Perpetual Trading & Staking

Smart Contract Audit Report
Prepared for Perp88

Date Issued:	Nov 8, 2022
Project ID:	AUDIT2022048
Version:	v1.0
Confidentiality Level:	Public

inspeX

CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2022048
Version	v1.0
Client	Perp88
Project	Perpetual Trading & Staking
Auditor(s)	Natsasit Jirathammanuwat Darunphop Pengkumta Phitchakorn Apiratisakul Sorawish Laovakul
Author(s)	Natsasit Jirathammanuwat Darunphop Pengkumta Phitchakorn Apiratisakul Sorawish Laovakul
Reviewer	Peeraphut Punsuwan
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Nov 8, 2022	Full report	Natsasit Jirathammanuwat Darunphop Pengkumta Phitchakorn Apiratisakul Sorawish Laovakul

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by Perp88, Inspex team conducted an audit to verify the security posture of the Perpetual Trading & Staking smart contracts between Oct 12, 2022 and Oct 31, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Perpetual Trading & Staking smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 5 critical, 5 high, 7 medium, 4 low, 2 very low, and 1 info-severity issues. With the project team's prompt response 5 critical, 5 high, 7 medium, 3 low, 2 very low, and 1 info-severity issues were resolved or mitigated in the reassessment, while 1 low-severity issue was acknowledged by the team. Therefore, Inspex trusts that Perpetual Trading & Staking smart contracts have sufficient protections to be safe for public use. However, in the long run, Inspex suggests resolving all issues found in this report.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Perp88 is a decentralized perpetual trading platform that provides the ability for leveraged and spot trading against the liquidity pool. The liquidity provider can deposit eligible assets into the liquidity pool to earn fees on the platform. The assets in the pool can be utilized by deploying funds to the farming strategy to earn additional yield.

When the users open the leveraged position, the funds in the liquidity pool will be reserved for the maximum profit of the position. Opening a long position requires the user to provide the same assets as collateral. Opening a short position requires the user to provide stable coins as collateral. When a position is closed, the profit or loss will be calculated and reflected in the user's position.

Liquidity providers can deposit the eligible assets into the liquidity pools for obtaining the PLP token. The PLP token can be used for staking to earn profit from countertrading and protocol revenue (position opening and closing fees, borrowing fees) in the reward token.

Scope Information:

Project Name	Perpetual Trading & Staking
Website	https://perp88.com/
Smart Contract Type	Ethereum Smart Contract
Chain	Polygon
Programming Language	Solidity
Category	AMM, Futures, Yield Farming

Audit Information:

Audit Method	Whitebox
Audit Date	Oct 12, 2022 - Oct 31, 2022
Reassessment Date	Nov 8, 2022

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

Initial Audit: (Commit: 95e6e456fd170b480935c6df547a23b14e5c0d9f)

Contract	Location (URL)
PoolOracle	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/PoolOracle.sol
Orderbook	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/Orderbook.sol
PoolDiamond	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/PoolDiamond.sol
PoolRouter	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/PoolRouter.sol
AccessControlFacet	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/facets/AccessControlFacet.sol
AdminFacet	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/facets/AdminFacet.sol
DiamondCutFacet	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/facets/DiamondCutFacet.sol
DiamondLoupeFacet	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/facets/DiamondLoupeFacet.sol
FarmFacet	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/facets/FarmFacet.sol
FundingRateFacet	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/facets/FundingRateFacet.sol
GetterFacet	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/facets/GetterFacet.sol
LiquidityFacet	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/facets/LiquidityFacet.sol
OwnershipFacet	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/facets/OwnershipFacet.sol
PerpTradeFacet	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/facets/PerpTradeFacet.sol

TransparentUpgradeableProxy	https://github.com/perp88/contracts/blob/95e6e456fd/src/proxy/TransparentUpgradeableProxy.sol
AdHocMintRewarder	https://github.com/perp88/contracts/blob/95e6e456fd/src/staking/AdHocMintRewarder.sol
Compounder	https://github.com/perp88/contracts/blob/95e6e456fd/src/staking/Compounder.sol
FeedableRewarder	https://github.com/perp88/contracts/blob/95e6e456fd/src/staking/FeedableRewarder.sol
PLPStaking	https://github.com/perp88/contracts/blob/95e6e456fd/src/staking/PLPStaking.sol
RewardDistributor	https://github.com/perp88/contracts/blob/95e6e456fd/src/staking/RewardDistributor.sol
WFeedableRewarder	https://github.com/perp88/contracts/blob/95e6e456fd/src/staking/WFeedableRewarder.sol
PLP	https://github.com/perp88/contracts/blob/95e6e456fd/src/tokens/PLP.sol
Math	https://github.com/perp88/contracts/blob/95e6e456fd/src/utils/Math.sol
Multicall	https://github.com/perp88/contracts/blob/95e6e456fd/src/utils/Multicall.sol
Vester	https://github.com/perp88/contracts/blob/95e6e456fd/src/vesting/Vester.sol
AccessControlInitializer	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/initializers/AccessControlInitializer.sol
DiamondInitializer	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/initializers/DiamondInitializer.sol
PoolConfigInitializer	https://github.com/perp88/contracts/blob/95e6e456fd/src/core/pool-diamond/initializers/PoolConfigInitializer.sol

Reassessment: (Commit: 87a0aa0cc0d0bcf78d3c79d181d416550098654e)

Contract	Location (URL)
PoolOracle	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/PoolOracle.sol
Orderbook	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/Orderbook.sol
PoolDiamond	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/PoolDiamond.sol

PoolRouter	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/PoolRouter.sol
AccessControlFacet	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/facets/AccessControlFacet.sol
AdminFacet	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/facets/AdminFacet.sol
DiamondCutFacet	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/facets/DiamondCutFacet.sol
DiamondLoupeFacet	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/facets/DiamondLoupeFacet.sol
FarmFacet	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/facets/FarmFacet.sol
FundingRateFacet	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/facets/FundingRateFacet.sol
GetterFacet	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/facets/GetterFacet.sol
LiquidityFacet	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/facets/LiquidityFacet.sol
OwnershipFacet	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/facets/OwnershipFacet.sol
PerpTradeFacet	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/facets/PerpTradeFacet.sol
TransparentUpgradeableProxy	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/proxy/TransparentUpgradeableProxy.sol
AdHocMintRewarder	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/staking/AdHocMintRewarder.sol
Compounder	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/staking/Compounder.sol
FeedableRewarder	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/staking/FeedableRewarder.sol
PLPStaking	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/staking/PLPStaking.sol
RewardDistributor	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/staking/RewardDistributor.sol

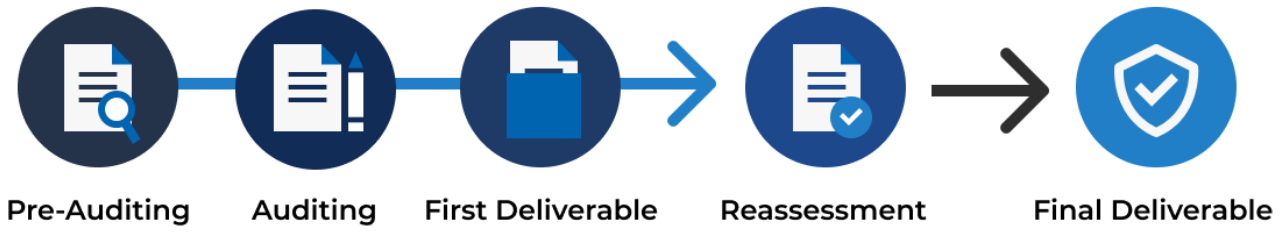
WFeedableRewarder	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/staking/WFeedableRewarder.sol
PLP	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/tokens/PLP.sol
Math	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/utils/Math.sol
Multicall	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/utils/Multicall.sol
Vester	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/vesting/Vester.sol
AccessControlInitializer	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/initializers/AccessControlInitializer.sol
DiamondInitializer	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/initializers/DiamondInitializer.sol
PoolConfigInitializer	https://github.com/perp88/contracts/blob/87a0aa0cc0/solidity/contracts/core/pool-diamond/initializers/PoolConfigInitializer.sol

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The testing items checked are based on our Smart Contract Security Testing Guide (SCSTG) v1.0 (https://github.com/InspexCo/SCSTG/releases/download/v1.0/SCSTG_v1.0.pdf) which covers most prevalent risks in smart contracts. The latest version of the document can also be found at https://inspex.gitbook.io/testing-guide/.

The following audit items were checked during the auditing activity:

Testing Category	Testing Items
1. Architecture and Design	1.1. Proper measures should be used to control the modifications of smart contract logic 1.2. The latest stable compiler version should be used 1.3. The circuit breaker mechanism should not prevent users from withdrawing their funds 1.4. The smart contract source code should be publicly available 1.5. State variables should not be unfairly controlled by privileged accounts 1.6. Least privilege principle should be used for the rights of each role
2. Access Control	2.1. Contract self-destruct should not be done by unauthorized actors 2.2. Contract ownership should not be modifiable by unauthorized actors 2.3. Access control should be defined and enforced for each actor roles 2.4. Authentication measures must be able to correctly identify the user 2.5. Smart contract initialization should be done only once by an authorized party 2.6. tx.origin should not be used for authorization
3. Error Handling and Logging	3.1. Function return values should be checked to handle different results 3.2. Privileged functions or modifications of critical states should be logged 3.3. Modifier should not skip function execution without reverting
4. Business Logic	4.1. The business logic implementation should correspond to the business design 4.2. Measures should be implemented to prevent undesired effects from the ordering of transactions 4.3. msg.value should not be used in loop iteration
5. Blockchain Data	5.1. Result from random value generation should not be predictable 5.2. Spot price should not be used as a data source for price oracles 5.3. Timestamp should not be used to execute critical functions 5.4. Plain sensitive data should not be stored on-chain 5.5. Modification of array state should not be done by value 5.6. State variable should not be used without being initialized

Testing Category	Testing Items
6. External Components	6.1. Unknown external components should not be invoked 6.2. Funds should not be approved or transferred to unknown accounts 6.3. Reentrant calling should not negatively affect the contract states 6.4. Vulnerable or outdated components should not be used in the smart contract 6.5. Deprecated components that have no longer been supported should not be used in the smart contract 6.6. Delegatecall should not be used on untrusted contracts
7. Arithmetic	7.1. Values should be checked before performing arithmetic operations to prevent overflows and underflows 7.2. Explicit conversion of types should be checked to prevent unexpected results 7.3. Integer division should not be done before multiplication to prevent loss of precision
8. Denial of Services	8.1. State changing functions that loop over unbounded data structures should not be used 8.2. Unexpected revert should not make the whole smart contract unusable 8.3. Strict equalities should not cause the function to be unusable
9. Best Practices	9.1. State and function visibility should be explicitly labeled 9.2. Token implementation should comply with the standard specification 9.3. Floating pragma version should not be used 9.4. Builtin symbols should not be shadowed 9.5. Functions that are never called internally should not have public visibility 9.6. Assert statement should not be used for validating common conditions

3.3. Risk Rating

OWASP Risk Rating Methodology (https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker
- **Impact**: a measure of the damage caused by a successful attack

Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

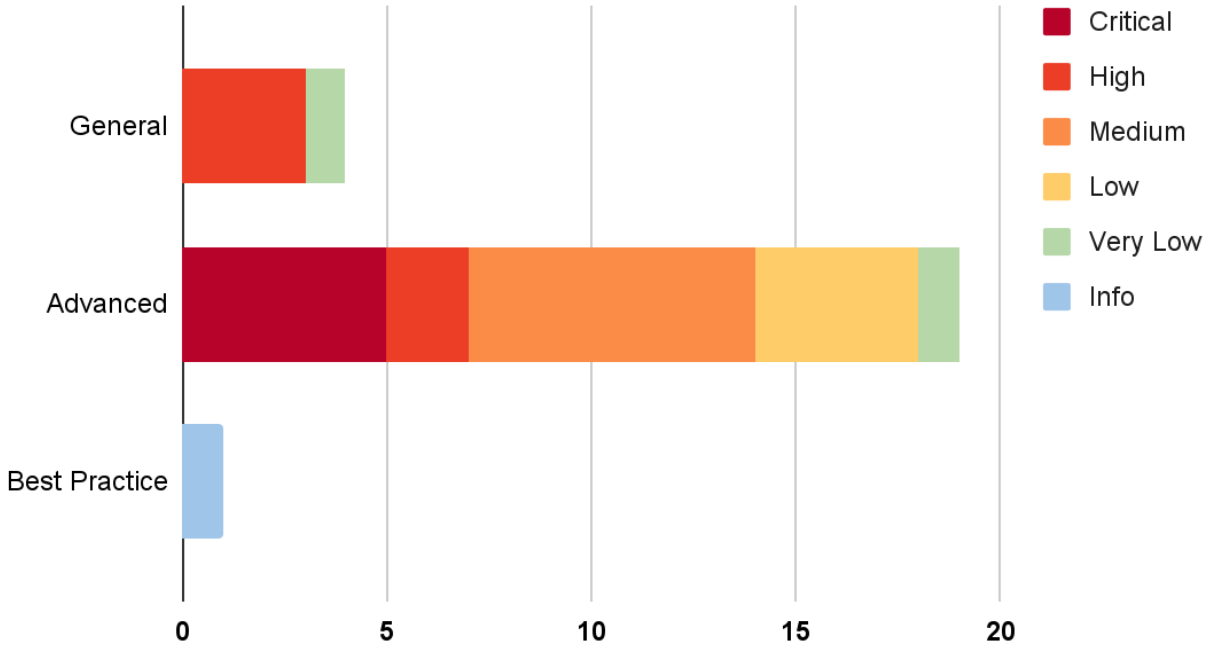
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Impact \ Likelihood	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

