config.rs

```
pub fn set_flp_stake_config<'info>(
    ctx: Context<'_, '_, '_, 'info, SetFlpStakeConfig<'info>>,
    params: &SetFlpStakeConfigParams,
) -> Result<u8> {
    // validate signatures
    let mut multisig = ctx.accounts.multisig.load_mut()?;

    let signatures_left = multisig.sign_multisig(
        &ctx.accounts.admin,
        &Multisig::get_account_infos(&ctx)[1..],
        &Multisig::get_instruction_data(AdminInstruction::SetFlpStakeConfig,
params)?,
    )?;
    if signatures_left > 0 {
        msg!(
            "Instruction has been signed but more signatures are required: {}",
            signatures_left
        );
        return Ok(signatures_left);
    }

    let flp_stake = ctx.accounts.flp_stake_account.as_mut();
    flp_stake.fee_share_bps = params.fee_share_bps;
```

programs/perpetuals/src/instructions/collect_stake_reward.rs

```
   let user_reward = math::checked_as_u64(math::checked_div(
        math::checked_mul(flp_stake.unclaimed_rewards as u128,
            flp_stake.fee_share_bps as u128
        )?,
        Perpetuals::BPS_POWER
    )?)?;

    // transfer tokens to user
    msg!("Transfer flp tokens");
    perpetuals.transfer_tokens(
        ctx.accounts.fee_custody_token_account.to_account_info(),
        ctx.accounts.receiving_token_account.to_account_info(),
        ctx.accounts.transfer_authority.to_account_info(),
        ctx.accounts.token_program.to_account_info(),
        user_reward,
    )?;

    fee_custody.fees_stats.paid = math::checked_add(fee_custody.fees_stats.paid,
user_reward as u128)?;
    fee_custody.fees_stats.protocol_fee = math::checked_add(
        fee_custody.fees_stats.protocol_fee,
        math::checked_sub(flp_stake.unclaimed_rewards, user_reward)?
    )?;

    if ctx.remaining_accounts.len() == 1 {
        let mut trading_account = Box::new(Account::
<Trading>::try_from(&ctx.remaining_accounts[0])?);
        trading_account.stats.lp_rewards_usd =
math::checked_add(trading_account.stats.lp_rewards_usd, user_reward as u128)?;
        // convert this to usd value later when shofting from non usdc fee
distribution
        trading_account.exit(&ID)?;
    }
    flp_stake.unclaimed_rewards = 0;
```

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:C/R:N/S:U (2.0)

Recommendation

To effectively address this issue, it is advisable to bolster the `SetFlpStakeConfig` instruction with robust validation checks. These checks should meticulously verify that the provided **fee_share_bps** value is both valid and non-zero. By conducting this validation prior to any alterations to the user's Flp stake account, the likelihood of misconfigurations can be significantly reduced.
Additionally, it is worth considering the synchronization of this value with the current pool's **staking_fee_share_bps** if the latter is non-zero. This ensures alignment between the user's Flp stake configuration and the prevailing parameters of the associated pool, promoting consistency and minimizing potential discrepancies.

Remediation Plan

SOLVED: The **Flash Trade team** fixed this issue in commits `1e44c35` and `60b3258`: several adjustments have been made to ensure that the `staking_fee_share_bps` of the pool cannot be set to zero. These modifications have been applied to the **AddPool**, **InitStaking**, **SetPoolConfig** instructions.
Additionally, measures have been taken to validate the `fee_share_bps` of the user's `flp_stake_account` to prevent it from being set in the **SetFlpStakingConfig** instruction.