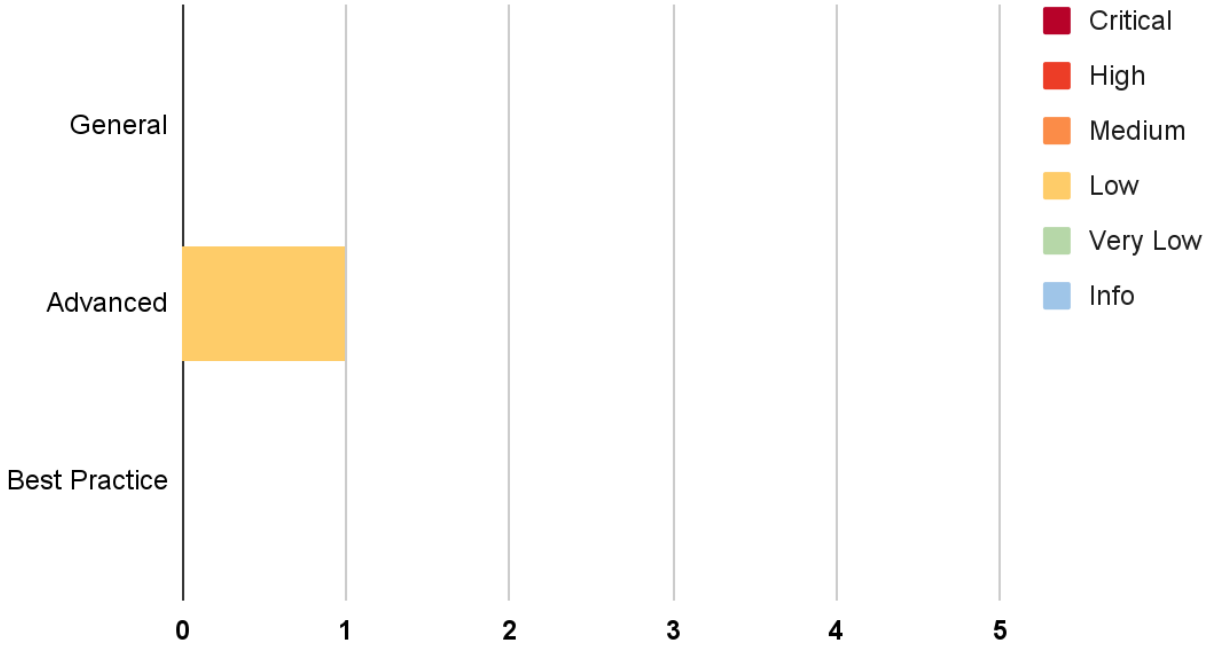
4. Summary of Findings

The following charts show the number of the issues found during the assessment and the issues acknowledged in the reassessment, categorized into three categories: **General**, **Advanced**, and **Best Practice**.

Assessment:



Reassessment:



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Improper Flashloan Implementation	Advanced	Critical	Resolved
IDX-002	Insufficient Pool Address Check	Advanced	Critical	Resolved
IDX-003	Lack of Position Healthy Check	Advanced	Critical	Resolved
IDX-004	Missing Update for totalOf state in Farm Function	Advanced	Critical	Resolved
IDX-005	Missing Update for lastFundingTimeOf of Collateral Token	Advanced	Critical	Resolved
IDX-006	Use of Upgradable Contract Design	General	High	Resolved *
IDX-007	Improper Profit Transfer	Advanced	High	Resolved
IDX-008	Centralized Control of State Variable	General	High	Resolved *
IDX-009	Arbitrary PLP Token Minting	Advanced	High	Resolved *
IDX-010	External Call to Untrusted Third Party Component	General	High	Resolved *
IDX-011	Denial of Service from Rewarders Configuration	Advanced	Medium	Resolved
IDX-012	Improper Share Calculation	Advanced	Medium	Resolved *
IDX-013	Improper Sanity Check	Advanced	Medium	Resolved
IDX-014	User Reward Miscalculation	Advanced	Medium	Resolved

IDX-015	Arbitrary Reward Rate Set	Advanced	Medium	Resolved
IDX-016	Insufficient roundDepth Input Validation	Advanced	Medium	Resolved
IDX-017	Design Flaw in Fixed Rate Token Swap	Advanced	Medium	Resolved *
IDX-018	Arbitrary Resetting PLP Transfer Cooldown	Advanced	Low	Acknowledged
IDX-019	Improper Parameter Control of Calculation Parameter	Advanced	Low	Resolved
IDX-020	Incorrect Order Book Swap Rate	Advanced	Low	Resolved
IDX-021	Reward Loss When Using onWithdraw() Function	Advanced	Low	Resolved *
IDX-022	Insufficient Logging for Privileged Functions	General	Very Low	Resolved
IDX-023	Missing Duplication Check	Advanced	Very Low	Resolved
IDX-024	Use of Deprecated Function	Best Practice	Info	Resolved

* The mitigations or clarifications by Perp88 can be found in Chapter 5.

5. Detailed Findings Information

5.1. Improper Flashloan Implementation

ID	IDX-001
Target	LiquidityFacet
Category	Advanced Smart Contract Vulnerability
CWE	CWE-841: Improper Enforcement of Behavioral Workflow
Risk	**Severity: Critical** **Impact: High** The attacker can use the `flashLoan()` function to drain the tokens from the pool without repaying the loan. **Likelihood: High** This issue occurs when any functions modify the `poolV1ds.totalOf[tokens]` state in the callback of the `flashLoan()` function. For example, the `claimAndFeedProtocolRevenue()` function can be called inside the `onFlashLoan()` callback function without any restriction.
Status	**Resolved** The Perp88 team has resolved this issue by implementing the check of the `poolV1ds.totalOf` state properly.

5.1.1. Description

The Perp88 platform provides the flash loan service with the `flashLoan()` function in the `LiquidityFacet` contract. After all tokens are transferred to the borrower, the `onFlashLoan()` callback function is called in line 487. At the end, the borrower must return all borrowed tokens, including the fee, the balance of `token[i]` in the pool must be greater than the `poolV1ds.totalOf[tokens[i]]` + `fee[i]` in order to pass the check in lines 490-493.

LiquidityFacet.sol

```
463 function flashLoan(
464   FlashLoanBorrowerInterface borrower,
465   address[] calldata receivers,
466   address[] calldata tokens,
467   uint256[] calldata amounts,
468   bytes calldata data
469 ) external nonReentrant {
470   if (receivers.length != tokens.length || receivers.length != amounts.length)
471     revert LiquidityFacet_BadLength();
472
```

```
473   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
474     .poolV1DiamondStorage();
475
476   uint256[] memory fees = new uint256[](tokens.length);
477   for (uint256 i = 0; i < tokens.length; ) {
478     fees[i] = (amounts[i] * LibPoolConfigV1.flashLoanFeeBps()) / BPS;
479
480     ERC20(tokens[i]).safeTransfer(receivers[i], amounts[i]);
481
482     unchecked {
483       ++i;
484     }
485   }
486
487   borrower.onFlashLoan(msg.sender, tokens, amounts, fees, data);
488
489   for (uint256 i = 0; i < tokens.length; ) {
490     if (
491       ERC20(tokens[i]).balanceOf(address(this)) <
492       poolV1ds.totalOf[tokens[i]] + fees[i]
493     ) revert LiquidityFacet_BadFlashLoan();
494
495     // Collect fee
496     poolV1ds.feeReserveOf[tokens[i]] += fees[i];
497
498     emit FlashLoan(
499       address(borrower),
500       tokens[i],
501       amounts[i],
502       fees[i],
503       receivers[i]
504     );
505
506     unchecked {
507       ++i;
508     }
509   }
510 }
```

Normally, the `poolV1ds.totalOf` state should always sync with the balance of tokens in the pool. However, if the `poolV1ds.totalOf` state is updated while the token is borrowed out, the `poolV1ds.totalOf` state will be incorrect. The `poolV1ds.totalOf` state can be updated in many functions.

Since the `flashLoan()` function has the `nonReentrant` modifier, the function that can be used to update the `poolV1ds.totalOf` state inside the `onFlashLoan()` callback function must not have the `nonReentrant` modifier either. Unfortunately, there are several functions that still match the conditions as shown in the following table.

Target	Function	Caller
AdminFacet.sol (L: 423)	withdrawFeeReserve()	Only Treasury
FarmFacet.sol (L: 54)	setStrategyOf()	Only Owner
FarmFacet.sol (L: 121)	farm()	Only pool or FarmKeeper
PerpTradeFacet.sol (L: 773)	liquidate()	Allowed Liquidator or Anyone (if `isAllowAllLiquidators` is set to `true`)
RewardDistributor.sol (L: 102)	claimAndFeedProtocolRevenue()	Anyone

For example, the `withdrawFeeReserve()` function has a call to the `LibPoolV1.pushTokens()` function which also update the `poolV1ds.totalOf` state to the current balance of the token in line 383 of `LibPoolV1.sol`.

AdminFacet.sol

```
423 function withdrawFeeReserve(
424   address token,
425   address to,
426   uint256 amount
427 ) external {
428   // Load diamond storage
429   LibPoolV1.PoolV1DiamondStorage storage ds = LibPoolV1
430     .poolV1DiamondStorage();
431
432   if (msg.sender != LibPoolConfigV1.treasury()) revert AdminFacet_Forbidden();
433
434   ds.feeReserveOf[token] -= amount;
435   LibPoolV1.pushTokens(token, to, amount);
436
437   emit WithdrawFeeReserve(token, to, amount);
438 }
```

LibPoolV1.sol

```
375 function pushTokens(
376   address token,
377   address to,
378   uint256 amount
379 ) internal {
380   PoolV1DiamondStorage storage poolV1ds = poolV1DiamondStorage();
381
382   IERC20(token).safeTransfer(to, amount);
383   poolV1ds.totalOf[token] = IERC20(token).balanceOf(address(this));
```

```
384 }
```

To exploit this issue, the attacker has to create a contract which has a call to the `flashLoan()` function and contains the `onFlashLoan()` callback function that calls to one of the functions mentioned above. After abusing the `poolV1ds.totalOf` state, the attacker only needs to repay fewer tokens (at least the flashloan fee) to pass the flashloan check.

5.1.2. Remediation

Inspex suggests implementing the check of the `poolV1ds.totalOf` state properly by using it both before the token is transferred to the borrower and after the token is returned to validate the loan repayment. For example:

Adding a call `LibPoolV1.updateTotalOf()` function to update the `poolV1ds.TotalOf` state and store it in the `totalOfBefore` variable in lines 479-480. Then calling `LibPoolV1.updateTotalOf()` function to update the `poolV1ds.TotalOf` state again and comparing the current `poolV1ds.TotalOf` state with `totalOfBefore` variable as shown in lines 493-497 instead.

LiquidityFacet.sol

```
463 function flashLoan(
464   FlashLoanBorrowerInterface borrower,
465   address[] calldata receivers,
466   address[] calldata tokens,
467   uint256[] calldata amounts,
468   bytes calldata data
469 ) external nonReentrant {
470   if (receivers.length != tokens.length || receivers.length != amounts.length)
471     revert LiquidityFacet_BadLength();
472
473   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
474     .poolV1DiamondStorage();
475
476   uint256[] memory totalOfBefore = new uint256[](tokens.length);
477   uint256[] memory fees = new uint256[](tokens.length);
478   for (uint256 i = 0; i < tokens.length; ) {
479     LibPoolV1.updateTotalOf(tokens[i]);
480     totalOfBefore[i] = poolV1ds.totalOf[tokens[i]];
481     fees[i] = (amounts[i] * LibPoolConfigV1.flashLoanFeeBps()) / BPS;
482
483     ERC20(tokens[i]).safeTransfer(receivers[i], amounts[i]);
484
485     unchecked {
486       ++i;
487     }
488   }
489
```

```
490   borrower.onFlashLoan(msg.sender, tokens, amounts, fees, data);
491
492   for (uint256 i = 0; i < tokens.length; ) {
493     LibPoolV1.updateTotalOf(tokens[i]); // must update for make sure that the
    TotalOf state is not abused
494     if (
495       poolV1ds.totalOf[tokens[i]] <=
496       totalOfBefore[i] + fees[i]
497     ) revert LiquidityFacet_BadFlashLoan();
498
499     // Collect fee
500     poolV1ds.feeReserveOf[tokens[i]] += fees[i];
501
502     emit FlashLoan(
503       address(borrower),
504       tokens[i],
505       amounts[i],
506       fees[i],
507       receivers[i]
508     );
509
510     unchecked {
511       ++i;
512     }
513   }
514 }
```

5.2. Insufficient Pool Address Check

ID	IDX-002
Target	PoolRouter
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity: Critical** **Impact: High** The attacker can abuse the `PoolRouter` contract flow, resulting in the drain of all approved tokens from the platform user who approved tokens for the `PoolRouter` contract. **Likelihood: High** It is very likely that the platform users will have the approved tokens for the `PoolRouter` contract, since it is required to interact with the platform.
Status	Resolved The Perp88 team has resolved this issue by implementing the pool address as the state instead of the function parameter for all affected functions.