Remediation Hash

https://github.com/Halborn/Flash-trade--flash-contracts-closed/commit/60b32588eaa37bb529d25f5a1ed76ab4e68cdc47

7.7 (HAL-09) MISSING VALIDATION FOR ORACLE AUTHORITY ADDRESS

// INFORMATIONAL

Description

The AddPool instruction empowers administrators to add pools by supplying multiple accounts, including **oracle_authority**.

```
#[instruction(params: AddPoolParams)]
pub struct AddPool<'info> {
    #[account(mut)]
    pub admin: Signer<'info>,

    /// CHECK: authority for backup oracle
    pub oracle_authority: AccountInfo<'info>,
```

This account is designated for oracles of type Pyth, and it comes into play if the last update time of the oracle surpasses the value provided for **max_price_age_sec** (corresponding to oracleParam in AddCustody and SetCustodyConfig).
However, the instruction, as SetPoolConfig, currently lacks a validation check for the **oracle_authority** address.
programs/perpetuals/src/instructions/add_pool.rs

```
pool.name = params.name.clone();
    pool.permissions = params.permissions;
    pool.inception_time = perpetuals.get_time()?;
    pool.flp_mint = ctx.accounts.lp_token_mint.key();
    pool.oracle_authority = ctx.accounts.oracle_authority.key();
    pool.max_aum_usd = params.max_aum_usd;
    pool.bump =
*ctx.bumps.get("pool").ok_or(ProgramError::InvalidSeeds)?;
    pool.flp_mint_bump = *ctx
        .bumps
        .get("lp_token_mint")
        .ok_or(ProgramError::InvalidSeeds)?;
```

Consequently, it allows even a zero address to be set, which leads to a failure of the instruction when attempting to retrieve the oracle price in the aforementioned scenario.
programs/perpetuals/src/state/oracle.rs

```
if last_update_age_sec <= max_price_age_sec as i64 {
            Ok((
                OraclePrice::new(math::checked_as_u64(pyth_price.price)?,
pyth_price.expo),
                OraclePrice::new(math::checked_as_u64(pyth_ema_price.price)?,
pyth_ema_price.expo),
                pyth_price.conf,
                false,
                false
            ))
        } else {
            // Pyth oracle price is stale on mainnet, try to fetch price from
permissionless cache
            // Get what should be the Ed25519Program signature verification
instruction.
            let signature_ix: Instruction =
                sysvar::instructions::load_instruction_at_checked(0, ix_sysvar)?;
            /*require_eq!(
                signature_ix.program_id,
                ed25519_program::ID,
                PerpetualsError::PermissionlessOracleMissingSignature
            );*/
            let expected_size = 112 + 4 + (36 * custodies_len); //expected size for
Vec<CustomOracle>
            require!(
                signature_ix.accounts.is_empty() // no accounts touched
                    && signature_ix.data[0] == 0x01 // only one ed25519 signature
                    && signature_ix.data.len() == expected_size, // data len matches
exactly the expected
                PerpetualsError::PermissionlessOracleMalformedEd25519Data
            );
            // Manually access offsets for signer pubkey and message data according
to:
            // https://docs.solana.com/developing/runtime-
facilities/programs#ed25519-program
            let signer_pubkey = &signature_ix.data[16..16 + 32];
            require!(
                signer_pubkey == authority.to_bytes(),
                PerpetualsError::PermissionlessOracleSignerMismatch
            );
```

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:H/D:M/Y:C/R:P/S:U (1.3)

Recommendation

To address the issue, it is necessary to implement a validation check within the AddPooland SetPoolConfig instructions to ensure that the **oracle_authority** address provided is not set to zero. This

can be achieved by adding a conditional statement that verifies the validity of the address before proceeding with the pool addition or modification process. If the address is found to be zero or invalid, an appropriate error message should be returned, indicating that a valid oracle authority address must be provided.
This enhancement will help prevent failures in the instruction due to invalid oracle authority addresses and ensure the smooth execution of pool addition operations.

Remediation Plan

ACKNOWLEDGED: The **Flash team** acknowledged this finding.

7.8 (HAL-02) REDUNDANT PERMISSIONS

// INFORMATIONAL

Description

Perpetuals, pools, and custody are structured with diverse data components, including a spectrum of permissions that govern operations within the program based on their designated values. These values are established during the initialization of these accounts through instructions like `Init`, `AddPool`, and `AddCustody`, and they remain adjustable via instructions such as `SetPermissions`, `SetPoolConfig` , and `SetCustodyConfig`. However, specific permissions like `allow_flp_staking`, `allow_liquidation`, `allow_fee_discounts`, and `allow_referral_rebates` lack operational control utility or any discernible purpose, rendering them irrelevant.

programs/perpetuals/src/state/perpetuals.rs

```
pub struct Permissions {
    pub allow_swap: bool,
    pub allow_add_liquidity: bool,
    pub allow_remove_liquidity: bool,
    pub allow_open_position: bool,
    pub allow_close_position: bool,
    pub allow_collateral_withdrawal: bool,
    pub allow_size_change: bool,
    pub allow_liquidation: bool,
    pub allow_flp_staking: bool,
    pub allow_fee_distribution: bool,
    pub allow_ungated_trading: bool,
    pub allow_fee_discounts: bool,
    pub allow_referral_rebates: bool,
}
```

programs/perpetuals/src/instructions/init.rs

```
pub struct InitParams {
    pub min_signatures: u8,
    pub permissions: Permissions,
}
```

programs/perpetuals/src/instructions/add_pool.rs

```
pub struct AddPoolParams {
    pub name: String,
    pub permissions: Permissions,
    pub max_aum_usd: u128,
    pub metadata_title: String,
    pub metadata_symbol: String,
    pub metadata_uri: String,
}
```

programs/perpetuals/src/instructions/add_custody.rs

```
pub struct AddCustodyParams {
    pub is_stable: bool,
    pub depeg_adjustment: bool,
    pub is_virtual: bool,
    pub oracle: OracleParams,
    pub pricing: PricingParams,
    pub permissions: Permissions,
    pub fees: Fees,
    pub borrow_rate: BorrowRateParams,
    pub ratios: Vec<TokenRatios>,
}
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation

To address this issue effectively, it is advisable to remove the redundant permissions that lack operational significance. These permissions ought to be eliminated from both the initialization process of Perpetuals, pools, and custody, as well as from the instructions responsible for modifying their configuration. Furthermore, it is essential to review and adjust any associated functionality or logic to align with the removal of these permissions. By taking these steps, the system will be streamlined and optimized, enhancing its clarity and functionality.

Remediation Plan

ACKNOWLEDGED: The **Flash team** acknowledged this finding.

7.9 (HAL-08) LACK OF ERROR HANDLING IF DIFFERENT COLLATERAL CUSTODY IN PERP COMPOSABILITY

// INFORMATIONAL

Description

Perp-composability enables users to execute swap operations concurrently with other actions such as opening or closing positions, as well as adding and removing collateral, all within a single instruction call. Specifically, these combinations include:

- Swap and OpenPosition
- Swap and AddCollateral
- ClosePosition and Swap
- RemoveCollateral and Swap

To achieve this, users must provide all necessary accounts to execute the set of transactions as desired. However, the current implementation lacks validation to ensure that the dispensing custody of the swap in the first and second instructions corresponds to the collateral custody. Similarly, the collateral custody of the swap in the third and fourth instructions does not necessarily match the receiving custody.
While this oversight does not pose a direct security risk as the second part of the instruction will fail in the perpetuals program if mismatches occur, it is advisable to add validation in perp-composability to gracefully handle errors.

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation

To address the issue, it is essential to implement validation checks within the perp-composability functionality to ensure the alignment of custody accounts in swap operations. This can be achieved by verifying that the dispensing custody of the swap matches the collateral custody in the first and second instructions, and that the collateral custody of the swap matches the receiving custody in the third and fourth instructions.
Additionally, error handling mechanisms should be incorporated to gracefully handle cases where custody mismatches occur. This could involve returning appropriate error messages or instructions to guide users in rectifying the discrepancies.

Remediation Plan

ACKNOWLEDGED: The **Flash team** acknowledged this finding.

Remediation Hash

https://github.com/flash-trade/flash-contracts-closed/commit/1e44c35c0a5a0cb8927111ef40cf5284be6bca8b

8. AUTOMATED TESTING

Cargo-Audit

ID	Crate	Description
RUSTSEC-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on `ed25519-dalek`
RUSTSEC-2024-0003	h2	Resource exhaustion vulnerability in h2 may lead to Denial of Service (DoS)
RUSTSEC-2024-0019	mio	Tokens for named pipes may be delivered after deregistration
RUSTSEC-2023-0065	tungstenite	Tungstenite allows remote attackers to cause a denial of service

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.