5.2.1. Description

Every function in the `PoolRouter` contract gets the `pool` address as an input parameter. For example, in line 38 of the `addLiquidity()` function, if the malicious pool address is supplied to this function along with the `receiver` as a victim address, the victim's tokens will be transferred from the wallet to the `PoolRouter` contract with the `safeTransferFrom()` function call in line 55.

PoolRouter.sol

```
37 function addLiquidity(
38   address pool,
39   address token,
40   uint256 amount,
41   address receiver,
42   uint256 minLiquidity
43 ) external returns (uint256) {
44   IERC20(token).safeTransferFrom(msg.sender, address(pool), amount);
45
46   uint256 receivedAmount = LiquidityFacetInterface(pool).addLiquidity(
47     msg.sender,
48     token,
49     receiver
50   );
```

```
51
52    if (receivedAmount < minLiquidity)
53      revert PoolRouter_InsufficientOutputAmount(minLiquidity, receivedAmount);
54
55    IERC20(address(GetterFacetInterface(pool).plp())).safeTransferFrom(
56      receiver,
57      address(this),
58      receivedAmount
59    );
60
61    GetterFacetInterface(pool).plp().approve(
62      address(plpStaking),
63      receivedAmount
64    );
65
66    plpStaking.deposit(
67      receiver,
68      address(GetterFacetInterface(pool).plp()),
69      receivedAmount
70    );
71
72    return receivedAmount;
73  }
```

With this issue, the attacker can also withdraw the tokens in the contract by calling the `decreasePosition()` function with a malicious pool address. Any condition check will be passed by implementing the malicious contract for handling those conditions. Then the token will be transferred to the provided receiver address with the `safeTransfer()` function call at line 303.

PoolRouter.sol

```
261 function decreasePosition(
262   address pool,
263   uint256 subAccountId,
264   address collateralToken,
265   address indexToken,
266   uint256 collateralDelta,
267   uint256 sizeDelta,
268   bool isLong,
269   address receiver,
270   uint256 acceptablePrice,
271   address tokenOut,
272   uint256 minAmountOut
273 ) external {
274   PoolOracle oracle = PoolOracle(GetterFacetInterface(pool).oracle());
275   if (isLong) {
276     uint256 actualPrice = oracle.getMinPrice(indexToken);
```

```
277       if (!(actualPrice >= acceptablePrice))
278         revert PoolRouter_MarkPriceTooLow(acceptablePrice, actualPrice);
279     } else {
280       uint256 actualPrice = oracle.getMaxPrice(indexToken);
281       if (!(actualPrice <= acceptablePrice))
282         revert PoolRouter_MarkPriceTooHigh(acceptablePrice, actualPrice);
283     }
284
285     uint256 amountOutFromPosition = PerpTradeFacetInterface(pool)
286       .decreasePosition(
287         msg.sender,
288         subAccountId,
289         collateralToken,
290         indexToken,
291         collateralDelta,
292         sizeDelta,
293         isLong,
294         address(this)
295       );
296
297     if (collateralToken == tokenOut) {
298       if (amountOutFromPosition < minAmountOut)
299         revert PoolRouter_InsufficientOutputAmount(
300           minAmountOut,
301           amountOutFromPosition
302         );
303       IERC20(tokenOut).safeTransfer(receiver, amountOutFromPosition);
304     } else {
305       _swap(
306         pool,
307         address(this),
308         collateralToken,
309         tokenOut,
310         amountOutFromPosition,
311         minAmountOut,
312         receiver
313       );
314     }
315   }
```

5.2.2. Remediation

Inspex suggests implementing the `pool` address as the state instead of the function parameter for all affected functions by setting the pool address in the constructor once the contract is deployed.

PoolRouter.sol

```
30   address public immutable pool;
```

```
31
32 constructor(address wNative_, address plpStaking_, address pool_) {
33   WNATIVE = IWNative(wNative_);
34   plpStaking = IStaking(plpStaking_);
35   pool = pool_;
36 }
```

5.3. Lack of Position Healthy Check

ID	IDX-003
Target	PerpTradeFacet
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity: Critical** **Impact: High** Any position in the platform can be liquidated even in its healthy state. This completely broke the platform business. **Likelihood: High** Allowed liquidators or anyone (if `isAllowAllLiquidators` state is set to `true`) can liquidate any position in the platform and gain the liquidation fee.
Status	**Resolved** The Perp88 team has resolved this issue by adding a position state validation to the `liquidate()` function to prevent liquidation of a healthy position.

5.3.1. Description

In the `PerpTradeFacet` contract, the `checkLiquidation()` function returns the state of the position, which can be in 3 states (`HEALTHY`, `SOFT_LIQUIDATE`, `LIQUIDATE`).

PerpTradeFacet.sol

```
135 function checkLiquidation(
136   address account,
137   address collateralToken,
138   address indexToken,
139   bool isLong,
140   bool isRevertOnError
141 )
142   public
143   view
144   returns (
145     LiquidationState,
146     uint256,
147     uint256,
148     int256
149   )
150 {
151   // ... redacted checkLiquidation process ...
```

PerpTradeFacetInterface.sol

```
5  enum LiquidationState {
6    HEALTHY,
7    SOFT_LIQUIDATE,
8    LIQUIDATE
9  }
```

In the `liquidate()` function, the `vars.liquidationState` variable stores the state of the position, but there is only one condition check that uses the `vars.liquidationState` value to check whether the state is equal to `SOFT_LIQUIDATE` state or not, as shown in line 827. However, both `HEALTHY` and `LIQUIDATE` states are not checked. Then the position is treated as a `LIQUIDATE` state until the end of the function.

PerpTradeFacet.sol

```
773  function liquidate(
774    address primaryAccount,
775    uint256 subAccountId,
776    address collateralToken,
777    address indexToken,
778    bool isLong,
779    address to
780  ) external {
781    LiquidateLocalVars memory vars;
782    // Load diamond storage
783    LibPoolV1.PoolV1DiamondStorage storage ds = LibPoolV1
784      .poolV1DiamondStorage();
785
786    if (!LibPoolConfigV1.isAllowedLiquidators(msg.sender))
787      revert PerpTradeFacet_BadLiquidator();
788
789    // Realize profit/loss result from the farm strategy
790    LibPoolV1.realizedFarmPnL(collateralToken);
791    // If indexToken != collateralToken, need to realize indexToken as well
792    if (collateralToken != indexToken) LibPoolV1.realizedFarmPnL(indexToken);
793
794    FundingRateFacetInterface(address(this)).updateFundingRate(
795      collateralToken,
796      indexToken
797    );
798
799    vars.subAccount = GetterFacetInterface(address(this)).getSubAccount(
800      primaryAccount,
801      subAccountId
802    );
803
804    vars.posId = LibPoolV1.getPositionId(
805      vars.subAccount,
```

```
806     collateralToken,
807     indexToken,
808     isLong
809   );
810   LibPoolV1.Position memory position = ds.positions[vars.posId];
811
812   if (position.size == 0) revert PerpTradeFacet_BadPositionSize();
813
814   (
815     vars.liquidationState,
816     vars.borrowingFee,
817     vars.positionFee,
818     vars.fundingFee
819   ) = checkLiquidation(
820     vars.subAccount,
821     collateralToken,
822     indexToken,
823     isLong,
824     false
825   );
826   vars.marginFee = vars.borrowingFee + vars.positionFee;
827   if (vars.liquidationState == LiquidationState.SOFT_LIQUIDATE) {
828     // Position's leverage is exceeded, but there is enough collateral to
    soft-liquidate.
829     _decreasePosition(
830       primaryAccount,
831       subAccountId,
832       collateralToken,
833       indexToken,
834       0,
835       position.size,
836       isLong,
837       position.primaryAccount
838     );
839     return;
840   }
841
842   // ... redacted liquidate process ...
```

5.3.2. Remediation

Inspex suggests adding a position state validation to the `liquidate()` function to prevent liquidation of a healthy position as shown in line 826.

PerpTradeFacet.sol

```
773 function liquidate(
774   address primaryAccount,
```

```
775     uint256 subAccountId,
776     address collateralToken,
777     address indexToken,
778     bool isLong,
779     address to
780   ) external {
781     LiquidateLocalVars memory vars;
782     // Load diamond storage
783     LibPoolV1.PoolV1DiamondStorage storage ds = LibPoolV1
784       .poolV1DiamondStorage();
785
786     if (!LibPoolConfigV1.isAllowedLiquidators(msg.sender))
787       revert PerpTradeFacet_BadLiquidator();
788
789     // Realize profit/loss result from the farm strategy
790     LibPoolV1.realizedFarmPnL(collateralToken);
791     // If indexToken != collateralToken, need to realize indexToken as well
792     if (collateralToken != indexToken) LibPoolV1.realizedFarmPnL(indexToken);
793
794     FundingRateFacetInterface(address(this)).updateFundingRate(
795       collateralToken,
796       indexToken
797     );
798
799     vars.subAccount = GetterFacetInterface(address(this)).getSubAccount(
800       primaryAccount,
801       subAccountId
802     );
803
804     vars.posId = LibPoolV1.getPositionId(
805       vars.subAccount,
806       collateralToken,
807       indexToken,
808       isLong
809     );
810     LibPoolV1.Position memory position = ds.positions[vars.posId];
811
812     if (position.size == 0) revert PerpTradeFacet_BadPositionSize();
813
814     (
815       vars.liquidationState,
816       vars.borrowingFee,
817       vars.positionFee,
818       vars.fundingFee
819     ) = checkLiquidation(
820       vars.subAccount,
821       collateralToken,
```

```
822       indexToken,
823       isLong,
824       false
825     );
826     if (vars.liquidationState == LiquidationState.HEALTHY) revert
    PerpTradeFacet_BadPositionState();
827     vars.marginFee = vars.borrowingFee + vars.positionFee;
828     if (vars.liquidationState == LiquidationState.SOFT_LIQUIDATE) {
829       // Position's leverage is exceeded, but there is enough collateral to
    soft-liquidate.
830       _decreasePosition(
831         primaryAccount,
832         subAccountId,
833         collateralToken,
834         indexToken,
835         0,
836         position.size,
837         isLong,
838         position.primaryAccount
839       );
840       return;
841     }
842
843     // ... redacted liquidate process ...
```

5.4. Missing Update for totalOf state in Farm Function

ID	IDX-004
Target	FarmFacet
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity: Critical** **Impact: High** A mismatch between the actual token balance in the pool and `totalOf` state can cause miscalculations in a function that uses these values. An attacker could take advantage of state mismatch to pass the check between token balance and `totalOf`, which could impact platform and user funds. **Likelihood: High** The pool itself and the FarmKeeper role, can use the `farm()` function on the token that has a strategy to withdraw the token from the strategy.
Status	Resolved The Perp88 team has resolved this issue by using the `updateTotalOf` function to update the `totalOf` state.

5.4.1. Description

The `farm()` function is used for realizing profits or losses from the strategy at lines 139–159. If the strategy gains profit, it will withdraw the token and send it to the caller, which is the pool in this case. After the token transfer to the pool, the `totalOf` state did not update based on the withdrawn token amount.

This also happens when the strategy rebalances in lines 180–192: if `strategyData.principle` is greater than `targetDeployedFunds` the function will call `strategy.withdraw()` to transfer the token back to the pool.

FarmFacet.sol

```
121 function farm(address token, bool isRebalanceNeeded)
122     external
123     onlyPoolDiamondOrFarmKeeper
124 {
125   // Load PoolV1 diamond storage
126   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
127     .poolV1DiamondStorage();
128
129   // Load PoolConfig Diamond storage
130   LibPoolConfigV1.PoolConfigV1DiamondStorage
```

```
131       storage poolConfigDs = LibPoolConfigV1.poolConfigV1DiamondStorage();
132
133     // Load relevant variables
134     LibPoolConfigV1.StrategyData memory strategyData = poolConfigDs
135       .strategyDataOf[token];
136     StrategyInterface strategy = poolConfigDs.strategyOf[token];
137
138     // Realized profits or losses from strategy
139     int256 balanceChange = strategy.realized(
140       strategyData.principle,
141       msg.sender
142     );
143     // If there is no change in balance, and does not need to rebalance, then
    stop it here.
144     if (balanceChange == 0 && !isRebalanceNeeded) return;
145
146     if (balanceChange > 0) {
147       // If there is a profit, then increase pool liquidity
148       uint256 profits = uint256(balanceChange);
149       LibPoolV1.increasePoolLiquidity(token, profits);
150
151       emit StrategyRealizedProfit(token, profits);
152     } else if (balanceChange < 0) {
153       // If there is a loss, then decrease pool liquidity
154       uint256 losses = uint256(-balanceChange);
155       LibPoolV1.decreasePoolLiquidity(token, losses);
156       strategyData.principle -= losses.toUint128();
157
158       emit StrategyRealizedLoss(token, losses);
159     }
160
161     // If rebalance to make sure the strategy has the right amount of funds to
    deploy, then do it.
162     if (isRebalanceNeeded) {
163       // Calculate the target amount of funds to be deployed
164       uint256 targetDeployedFunds = ((poolV1ds.liquidityOf[token] -
165         poolV1ds.reservedOf[token]) * strategyData.targetBps) / 10000;
166
167       if (strategyData.principle < targetDeployedFunds) {
168         // If strategy short of funds, then deposit more funds
169         // Find out how much more funds to deposit
170         uint256 amountOut = targetDeployedFunds - strategyData.principle;
171
172         // Transfer funds from pool to strategy and run it
173         LibPoolV1.pushTokens(token, address(strategy), amountOut);
174         strategy.run(amountOut);
175
```

```
176       // Update how much pool put in the strategy
177       strategyData.principle += amountOut.toUint128();
178
179       emit StrategyInvest(token, amountOut);
180     } else if (strategyData.principle > targetDeployedFunds) {
181       // If strategy has more funds than it should be, then withdraw some funds
182       // Find out how much funds to withdraw
183       uint256 amountIn = strategyData.principle - targetDeployedFunds;
184
185       // Withdraw funds from strategy and transfer it back to pool
186       uint256 actualAmountIn = strategy.withdraw(amountIn);
187
188       // Update how much pool put in the strategy
189       strategyData.principle -= actualAmountIn.toUint128();
190
191       emit StrategyDivest(token, actualAmountIn);
192     }
193   }
194
195   poolConfigDs.strategyDataOf[token] = strategyData;
196 }
```

An attacker can use the un-updated `totalOf` state to pass the check in some functions.

For example, in the `flashLoan()` function from `LiquidityFacet` contract, in line 491, the function needs a token balance in the pool greater than or equal to `totalOf` state including the fee. Since the `totalOf` state did not update after the pool withdrew the token from strategy, the balance in this pool will be greater than the `totalOf` state and the attacker can repay with the amount that passes the check and keep the rest as profits.

LiquidityFacet.sol

```
463 function flashLoan(
464   FlashLoanBorrowerInterface borrower,
465   address[] calldata receivers,
466   address[] calldata tokens,
467   uint256[] calldata amounts,
468   bytes calldata data
469 ) external nonReentrant {
470   if (receivers.length != tokens.length || receivers.length != amounts.length)
471     revert LiquidityFacet_BadLength();
472
473   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
474     .poolV1DiamondStorage();
475
476   uint256[] memory fees = new uint256[](tokens.length);
477   for (uint256 i = 0; i < tokens.length; ) {
```

```
478       fees[i] = (amounts[i] * LibPoolConfigV1.flashLoanFeeBps()) / BPS;
479
480       ERC20(tokens[i]).safeTransfer(receivers[i], amounts[i]);
481
482       unchecked {
483         ++i;
484       }
485     }
486
487     borrower.onFlashLoan(msg.sender, tokens, amounts, fees, data);
488
489     for (uint256 i = 0; i < tokens.length; ) {
490       if (
491         ERC20(tokens[i]).balanceOf(address(this)) <
492         poolV1ds.totalOf[tokens[i]] + fees[i]
493       ) revert LiquidityFacet_BadFlashLoan();
494
495       // Collect fee
496       poolV1ds.feeReserveOf[tokens[i]] += fees[i];
497
498       emit FlashLoan(
499         address(borrower),
500         tokens[i],
501         amounts[i],
502         fees[i],
503         receivers[i]
504       );
505
506       unchecked {
507         ++i;
508       }
509     }
510 }
```

5.4.2. Remediation

Inspex suggests updating the `totalOf` state to match the current balance of the tokens in the pool when a token is withdrawn from strategy at lines 151 and 193.

FarmFacet.sol

```
121 function farm(address token, bool isRebalanceNeeded)
122     external
123     onlyPoolDiamondOrFarmKeeper
124 {
125   // Load PoolV1 diamond storage
126   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
127     .poolV1DiamondStorage();
```

```
128
129   // Load PoolConfig Diamond storage
130   LibPoolConfigV1.PoolConfigV1DiamondStorage
131     storage poolConfigDs = LibPoolConfigV1.poolConfigV1DiamondStorage();
132
133   // Load relevant variables
134   LibPoolConfigV1.StrategyData memory strategyData = poolConfigDs
135     .strategyDataOf[token];
136   StrategyInterface strategy = poolConfigDs.strategyOf[token];
137
138   // Realized profits or losses from strategy
139   int256 balanceChange = strategy.realized(
140     strategyData.principle,
141     msg.sender
142   );
143   // If there is no change in balance, and does not need to rebalance, then
    stop it here.
144   if (balanceChange == 0 && !isRebalanceNeeded) return;
145
146   if (balanceChange > 0) {
147     // If there is a profit, then increase pool liquidity
148     uint256 profits = uint256(balanceChange);
149     LibPoolV1.increasePoolLiquidity(token, profits);
150
151     LibPoolV1.updateTotalOf(token);
152
153     emit StrategyRealizedProfit(token, profits);
154   } else if (balanceChange < 0) {
155     // If there is a loss, then decrease pool liquidity
156     uint256 losses = uint256(-balanceChange);
157     LibPoolV1.decreasePoolLiquidity(token, losses);
158     strategyData.principle -= losses.toUint128();
159
160     emit StrategyRealizedLoss(token, losses);
161 }
162
163   // If rebalance to make sure the strategy has the right amount of funds to
    deploy, then do it.
164   if (isRebalanceNeeded) {
165     // Calculate the target amount of funds to be deployed
166     uint256 targetDeployedFunds = ((poolV1ds.liquidityOf[token] -
167       poolV1ds.reservedOf[token]) * strategyData.targetBps) / 10000;
168
169     if (strategyData.principle < targetDeployedFunds) {
170       // If strategy short of funds, then deposit more funds
171       // Find out how much more funds to deposit
172       uint256 amountOut = targetDeployedFunds - strategyData.principle;
```

```
173
174       // Transfer funds from pool to strategy and run it
175       LibPoolV1.pushTokens(token, address(strategy), amountOut);
176       strategy.run(amountOut);
177
178       // Update how much pool put in the strategy
179       strategyData.principle += amountOut.toUint128();
180
181       emit StrategyInvest(token, amountOut);
182     } else if (strategyData.principle > targetDeployedFunds) {
183       // If strategy has more funds than it should be, then withdraw some funds
184       // Find out how much funds to withdraw
185       uint256 amountIn = strategyData.principle - targetDeployedFunds;
186
187       // Withdraw funds from strategy and transfer it back to pool
188       uint256 actualAmountIn = strategy.withdraw(amountIn);
189
190       // Update how much pool put in the strategy
191       strategyData.principle -= actualAmountIn.toUint128();
192
193       LibPoolV1.updateTotalOf(token);
194
195       emit StrategyDivest(token, actualAmountIn);
196     }
197   }
198
199   poolConfigDs.strategyDataOf[token] = strategyData;
200 }
```

5.5. Missing Update for lastFundingTimeOf of Collateral Token

ID	IDX-005
Target	FundingRateFacet
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity: Critical** **Impact: High** The attacker can add countless `borrowingRate` to `poolV1ds.sumBorrowingRateOf[collateralToken]` state, resulting in an exceedingly high borrowing fee collection. An increase in the borrowing fee also leads to the liquidation of any position on the platform, as an example. **Likelihood: High** It is very likely to happen since the attacker could gain an advantage by repeatedly calling the `updateFundingRate()` function, and this function can be called by anyone. For example, liquidating all positions to get the profit.
Status	Resolved The Perp88 team has resolved this issue as suggested by updating both the borrowing rate and funding rate simultaneously.

5.5.1. Description

In the `FundingRateFacet` contract, the `poolV1ds.lastFundingTimeOf[collateralToken]` is not updated in the `updateFundingRate()` function, which allows the attacker to repeatedly call this function for adding `borrowingRate` to `poolV1ds.sumBorrowingRateOf[collateralToken]` countlessly in line 47.

FundingRateFacet.sol

```
18 function updateFundingRate(address collateralToken, address indexToken)
19   external
20 {
21   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
22     .poolV1DiamondStorage();
23
24   uint256 fundingInterval = LibPoolConfigV1.fundingInterval();
25
26   // If this is the first time that the funding and borrowing rate are accrued,
27   // set the initial funding time here
28   if (poolV1ds.lastFundingTimeOf[collateralToken] == 0) {
29     poolV1ds.lastFundingTimeOf[collateralToken] =
30       (block.timestamp / fundingInterval) *
31       fundingInterval;
```

```
32    }
33    if (poolV1ds.lastFundingTimeOf[indexToken] == 0) {
34      poolV1ds.lastFundingTimeOf[indexToken] =
35        (block.timestamp / fundingInterval) *
36        fundingInterval;
37    }
38
39    // If block.timestamp is not passed the next funding interval, skip updating
      borrowing rate.
40    if (
41      poolV1ds.lastFundingTimeOf[collateralToken] + fundingInterval <=
42      block.timestamp
43    ) {
44      uint256 borrowingRate = GetterFacetInterface(address(this))
45        .getNextBorrowingRate(collateralToken);
46      unchecked {
47        poolV1ds.sumBorrowingRateOf[collateralToken] += borrowingRate;
48      }
49
50      emit UpdateBorrowingRate(
51        collateralToken,
52        poolV1ds.sumBorrowingRateOf[collateralToken]
53      );
54    }
55
56    // If block.timestamp is not passed the next funding interval, skip updating
      funding rate
57    if (
58      poolV1ds.lastFundingTimeOf[indexToken] + fundingInterval <=
59      block.timestamp
60    ) {
61      (int256 fundingRateLong, int256 fundingRateShort) = GetterFacetInterface(
62        address(this)
63      ).getNextFundingRate(indexToken);
64      unchecked {
65        poolV1ds.accumFundingRateLong[indexToken] += fundingRateLong;
66        poolV1ds.accumFundingRateShort[indexToken] += fundingRateShort;
67        poolV1ds.lastFundingTimeOf[indexToken] =
68          (block.timestamp / fundingInterval) *
69          fundingInterval;
70      }
71
72      emit UpdateFundingRate(
73        indexToken,
74        poolV1ds.accumFundingRateLong[indexToken],
75        poolV1ds.accumFundingRateShort[indexToken]
76      );
```

```
77    }
78  }
```

In the `GetterFacet` contract, at line 382, it gets the `borrowingRate` from the `getBorrowingFee` function. The `borrowingRate` may continue to rise indefinitely since the `ds.sumBorrowingRateOf[collateralToken]` of `FundingRateFacet.sol` can be increased countless times.

GetterFacet.sol

```
368 function getBorrowingFee(
369   address, /* account */
370   address collateralToken,
371   address, /* indexToken */
372   bool, /* isLong */
373   uint256 size,
374   uint256 entryBorrowingRate
375 ) public view returns (uint256) {
376   // Load diamond storage
377   LibPoolV1.PoolV1DiamondStorage storage ds = LibPoolV1
378     .poolV1DiamondStorage();
379
380   if (size == 0) return 0;
381
382   uint256 borrowingRate = ds.sumBorrowingRateOf[collateralToken] -
383     entryBorrowingRate;
384   if (borrowingRate == 0) return 0;
385
386   return (size * borrowingRate) / FUNDING_RATE_PRECISION;
387 }
```

In the `PerptradeFacet` contract, the `_collectMarginFee()` function, which is normally used to collect and store platform fees, has a call to the `GetterFacetInterface.getBorrowingFee()` function at line 314. If this issue has been abused, the abnormally high `borrowingFeeUsd` value will be converted and stored in the `ds.feeReserveOf` state. Moreover, the abnormal return value will also be used in other functions and break their logic.

PerptradeFacet.sol

```
286 function _collectMarginFee(
287   address primaryAccount,
288   address account,
289   address collateralToken,
290   address indexToken,
291   bool isLong,
292   uint256 sizeDelta,
293   uint256 size,
```

```
294     uint256 entryBorrowingRate
295   ) internal returns (uint256) {
296     // Load diamond storage
297     LibPoolV1.PoolV1DiamondStorage storage ds = LibPoolV1
298       .poolV1DiamondStorage();
299
300     uint256 feeUsd = GetterFacetInterface(address(this)).getPositionFee(
301       account,
302       collateralToken,
303       indexToken,
304       isLong,
305       sizeDelta
306     );
307     emit CollectPositionFee(
308       primaryAccount,
309       collateralToken,
310       feeUsd,
311       LibPoolV1.convertUsde30ToTokens(collateralToken, feeUsd, true)
312     );
313
314     uint256 borrowingFeeUsd = GetterFacetInterface(address(this))
315       .getBorrowingFee(
316         account,
317         collateralToken,
318         indexToken,
319         isLong,
320         size,
321         entryBorrowingRate
322       );
323
324     emit CollectBorrowingFee(
325       primaryAccount,
326       collateralToken,
327       borrowingFeeUsd,
328       LibPoolV1.convertUsde30ToTokens(collateralToken, borrowingFeeUsd, true)
329     );
330
331     feeUsd += borrowingFeeUsd;
332
333     uint256 feeTokens = LibPoolV1.convertUsde30ToTokens(
334       collateralToken,
335       feeUsd,
336       true
337     );
338     ds.feeReserveOf[collateralToken] += feeTokens;
339
340     return feeUsd;
```

```
341 }
```

For example, a liquidation flow, the `checkLiquidation()` function checks the position state by calling the `GetterFacetInterface.getBorrowingFee()` function, which will store the `borrowingRate` value into the `vars.borrowingFee` at line 173.

At line 188, the `vars.marginFee` is the sum of the `vars.borrowingFee` and `vars.positionFee`, the higher `vars.borrowingFee`, the higher `vars.marginFee`.

Subsequently, `vars.marginFee` value is used for calculating the position state whether it can be liquidated or not at lines 205 and 217.

PerptradeFacet.sol

```
135 function checkLiquidation(
136   address account,
137   address collateralToken,
138   address indexToken,
139   bool isLong,
140   bool isRevertOnError
141 )
142   public
143   view
144   returns (
145     LiquidationState,
146     uint256,
147     uint256,
148     int256
149 )
150 {
151   CheckLiquidationLocalVars memory vars;
152
153   // Load diamond storage
154   LibPoolV1.PoolV1DiamondStorage storage ds = LibPoolV1
155     .poolV1DiamondStorage();
156
157   LibPoolV1.Position memory position = ds.positions[
158     LibPoolV1.getPositionId(account, collateralToken, indexToken, isLong)
159   ];
160
161   // Negative fundingFee means profits to the position
162   (vars.isProfit, vars.delta, vars.fundingFee) = GetterFacetInterface(
163     address(this)
164   ).getDelta(
165       indexToken,
166       position.size,
167       position.averagePrice,
```

```
168       isLong,
169       position.lastIncreasedTime,
170       position.entryFundingRate,
171       position.fundingFeeDebt
172     );
173   vars.borrowingFee = GetterFacetInterface(address(this)).getBorrowingFee(
174     account,
175     collateralToken,
176     indexToken,
177     isLong,
178     position.size,
179     position.entryBorrowingRate
180   );
181   vars.positionFee = GetterFacetInterface(address(this)).getPositionFee(
182     account,
183     collateralToken,
184     indexToken,
185     isLong,
186     position.size
187   );
188   vars.marginFee = vars.borrowingFee + vars.positionFee;
189
190   if (!vars.isProfit && position.collateral < vars.delta) {
191     if (isRevertOnError) revert PerpTradeFacet_LossesExceedCollateral();
192     return (
193       LiquidationState.LIQUIDATE,
194       vars.borrowingFee,
195       vars.positionFee,
196       vars.fundingFee
197     );
198   }
199
200   uint256 remainingCollateral = position.collateral;
201   if (!vars.isProfit) {
202     remainingCollateral -= vars.delta;
203   }
204
205   if (remainingCollateral < vars.marginFee) {
206     if (isRevertOnError) revert PerpTradeFacet_FeeExceedCollateral();
207     // Cap the fee to the remainingCollateral.
208     return (
209       LiquidationState.LIQUIDATE,
210       0,
211       remainingCollateral,
212       vars.fundingFee
213     );
214   }
```

```
215
216   if (
217     remainingCollateral < vars.marginFee + LibPoolConfigV1.liquidationFeeUsd()
218   ) {
219     if (isRevertOnError)
220       revert PerpTradeFacet_LiquidationFeeExceedCollateral();
221     // Cap the fee to the margin fee
222     return (
223       LiquidationState.LIQUIDATE,
224       vars.borrowingFee,
225       vars.positionFee,
226       vars.fundingFee
227     );
228   }
229
230   if (
231     remainingCollateral * LibPoolConfigV1.maxLeverage() < position.size * BPS
232   ) {
233     if (isRevertOnError) revert PerpTradeFacet_MaxLeverageExceed();
234     return (
235       LiquidationState.SOFT_LIQUIDATE,
236       vars.borrowingFee,
237       vars.positionFee,
238       vars.fundingFee
239     );
240   }
241
242   return (
243     LiquidationState.HEALTHY,
244     vars.borrowingFee,
245     vars.positionFee,
246     vars.fundingFee
247   );
248 }
```

As a result, the attacker can repeatedly execute the `updateFundingRate()` function to increase the `borrowingRate` state, then execute the `liquidate()` function to liquidate all positions in the pool.

5.5.2. Remediation

Inspex suggests updating both borrowing rate and funding rate simultaneously when the condition is met. For example implementing the internal `updateBorrowingRateAndFundingRate()` function for handling the borrowing rate and funding rate updating.

FundingRateFacet.sol

```
18 function updateFundingRate(address collateralToken, address indexToken)
19   external
```

```
20  {
21    updateBorrowingRateAndFundingRate(collateralToken);
22    if (collateralToken != indexToken)
23      updateBorrowingRateAndFundingRate(indexToken);
24  }
25
26  function updateBorrowingRateAndFundingRate(address token) internal {
27    LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
28      .poolV1DiamondStorage();
29
30    uint256 fundingInterval = LibPoolConfigV1.fundingInterval();
31
32    // If this is the first time that the funding and borrowing rate are accrued,
33    // set the initial funding time here
34    if (poolV1ds.lastFundingTimeOf[token] == 0) {
35      poolV1ds.lastFundingTimeOf[token] =
36        (block.timestamp / fundingInterval) *
37        fundingInterval;
38      return;
39    }
40
41    // If block.timestamp is not passed the next funding interval, skip updating
42    if (
43      poolV1ds.lastFundingTimeOf[token] + fundingInterval <=
44      block.timestamp
45    ) {
46      //update borrowing rate
47      uint256 borrowingRate = GetterFacetInterface(address(this))
48        .getNextBorrowingRate(token);
49      unchecked {
50        poolV1ds.sumBorrowingRateOf[token] += borrowingRate;
51      }
52
53      emit UpdateBorrowingRate(
54        token,
55        poolV1ds.sumBorrowingRateOf[token]
56      );
57
58      // update funding rate
59      (int256 fundingRateLong, int256 fundingRateShort) = GetterFacetInterface(
60        address(this)
61      ).getNextFundingRate(token);
62      unchecked {
63        poolV1ds.accumFundingRateLong[token] += fundingRateLong;
64        poolV1ds.accumFundingRateShort[token] += fundingRateShort;
65        poolV1ds.lastFundingTimeOf[token] =
66          (block.timestamp / fundingInterval) *
```

```
67            fundingInterval;
68        }
69
70        emit UpdateFundingRate(
71          token,
72          poolV1ds.accumFundingRateLong[token],
73          poolV1ds.accumFundingRateShort[token]
74        );
75      }
76    }
```

5.6. Use of Upgradable Contract Design

ID	IDX-006
Target	PoolOracle Orderbook PoolDiamond AccessControlFacet AdminFacet DiamondCutFacet DiamondLoupeFacet FarmFacet FundingRateFacet GetterFacet LiquidityFacet OwnershipFacet PerpTradeFacet TransparentUpgradeableProxy AdHocMintRewarder Compounder FeedableRewarder PLPStaking RewardDistributor WFeedableRewarder PLP Vester
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The logic of affected contracts can be arbitrarily changed. This allows the proxy owner to perform malicious actions e.g., stealing the users' funds anytime they want. **Likelihood: Medium** This action can be performed by the proxy owner without any restriction.
Status	Resolved * The Perp88 team has mitigated this issue by implementing a Timelock contract as the owner of all contracts to prevent immediate changes or upgrades to the contract and also to provide transparency in the process of maintaining contract upgrades. However, the timelock mechanism was not in use at the time of the reassessment. Therefore, Inspex suggests the platform users to confirm the usage of the timelock mechanism before using the platform.

5.6.1. Description

Smart contracts are designed to be used as agreements that cannot be changed forever. When a smart contract is upgraded, the agreement can be changed from what was previously agreed upon.

As these smart contracts are upgradable, the logic of them can be modified by the owner anytime, making the smart contracts untrustworthy.

5.6.2. Remediation

Inspex suggests deploying the contracts without the proxy pattern or any solution that can make the smart contracts upgradeable.

However, if upgradability is needed, Inspex suggests mitigating this issue by implementing a Timelock mechanism with a sufficient length of time to delay the changes e.g., 24 hours on the proxy owner role. This allows the platform users to monitor the timelock and be notified of the potential changes being done on the smart contracts.

5.7. Improper Profit Transfer

ID	IDX-007
Target	FarmFacet
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** High **Impact:** High The profit will be sent to the Farmkeeper rather than the pool when the Farmkeeper uses the `farm()` function to rebalance. **Likelihood:** Medium The pool itself or the Farmkeeper, which the contract owner can set, are the only entities that are permitted to call the `farm()` functions directly.
Status	Resolved The Perp88 team has resolved this issue by removing the sender argument from the `realized()` function on the `StrategyInterface` contract and the `MockStrategy` contract, and then using the `msg.sender` variable when transferring the profit token to the pool instead.

5.7.1. Description

The `farm()` function in the `FarmFacet` contract is used to realize profits or losses from strategy. The `farm()` function has the `onlyPoolDiamondOrFarmKeeper` modifier, which makes it callable by the pool itself and the farmkeeper only.

In line 139, to calculate the `balanceChange` value, the `farm()` function calls the `realize()` function on the `MockStrategy` contract. In the case that the farmkeeper calls the `farm()` function, the `msg.sender` argument of the `realize()` function will be the Farmkeeper's address.

FarmFacet.sol

```
121 function farm(address token, bool isRebalanceNeeded)
122   external
123   onlyPoolDiamondOrFarmKeeper
124 {
125   // Load PoolV1 diamond storage
126   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
127     .poolV1DiamondStorage();
128
129   // Load PoolConfig Diamond storage
130   LibPoolConfigV1.PoolConfigV1DiamondStorage
```

```
131     storage poolConfigDs = LibPoolConfigV1.poolConfigV1DiamondStorage();
132
133   // Load relevant variables
134   LibPoolConfigV1.StrategyData memory strategyData = poolConfigDs
135     .strategyDataOf[token];
136   StrategyInterface strategy = poolConfigDs.strategyOf[token];
137
138   // Realized profits or losses from strategy
139   int256 balanceChange = strategy.realized(
140     strategyData.principle,
141     msg.sender
142   );
143   // If there is no change in balance, and does not need to rebalance, then
    stop it here.
144   if (balanceChange == 0 && !isRebalanceNeeded) return;
145
146   if (balanceChange > 0) {
147     // If there is a profit, then increase pool liquidity
148     uint256 profits = uint256(balanceChange);
149     LibPoolV1.increasePoolLiquidity(token, profits);
150
151     emit StrategyRealizedProfit(token, profits);
152   } else if (balanceChange < 0) {
153     // If there is a loss, then decrease pool liquidity
154     uint256 losses = uint256(-balanceChange);
155     LibPoolV1.decreasePoolLiquidity(token, losses);
156     strategyData.principle -= losses.toUint128();
157
158     emit StrategyRealizedLoss(token, losses);
159   }
160
161   // If rebalance to make sure the strategy has the right amount of funds to
    deploy, then do it.
162   if (isRebalanceNeeded) {
163     // Calculate the target amount of funds to be deployed
164     uint256 targetDeployedFunds = ((poolV1ds.liquidityOf[token] -
165       poolV1ds.reservedOf[token]) * strategyData.targetBps) / 10000;
166
167     if (strategyData.principle < targetDeployedFunds) {
168       // If strategy short of funds, then deposit more funds
169       // Find out how much more funds to deposit
170       uint256 amountOut = targetDeployedFunds - strategyData.principle;
171
172       // Transfer funds from pool to strategy and run it
173       LibPoolV1.pushTokens(token, address(strategy), amountOut);
174       strategy.run(amountOut);
175
```

```
176       // Update how much pool put in the strategy
177       strategyData.principle += amountOut.toUint128();
178
179       emit StrategyInvest(token, amountOut);
180     } else if (strategyData.principle > targetDeployedFunds) {
181       // If strategy has more funds than it should be, then withdraw some funds
182       // Find out how much funds to withdraw
183       uint256 amountIn = strategyData.principle - targetDeployedFunds;
184
185       // Withdraw funds from strategy and transfer it back to pool
186       uint256 actualAmountIn = strategy.withdraw(amountIn);
187
188       // Update how much pool put in the strategy
189       strategyData.principle -= actualAmountIn.toUint128();
190
191       emit StrategyDivest(token, actualAmountIn);
192     }
193   }
194
195   poolConfigDs.strategyDataOf[token] = strategyData;
196 }
```

In line 61, the `realized()` function receives the `msg.sender` argument, which is the Farmkeeper's address, and transfers the tokens to Farmkeeper's address instead of the pool address if the strategy is in profit.

MockStrategy.sol

```
52 function realized(uint256 principle, address sender)
53   external
54   onlyPool
55   returns (int256 amountDelta)
56 {
57   (bool isProfit, uint256 amount) = getStrategyDelta(principle);
58   if (isProfit) {
59     vault.withdraw(_roundedValueToShare(amount));
60     uint256 balance = IERC20(token).balanceOf(address(this));
61     IERC20(token).transfer(sender, balance);
62     return int256(balance);
63   } else {
64     return -int256(amount);
65   }
66 }
```

5.7.2. Remediation

Inspex suggests sending the `address(this)` as an argument rather than `msg.sender` for the `realize()` function in line 141 to prevent the strategy contract from transferring the tokens to the farmkeeper's address when the farmkeeper calls the `farm()` function in the `FarmFacet` contract.

FarmFacet.sol

```
121 function farm(address token, bool isRebalanceNeeded)
122   external
123   onlyPoolDiamondOrFarmKeeper
124 {
125   // Load PoolV1 diamond storage
126   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
127     .poolV1DiamondStorage();
128
129   // Load PoolConfig Diamond storage
130   LibPoolConfigV1.PoolConfigV1DiamondStorage
131     storage poolConfigDs = LibPoolConfigV1.poolConfigV1DiamondStorage();
132
133   // Load relevant variables
134   LibPoolConfigV1.StrategyData memory strategyData = poolConfigDs
135     .strategyDataOf[token];
136   StrategyInterface strategy = poolConfigDs.strategyOf[token];
137
138   // Realized profits or losses from strategy
139   int256 balanceChange = strategy.realized(
140     strategyData.principle,
141     address(this)
142   );
143   // If there is no change in balance, and does not need to rebalance, then
    stop it here.
144   if (balanceChange == 0 && !isRebalanceNeeded) return;
145
146   if (balanceChange > 0) {
147     // If there is a profit, then increase pool liquidity
148     uint256 profits = uint256(balanceChange);
149     LibPoolV1.increasePoolLiquidity(token, profits);
150
151     emit StrategyRealizedProfit(token, profits);
152   } else if (balanceChange < 0) {
153     // If there is a loss, then decrease pool liquidity
154     uint256 losses = uint256(-balanceChange);
155     LibPoolV1.decreasePoolLiquidity(token, losses);
156     strategyData.principle -= losses.toUint128();
157
158     emit StrategyRealizedLoss(token, losses);
159   }
160
161   // If rebalance to make sure the strategy has the right amount of funds to
    deploy, then do it.
162   if (isRebalanceNeeded) {
163     // Calculate the target amount of funds to be deployed
164     uint256 targetDeployedFunds = ((poolV1ds.liquidityOf[token] -
```

```
165       poolV1ds.reservedOf[token]) * strategyData.targetBps) / 10000;
166
167     if (strategyData.principle < targetDeployedFunds) {
168       // If strategy short of funds, then deposit more funds
169       // Find out how much more funds to deposit
170       uint256 amountOut = targetDeployedFunds - strategyData.principle;
171
172       // Transfer funds from pool to strategy and run it
173       LibPoolV1.pushTokens(token, address(strategy), amountOut);
174       strategy.run(amountOut);
175
176       // Update how much pool put in the strategy
177       strategyData.principle += amountOut.toUint128();
178
179       emit StrategyInvest(token, amountOut);
180     } else if (strategyData.principle > targetDeployedFunds) {
181       // If strategy has more funds than it should be, then withdraw some funds
182       // Find out how much funds to withdraw
183       uint256 amountIn = strategyData.principle - targetDeployedFunds;
184
185       // Withdraw funds from strategy and transfer it back to pool
186       uint256 actualAmountIn = strategy.withdraw(amountIn);
187
188       // Update how much pool put in the strategy
189       strategyData.principle -= actualAmountIn.toUint128();
190
191       emit StrategyDivest(token, actualAmountIn);
192     }
193   }
194
195   poolConfigDs.strategyDataOf[token] = strategyData;
196 }
```

5.8. Centralized Control of State Variable

ID	IDX-008
Target	PoolOracle Orderbook AccessControlFacet AdminFacet FarmFacet OwnershipFacet TransparentUpgradeableProxy Compounder FeedableRewarder PLPStaking RewardDistributor WFeedableRewarder PLP
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. **Likelihood: Medium** There is nothing to restrict the changes from being done; however, this action can only be done by the privileged roles.
Status	Resolved * The Perp88 team has mitigated this issue by implementing a Timelock contract as the owner of all contracts to prevent immediate changes or upgrades to the contract and also to provide transparency in the process of maintaining contract upgrades. However, the timelock mechanism was not in use at the time of the reassessment. Therefore, Inspex suggests the platform users to confirm the usage of the timelock mechanism before using the platform.

5.8.1. Description

Critical state variables can be updated at any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

However, there is currently no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

File	Contract	Function	Modifier/role
PoolOracle.sol (L: 131)	PoolOracle	setMaxStrictPriceDeviation()	onlyOwner
PoolOracle.sol (L: 142)	PoolOracle	setPriceFeed()	onlyOwner
PoolOracle.sol (L: 162)	PoolOracle	setRoundDepth()	onlyOwner
Orderbook.sol (L: 259)	Orderbook	setWhitelist()	onlyOwner
Orderbook.sol (L: 267)	Orderbook	setIsAllowAllExecutor()	onlyOwner
Orderbook.sol (L: 272)	Orderbook	setMinExecutionFee()	onlyOwner
Orderbook.sol (L: 278)	Orderbook	setMinPurchaseTokenAmountUsd()	onlyOwner
AccessControlFacet.sol (L: 43)	AccessControlFacet	grantRole()	adminRole
AccessControlFacet.sol (L: 59)	AccessControlFacet	revokeRole()	adminRole
AdminFacet.sol (L: 118)	AdminFacet	setPoolOracle()	onlyOwner
AdminFacet.sol (L: 130)	AdminFacet	setAllowLiquidators()	onlyOwner
AdminFacet.sol (L: 144)	AdminFacet	setFlashLoanFeeBps()	onlyOwner
AdminFacet.sol (L: 156)	AdminFacet	setFundingRate()	onlyOwner
AdminFacet.sol (L: 191)	AdminFacet	setIsAllowAllLiquidators()	onlyOwner
AdminFacet.sol (L: 206)	AdminFacet	setIsDynamicFeeEnable()	onlyOwner
AdminFacet.sol (L: 221)	AdminFacet	setIsLeverageEnable()	onlyOwner
AdminFacet.sol (L: 233)	AdminFacet	setIsSwapEnable()	onlyOwner
AdminFacet.sol (L: 242)	AdminFacet	setLiquidationFeeUsd()	onlyOwner
AdminFacet.sol (L: 260)	AdminFacet	setMaxLeverage()	onlyOwner
AdminFacet.sol (L: 271)	AdminFacet	setMinProfitDuration()	onlyOwner
AdminFacet.sol (L: 286)	AdminFacet	setMintBurnFeeBps()	onlyOwner
AdminFacet.sol (L: 298)	AdminFacet	setPositionFeeBps()	onlyOwner

AdminFacet.sol (L: 310)	AdminFacet	setRouter()	onlyOwner
AdminFacet.sol (L: 319)	AdminFacet	setSwapFeeBps()	onlyOwner
AdminFacet.sol (L: 341)	AdminFacet	setTaxBps()	onlyOwner
AdminFacet.sol (L: 359)	AdminFacet	setTokenConfigs()	onlyOwner
AdminFacet.sol (L: 396)	AdminFacet	setTreasury()	onlyOwner
AdminFacet.sol (L: 405)	AdminFacet	deleteTokenConfig()	onlyOwner
AdminFacet.sol (L: 440)	AdminFacet	setPlugin()	onlyOwner
FarmFacet.sol (L: 54)	FarmFacet	setStrategyOf()	onlyOwner
FarmFacet.sol (L: 108)	FarmFacet	setStrategyTargetBps()	onlyOwner
OwnershipFacet.sol (L: 8)	OwnershipFacet	transferOwnership()	contractOwner
TransparentUpgradeableProxy.sol (L: 86)	TransparentUpgradeableProxy	changeAdmin()	ifAdmin
TransparentUpgradeableProxy.sol (L: 95)	TransparentUpgradeableProxy	upgradeTo()	ifAdmin
TransparentUpgradeableProxy.sol (L: 86)	TransparentUpgradeableProxy	upgradeToAndCall()	ifAdmin
TransparentUpgradeableProxy.sol (L: 86)	TransparentUpgradeableProxy	changeAdmin()	ifAdmin
Compounder.sol (L: 33)	Compounder	addToken()	onlyOwner
Compounder.sol (L: 51)	Compounder	removeToken()	onlyOwner
Compounder.sol (L: 69)	Compounder	setCompoundToken()	onlyOwner
FeedableRewarder.sol (L: 137)	FeedableRewarder	feed()	onlyFeeder
FeedableRewarder.sol (L: 141)	FeedableRewarder	feedWithExpiredAt()	onlyFeeder
FeedableRewarder.sol (L: 148)	FeedableRewarder	setFeeder()	onlyOwner
PLPStaking.sol (L: 39)	PLPStaking	addStakingToken()	onlyOwner
PLPStaking.sol (L: 53)	PLPStaking	addRewarder()	onlyOwner

PLPStaking.sol (L: 76)	PLPStaking	setCompounder()	onlyOwner
RewardDistributor.sol (L: 69)	RewardDistributor	setParams()	onlyOwner
WFeedableRewarder.sol (L: 143)	WFeedableRewarder	feed()	onlyFeeder
WFeedableRewarder.sol (L: 147)	WFeedableRewarder	feedWithExpiredAt()	onlyFeeder
WFeedableRewarder.sol (L: 154)	WFeedableRewarder	setFeeder()	onlyOwner
PLP.sol (L: 34)	PLP	setLiquidityCooldown()	onlyOwner
PLP.sol (L: 45)	PLP	setWhitelist()	onlyOwner
PLP.sol (L: 51)	PLP	setMinter()	onlyOwner

5.8.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspex suggests implementing a community-run smart contract governance to control the use of these functions.

If removing the functions or implementing the smart contract governance is not possible, Inspex suggests mitigating the risk of this issue by using a timelock mechanism to delay the changes for a reasonable amount of time e.g., 24 hours.

5.9. Arbitrary PLP Token Minting

ID	IDX-009
Target	PLP
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The contract owner can mint unlimited PLP token by adding an arbitrary minter address to the contract. The minted PLP token can be used to remove liquidity from the pool. **Likelihood: Medium** Only the contract owner can perform this action.
Status	Resolved * The Perp88 team has mitigated this issue by implementing a Timelock contract as the owner of all contracts to prevent immediate changes or upgrades to the contract and also to provide transparency in the process of maintaining contract upgrades. However, the timelock mechanism was not in use at the time of the reassessment. Therefore, Inspex suggests the platform users to confirm the usage of the timelock mechanism before using the platform.

5.9.1. Description

In the PLP contract, the `onlyMinter` modifier is an access control modifier that allows to mint the PLP token by executing the `mint()` function with this modifier.

PLP.sol

```
56 function mint(address to, uint256 amount) public onlyMinter {
57   cooldown[to] = block.timestamp + liquidityCooldown;
58   _mint(to, amount);
59 }
```

Additionally, the contract owner can set any address to be the `onlyMinter` by calling the `setMinter` function.

PLP.sol

```
51 function setMinter(address minter, bool allow) external onlyOwner {
52   isMinter[minter] = allow;
53   emit PLP_SetMinter(minter, isMinter[minter], allow);
54 }
```

As a result, the owner can set any address as the `onlyMinter` modifier, which has the ability to mint an unlimited PLP Token.

5.9.2. Remediation

Inspex suggests allowing only the pool to mint the PLP token. This can be done by changing the `onlyMinter` modifier implementation, removing the `setMinter()` function, and setting the minter once in the `initialize()` function as follows:

PLP.sol

```
20  address public minter;
21  modifier onlyMinter() {
22    if (minter != msg.sender) revert PLP_NotMinter();
23    _;
24  }
25
26  function initialize(uint256 liquidityCooldown_, address pool_) external
    initializer {
27    OwnableUpgradeable.__Ownable_init();
28    ERC20Upgradeable.__ERC20_init("P88 Liquidity Provider", "PLP");
29
30    MAX_COOLDOWN_DURATION = 48 hours;
31    liquidityCooldown = liquidityCooldown_;
32    minter = pool_;
33  }
```

5.10. External Call to Untrusted Third Party Component

ID	IDX-010
Target	FarmFacet
Category	General Smart Contract Vulnerability
CWE	CWE-829: Inclusion of Functionality from Untrusted Control Sphere
Risk	**Severity:** High **Impact:** High The untrusted third party smart contract may harm the user's funds that are deposited by strategy. **Likelihood:** Medium The vault address of an untrusted smart contract can be set in strategy when deploying a strategy contract, which is controlled by the contract owner. It is possible that a third party who owns an untrusted contract may perform something malicious, intentionally or unintentionally.
Status	Resolved * The Perp88 has mitigated this issue by implementing a pending strategy mechanism that delays the strategy contract from being used by implementing the Timelock contract as the owner of all contracts to prevent immediate changes or upgrades to contracts, including the setting of the strategy of the platform.

5.10.1. Description

The `FarmFacet` contract has a `setStrategyOf()` function that could set the strategy address contract to the third party vault address.

FarmFacet.sol

```
54  function setStrategyOf(address token, StrategyInterface newStrategy)
55    external
56    onlyOwner
57  {
58    // Load PoolConfig Diamond storage
59    LibPoolConfigV1.PoolConfigV1DiamondStorage
60      storage poolConfigDs = LibPoolConfigV1.poolConfigV1DiamondStorage();
61
62    LibPoolConfigV1.StrategyData memory strategyData = poolConfigDs
63      .strategyDataOf[token];
64    StrategyInterface pendingStrategy = poolConfigDs.pendingStrategyOf[token];
65    if (strategyData.startTimestamp == 0 || pendingStrategy != newStrategy) {
66      // When adding new strategy or changing strategy
```

```
67      poolConfigDs.pendingStrategyOf[token] = newStrategy;
68      strategyData.startTimestamp = uint64(block.timestamp + STRATEGY_DELAY);
69    } else {
70      // When committing a new strategy
71      if (
72        strategyData.startTimestamp == 0 ||
73        block.timestamp < strategyData.startTimestamp
74      ) revert FarmFacet_TooEarlyToCommitStrategy();
75      if (address(poolConfigDs.strategyOf[token]) != address(0)) {
76        // If there is previous strategy, we need to withdraw all funds from it
77        int256 balanceChange = poolConfigDs.strategyOf[token].exit(
78          strategyData.principle
79        );
80        // Update totalOf[token] to sync physical balance with pool state
81        LibPoolV1.updateTotalOf(token);
82        // Realized profits/losses
83        if (balanceChange > 0) {
84          uint256 profit = uint256(balanceChange);
85          LibPoolV1.increasePoolLiquidity(token, profit);
86
87          emit StrategyRealizedProfit(token, profit);
88        } else if (balanceChange < 0) {
89          uint256 loss = uint256(-balanceChange);
90          LibPoolV1.decreasePoolLiquidity(token, loss);
91
92          emit StrategyRealizedLoss(token, loss);
93        }
94
95        emit StrategyDivest(token, strategyData.principle);
96      }
97      // Commit new strategy
98      poolConfigDs.strategyOf[token] = newStrategy;
99      strategyData.startTimestamp = 0;
100     strategyData.principle = 0;
101     poolConfigDs.pendingStrategyOf[token] = StrategyInterface(address(0));
102
103     emit SetStrategy(token, newStrategy);
104   }
105   poolConfigDs.strategyDataOf[token] = strategyData;
106 }
```

When the `farm()` function is called, the user's funds will be transferred to the strategy contract in line 173, and then the `run()` function call will deploy funds to the vault contract that is controlled by a third party.

FarmFacet.sol

```
121 function farm(address token, bool isRebalanceNeeded)
122   external
```

```
123   onlyPoolDiamondOrFarmKeeper
124 {
125   // Load PoolV1 diamond storage
126   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
127     .poolV1DiamondStorage();
128
129   // Load PoolConfig Diamond storage
130   LibPoolConfigV1.PoolConfigV1DiamondStorage
131     storage poolConfigDs = LibPoolConfigV1.poolConfigV1DiamondStorage();
132
133   // Load relevant variables
134   LibPoolConfigV1.StrategyData memory strategyData = poolConfigDs
135     .strategyDataOf[token];
136   StrategyInterface strategy = poolConfigDs.strategyOf[token];
137
138   // Realized profits or losses from strategy
139   int256 balanceChange = strategy.realized(
140     strategyData.principle,
141     msg.sender
142   );
143   // If there is no change in balance, and does not need to rebalance, then
    stop it here.
144   if (balanceChange == 0 && !isRebalanceNeeded) return;
145
146   if (balanceChange > 0) {
147     // If there is a profit, then increase pool liquidity
148     uint256 profits = uint256(balanceChange);
149     LibPoolV1.increasePoolLiquidity(token, profits);
150
151     emit StrategyRealizedProfit(token, profits);
152   } else if (balanceChange < 0) {
153     // If there is a loss, then decrease pool liquidity
154     uint256 losses = uint256(-balanceChange);
155     LibPoolV1.decreasePoolLiquidity(token, losses);
156     strategyData.principle -= losses.toUint128();
157
158     emit StrategyRealizedLoss(token, losses);
159   }
160
161   // If rebalance to make sure the strategy has the right amount of funds to
    deploy, then do it.
162   if (isRebalanceNeeded) {
163     // Calculate the target amount of funds to be deployed
164     uint256 targetDeployedFunds = ((poolV1ds.liquidityOf[token] -
165       poolV1ds.reservedOf[token]) * strategyData.targetBps) / 10000;
166
167     if (strategyData.principle < targetDeployedFunds) {
```

```
168       // If strategy short of funds, then deposit more funds
169       // Find out how much more funds to deposit
170       uint256 amountOut = targetDeployedFunds - strategyData.principle;
171
172       // Transfer funds from pool to strategy and run it
173       LibPoolV1.pushTokens(token, address(strategy), amountOut);
174       strategy.run(amountOut);
175
176       // Update how much pool put in the strategy
177       strategyData.principle += amountOut.toUint128();
178
179       emit StrategyInvest(token, amountOut);
180     } else if (strategyData.principle > targetDeployedFunds) {
181       // If strategy has more funds than it should be, then withdraw some funds
182       // Find out how much funds to withdraw
183       uint256 amountIn = strategyData.principle - targetDeployedFunds;
184
185       // Withdraw funds from strategy and transfer it back to pool
186       uint256 actualAmountIn = strategy.withdraw(amountIn);
187
188       // Update how much pool put in the strategy
189       strategyData.principle -= actualAmountIn.toUint128();
190
191       emit StrategyDivest(token, actualAmountIn);
192     }
193   }
194
195   poolConfigDs.strategyDataOf[token] = strategyData;
196 }
```

The risk of the platform funds will depend on the external third party smart contract or platform risk that the strategy contract interacts with.

5.10.2. Remediation

Inspex suggests performing external calls to only known and trusted smart contracts and avoiding the use of unreliable external third party smart contracts for managing the platform's funds.

5.11. Denial of Service from Rewarders Configuration

ID	IDX-011
Target	PLPStaking
Category	Advanced Smart Contract Vulnerability
CWE	CWE-703: Improper Check or Handling of Exceptional Conditions
Risk	**Severity:** Medium **Impact:** High The users cannot deposit or withdraw their capital from the contract if there is an address that does not conform with a rewarder contract in the rewarder list or the amount of the registered rewarder contracts is too high. **Likelihood:** Low Only the contract owner can add a new rewarder into the contract.
Status	Resolved The Perp88 team has resolved this issue by implementing the `removeRewarderForTokenByIndex()` function for removing the unwanted rewarders from the rewarder list in the contract.

5.11.1. Description

The `PLPStaking` contract is a contract that allows users to stake specific tokens to gain reward tokens from the rewarders.

The owner of the staking contract can manually add rewarder contracts into the staking contract through the `addStakingToken()` and `addRewarder()` functions. The new rewarders will be added to the `stakingTokenRewarders` and `rewarderStakingTokens` mappings, which could be collectively called a rewarder list.

PLPStaking.sol

```
39  function addStakingToken(address newToken, address[] memory newRewarders)
40    external
41    onlyOwner
42  {
43    uint256 length = newRewarders.length;
44    for (uint256 i = 0; i < length; ) {
45      _updatePool(newToken, newRewarders[i]);
46
47      unchecked {
48        ++i;
49      }
```

```
50    }
51  }
52
53  function addRewarder(address newRewarder, address[] memory newTokens)
54    external
55    onlyOwner
56  {
57    uint256 length = newTokens.length;
58    for (uint256 i = 0; i < length; ) {
59      _updatePool(newTokens[i], newRewarder);
60
61      unchecked {
62        ++i;
63      }
64    }
65  }
66
67  function _updatePool(address newToken, address newRewarder) internal {
68    stakingTokenRewarders[newToken].push(newRewarder);
69    rewarderStakingTokens[newRewarder].push(newToken);
70    isStakingToken[newToken] = true;
71    if (!isRewarder[newRewarder]) {
72      isRewarder[newRewarder] = true;
73    }
74  }
```

When a user calls the `deposit()` function to stake a token to the staking contract, the function calls the `onDeposit()` function from the address in the rewarder list (`stakingTokenRewarders`) . If one of them fails, the whole transaction will be reverted.

PLPStaking.sol

```
80  function deposit(
81    address to,
82    address token,
83    uint256 amount
84  ) external {
85    if (!isStakingToken[token]) revert PLPStaking_UnknownStakingToken();
86
87    uint256 length = stakingTokenRewarders[token].length;
88    for (uint256 i = 0; i < length; ) {
89      address rewarder = stakingTokenRewarders[token][i];
90
91      IRewarder(rewarder).onDeposit(to, amount);
92
93      unchecked {
94        ++i;
```

```
95      }
96    }
97
98    userTokenAmount[token][to] += amount;
99    IERC20Upgradeable(token).safeTransferFrom(
100     msg.sender,
101     address(this),
102     amount
103   );
104
105   emit LogDeposit(msg.sender, to, token, amount);
106 }
```

The rewarder list (`stakingTokenRewarders`) is also present in the `_withdraw()` function, which is a function for a user to withdraw their staked tokens. The function will loop through the `stakingTokenRewarders` mapping likewise on the `deposit()` function; the whole transaction will fail if there is an address that always fails to perform the `onWithdraw()` function.

PLPStaking.sol

```
124 function withdraw(address token, uint256 amount) external {
125   _withdraw(token, amount);
126   emit LogWithdraw(msg.sender, token, amount);
127 }
128
129 function _withdraw(address token, uint256 amount) internal {
130   if (!isStakingToken[token]) revert PLPStaking_UnknownStakingToken();
131   if (userTokenAmount[token][msg.sender] < amount)
132     revert PLPStaking_InsufficientTokenAmount();
133
134   uint256 length = stakingTokenRewarders[token].length;
135   for (uint256 i = 0; i < length; ) {
136     address rewarder = stakingTokenRewarders[token][i];
137
138     IRewarder(rewarder).onWithdraw(msg.sender, amount);
139
140     unchecked {
141       ++i;
142     }
143   }
144   userTokenAmount[token][msg.sender] -= amount;
145   IERC20Upgradeable(token).safeTransfer(msg.sender, amount);
146   emit LogWithdraw(msg.sender, token, amount);
147 }
```

The failures from calling the `deposit()` or `withdraw()` functions can also be caused from having too many rewarders in the rewarder list due to the gas usage.

5.11.2. Remediation

Inspex suggests adding a function for removing undesired rewarders from the rewarder list in the contract.

For example, implement a privilege function that removes a rewarder from the `stakingTokenRewarders` and `rewarderStakingTokens` states and set the `isRewarder` state of the removed rewarder to `false`. We can find the index of the removing rewarder externally and use it to remove the rewarder.

PLPStaking.sol

```
220 function removeRewarderForTokenByIndex(uint256 removeRewarderIndex, address
    token)
221   external
222   onlyOwner
223 {
224   uint256 tokenLength = stakingTokenRewarders[token].length;
225   address removedRewarder = stakingTokenRewarders[token][removeRewarderIndex];
226   stakingTokenRewarders[token][removeRewarderIndex] =
    stakingTokenRewarders[token][tokenLength - 1];
227   stakingTokenRewarders[token].pop();
228
229   uint256 rewarderLength = rewarderStakingTokens[removedRewarder].length;
230   for (uint i = 0; i < rewarderLength;) {
231     if (rewarderStakingTokens[removedRewarder][i] == token) {
232       rewarderStakingTokens[removedRewarder][i] =
    rewarderStakingTokens[removedRewarder][rewarderLength - 1];
233       rewarderStakingTokens[removedRewarder].pop();
234       if (rewarderLength == 1) isRewarder[removedRewarder] = false;
235
236       break;
237     }
238     unchecked {
239       ++i;
240     }
241   }
242 }
```

5.12. Improper Share Calculation

ID	IDX-012
Target	PLPStaking
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium The number of tokens is used to calculate shares, implying that the high value or low decimal tokens receive a smaller share than they should. This results in the available rewards for platform users being incorrect. **Likelihood:** Medium Only the contract owner could add allowed staking tokens into the contract. The share calculation will be accurate if the owner adds only the PLP token and not any other tokens.
Status	Resolved * The Perp88 team has clarified that only PLP token will be added to the `PLPStaking` contract. The staking share calculation will be calculated by using the PLP token amount in the contract. This could prevent the incorrect share calculation.

5.12.1. Description

The user's accumulated reward is calculated in the `AdHocMintRewarder` contract using the `_calculateUserAccReward()` function. The user's cumulative reward must be calculated using the user share value. Thus, the `_calculateUserAccReward()` function calls the `_userShare()` function in line 114 to calculate the user share value. The `calculateShare()` function is then called by the `_userShare()` function to calculate the value of the user share.

AdHocMintRewarder.sol

```
107 function _calculateUserAccReward(address user)
108   internal
109   view
110   returns (uint256)
111 {
112   // [100% APR] If a user stake N shares for a year, he will be rewarded with N
    tokens.
113   return
114     ((block.timestamp - userLastRewards[user]) * _userShare(user)) / YEAR;
115 }
116
```

```
117 function _userShare(address user) private view returns (uint256) {
118   return IStaking(staking).calculateShare(address(this), user);
119 }
```

In line 188, the `calculateShare()` function used the `userTokenAmount` to calculate the share value. This line treats all tokens as having the same value, even though their decimals are different.

PLPStaking.sol

```
179 function calculateShare(address rewarder, address user)
180   external
181   view
182   returns (uint256)
183 {
184   address[] memory tokens = rewarderStakingTokens[rewarder];
185   uint256 share = 0;
186   uint256 length = tokens.length;
187   for (uint256 i = 0; i < length; ) {
188     share += userTokenAmount[tokens[i]][user];
189
190     unchecked {
191       ++i;
192     }
193   }
194   return share;
195 }
```

The `calculateTotalShare()` function operates similarly to the `calculateShare()` function in this instance, but in addition it uses the `balanceOf(address(this))` value to calculate `totalShare` instead, which also treats all tokens as having the same value.

PLPStaking.sol

```
197 function calculateTotalShare(address rewarder)
198   external
199   view
200   returns (uint256)
201 {
202   address[] memory tokens = rewarderStakingTokens[rewarder];
203   uint256 totalShare = 0;
204   uint256 length = tokens.length;
205   for (uint256 i = 0; i < length; ) {
206     totalShare += IERC20Upgradeable(tokens[i]).balanceOf(address(this));
207
208     unchecked {
209       ++i;
210     }
211   }
```

```
212     return totalShare;
213 }
```

5.12.2. Remediation

In the case that the `PLPStaking` contract only allows users to stake only the PLP token in the contract, Inspex suggests validating the token address in the `addStakingToken()` and `addRewarder()` functions to ensure that only the PLP token is added. If the multiple tokens staking is needed, the share calculation must be separate for each pool to calculate the proper reward value.

5.13. Improper Sanity Check

ID	IDX-013
Target	AdminFacet
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Medium **Impact:** High The price oracle contract can be later changed by the facet owner. If a defective price oracle is added, it can not be changed anymore and every contract that depends on the oracle will always be reverted. **Likelihood:** Low The oracle address is a critical state that changing it causes a widespread impact. The owner must always be careful to edit such a state. But it can also be intentionally changed to halt the whole platform to act as an emergency function that prevents the users from moving their assets in or out of the platform. Therefore, it is unlikely that the platform owner will change the oracle address.
Status	Resolved The Perp88 team has resolved this issue by changing the sanity check target on the `AdminFacet` contract to the new oracle.

5.13.1. Description

The pool contract is written with the Diamond Facet pattern. There is a proxy contract that stores various configurations of the platform and shares the configurations across the facets. Therefore the sharing storage across the facets is called a diamond storage in the Diamond Facet pattern.

The `setPoolOracle()` function in the `AdminFacet` contract is a function for setting a new `PoolOracle` state in the diamond storage. Before the function sets a new oracle into the diamond storage in line 127, the function will call the `roundDepth()` function of the stored oracle state in line 124.

The problem will occur if a random address that is not a pool oracle is set into the diamond storage. The contract owner can not change the oracle back into the valid one because of the invalid oracle that has already been set can not perform the `roundDepth()` function and the invalid oracle also can not function as a proper oracle for the platform.

AdminFacet.sol

```
118 function setPoolOracle(PoolOracle newPoolOracle) external onlyOwner {
119   // Load diamond storage
120   LibPoolV1.PoolV1DiamondStorage storage ds = LibPoolV1
```

```
121         .poolV1DiamondStorage();
122
123     // Sanity check
124     ds.oracle.roundDepth();
125
126     emit SetPoolOracle(ds.oracle, newPoolOracle);
127     ds.oracle = newPoolOracle;
128 }
```

5.13.2. Remediation

Inspex suggests changing the sanity check target to check the new oracle instead of the old one.

AdminFacet.sol

```
118 function setPoolOracle(PoolOracle newPoolOracle) external onlyOwner {
119     // Load diamond storage
120     LibPoolV1.PoolV1DiamondStorage storage ds = LibPoolV1
121         .poolV1DiamondStorage();
122
123     // Sanity check
124     newPoolOracle.roundDepth();
125
126     emit SetPoolOracle(ds.oracle, newPoolOracle);
127     ds.oracle = newPoolOracle;
128 }
```

5.14. User Reward Miscalculation

ID	IDX-014
Target	AdHocMintRewarder
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** High When the `onDeposit()` or `onHarvest()` function is called for the first time, the `userLastRewards[user]` is still 0 which causes the `calculateUserAccReward(user)` to be miscalculated by (`block.timestamp - userLastRewards[user]`). Resulting in a very large reward amount. **Likelihood:** Low This issue only occurs when the rewarder is added after the user has already staked the token.
Status	Resolved The Perp88 team has resolved this issue by validating the `userLastRewards[user]` value to prevent reward miscalculation on the first deposit or harvest of the newly added rewarder.

5.14.1. Description

When a user stakes tokens on the platform to accumulate user rewards, the `onDeposit()` function is called. In line 64, the `onDeposit()` function calls the `_calculateUserAccReward()` function to calculate the user's accumulated reward.

AdHocMintRewarder.sol

```
59 function onDeposit(address user, uint256 shareAmount)
60   external
61   onlyStakingContract
62 {
63   // Accumulate user reward
64   userAccRewards[user] += _calculateUserAccReward(user);
65   userLastRewards[user] = block.timestamp.toUint64();
66   emit LogOnDeposit(user, shareAmount);
67 }
```

The `_calculateUserAccReward()` function returns the calculated user's accumulated reward. When the user deposits for the first time after the new rewarder was added and the `_userShare(user)` is not 0 (already deposited), the `userLastRewards[user]` will be 0 which makes the user's accumulated reward

calculation result in a large amount.

AdHocMintRewarder.sol

```
107 function _calculateUserAccReward(address user)
108   internal
109   view
110   returns (uint256)
111 {
112   // [100% APR] If a user stake N shares for a year, he will be rewarded with N
    tokens.
113   return
114     ((block.timestamp - userLastRewards[user]) * _userShare(user)) / YEAR;
115 }
```

Furthermore, the `_pendingReward()` function, which is triggered by the `onHarvest()` function, also calls the `_calculateUserAccReward()` function. When the user harvests for the first time after the new rewarder was added, the `userLastRewards[user]` will be 0 which makes the user's accumulated reward miscalculated.

AdHocMintRewarder.sol

```
80  function onHarvest(address user, address receiver)
81    external
82    onlyStakingContract
83  {
84    uint256 pendingRewardAmount = _pendingReward(user);
85
86    // Reset user reward accumulation.
87    // The next action will start accum reward from zero again.
88    userAccRewards[user] = 0;
89    userLastRewards[user] = block.timestamp.toUint64();
90
91    if (pendingRewardAmount != 0) {
92      _harvestToken(receiver, pendingRewardAmount);
93    }
94
95    emit LogHarvest(user, pendingRewardAmount);
96  }
97
98  function pendingReward(address user) external view returns (uint256) {
99    return _pendingReward(user);
100 }
101
102 function _pendingReward(address user) internal view returns (uint256) {
103   // (accumulated reward since the last action) + (jotted reward from the past)
104   return _calculateUserAccReward(user) + userAccRewards[user];
105 }
```

5.14.2. Remediation

Inspex suggests checking the `userLastRewards[user]` value is greater than 0 before calculating the reward to prevent the reward miscalculation on the first deposit or harvest of the newly added rewarder.

AdHocMintRewarder.sol

```
107 function _calculateUserAccReward(address user)
108   internal
109   view
110   returns (uint256)
111 {
112   // [100% APR] If a user stake N shares for a year, he will be rewarded with N
    tokens.
113   if (userLastRewards[user] > 0) {
114     return ((block.timestamp - userLastRewards[user]) * _userShare(user)) /
    YEAR;
115   } else {
116     return 0;
117   }
118 }
```

5.15. Arbitrary Reward Rate Set

ID	IDX-015
Target	RewardDistributor
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Medium **Impact:** Low The users can manipulate the reward rate of the rewarder contracts to claim all the eligible rewards in a short time. **Likelihood:** High Any user can call the `claimAndFeedProtocolRevenue()` function to manipulate the reward rate of the registered rewarder contracts.
Status	Resolved The Perp88 team has resolved this issue by implementing the `onlyFeeder` modifier for the `claimAndFeedProtocolRevenue()` function to prevent a user from manipulating the reward rate through this function.

5.15.1. Description

The `claimAndFeedProtocolRevenue()` function in the `RewardDistributor()` contract can be called by anyone. The function can indirectly manipulate the reward rate by changing the new reward period. The `feedingExpiredAt` parameter on the function will be passed into the `_feedRewardToRewarders()` function in line 138. Then, in the `_feedRewardToRewarders()` function, the `feedingExpiredAt` parameter will be passed to the `feedWithExpiredAt()` function on a contract with the `IFeedableRewarder` interface.

RewardDistributor.sol

```
102 function claimAndFeedProtocolRevenue(
103   address[] memory tokens,
104   uint256 feedingExpiredAt
105 ) external {
106   uint256 length = tokens.length;
107   for (uint256 i = 0; i < length; ) {
108     // 1. Withdraw protocol revenue
109     _withdrawProtocolRevenue(tokens[i]);
110
111     // 2. Collect dev fund
112     _collectDevFund(tokens[i]);
113
114     unchecked {
```

```
115       i++;
116     }
117   }
118
119   // Note: We need to seprate this loop from another loop, since if we have the
      input token
120   // the same token as reward token, in our case WMatic, we could end up in
      overcollecting
121   // dev fund.
122   // The reason is that we _collectDevFund() as input token and then
      _swapTokenToRewardToken,
123   // on the next loop with reward token as input token, the claimed revenue
      would get mixed up.
124   for (uint256 i = 0; i < length; ) {
125     // 3. Swap those revenue (along with surplus) to RewardToken Token
126     _swapTokenToRewardToken(
127       tokens[i],
128       IERC20Upgradeable(tokens[i]).balanceOf(address(this))
129     );
130
131     unchecked {
132       i++;
133     }
134   }
135
136   // At this point, we got a portion of reward tokens.
137   // 4. Feed reward to both rewarders
138   _feedRewardToRewarders(feedingExpiredAt);
139 }
```

RewardDistributor.sol

```
182 function _feedRewardToRewarders(uint256 feedingExpiredAt) internal {
183   uint256 totalRewardAmount = IERC20Upgradeable(rewardToken).balanceOf(
184     address(this)
185   );
186
187   uint256 plpStakingRewardAmount = (totalRewardAmount * plpStakingBps) /
188     10000;
189   uint256 dragonStakingRewardAmount = totalRewardAmount -
190     plpStakingRewardAmount;
191
192   // Approve and feed to PLPStaking
193   IERC20Upgradeable(rewardToken).approve(
194     plpStakingProtocolRevenueRewarder,
195     plpStakingRewardAmount
196   );
197   IFeedableRewarder(plpStakingProtocolRevenueRewarder).feedWithExpiredAt(
```

```
198       plpStakingRewardAmount,
199       feedingExpiredAt
200     );
201
202     // Approve and feed to DragonStaking
203     IERC20Upgradeable(rewardToken).approve(
204       dragonStakingProtocolRevenueRewarder,
205       dragonStakingRewardAmount
206     );
207     IFeedableRewarder(dragonStakingProtocolRevenueRewarder).feedWithExpiredAt(
208       dragonStakingRewardAmount,
209       feedingExpiredAt
210     );
211   }
```

By referring to the implementation on the `FeedableRewarder` contract, the passed parameter, `feedingExpiredAt`, will be used to calculate the duration in the `_feed()` function. Effectively, a user can set the reward duration to a short time, like `block.timestamp + 1`, which will have the rewarder distribute all rewards in `1` second. The duration can also be set to a long period, like `MAXUINT256`, which will diminish the reward to `0`.

FeedableRewarder.sol

```
141 function feedWithExpiredAt(uint256 feedAmount, uint256 expiredAt)
142   external
143   onlyFeeder
144 {
145   _feed(feedAmount, expiredAt - block.timestamp);
146 }
```

FeedableRewarder.sol

```
152 function _feed(uint256 feedAmount, uint256 duration) internal {
153   uint256 totalShare = _totalShare();
154   _forceUpdateRewardCalculationParams(totalShare);
155
156   {
157     // Transfer token, with decay check
158     uint256 balanceBefore = IERC20Upgradeable(rewardToken).balanceOf(
159       address(this)
160     );
161     IERC20Upgradeable(rewardToken).safeTransferFrom(
162       msg.sender,
163       address(this),
164       feedAmount
165     );
166
```

```
167     if (
168       IERC20Upgradeable(rewardToken).balanceOf(address(this)) -
169         balanceBefore !=
170       feedAmount
171     ) revert FeedableRewarderError_FeedAmountDecayed();
172   }
173
174   uint256 leftOverReward = rewardRateExpiredAt > block.timestamp
175     ? (rewardRateExpiredAt - block.timestamp) * rewardRate
176     : 0;
177   uint256 totalRewardAmount = leftOverReward + feedAmount;
178
179   rewardRate = totalRewardAmount / duration;
180   rewardRateExpiredAt = block.timestamp + duration;
181
182   emit LogFeed(feedAmount, rewardRate, rewardRateExpiredAt);
183 }
```

5.15.2. Remediation

Inspex suggests enforcing access control on the `claimAndFeedProtocolRevenue()` function to prevent a user from manipulating the reward rate through this function. By changing the function to a privilege function, the function must be controlled through a timelock or a governance contract, which can be referred to the remediation of the issue `IDX-008 Centralized Control of State Variable`.

RewardDistributor.sol

```
102 function claimAndFeedProtocolRevenue(
103   address[] memory tokens,
104   uint256 feedingExpiredAt
105 ) external onlyOwner {
106   uint256 length = tokens.length;
107   for (uint256 i = 0; i < length; ) {
108     // 1. Withdraw protocol revenue
109     _withdrawProtocolRevenue(tokens[i]);
110
111     // 2. Collect dev fund
112     _collectDevFund(tokens[i]);
113
114     unchecked {
115       i++;
116     }
117   }
118
119   // Note: We need to seprate this loop from another loop, since if we have the
    input token
120   // the same token as reward token, in our case WMatic, we could end up in
    overcollecting
```

```
121   // dev fund.
122   // The reason is that we _collectDevFund() as input token and then
      _swapTokenToRewardToken,
123   // on the next loop with reward token as input token, the claimed revenue
      would get mixed up.
124   for (uint256 i = 0; i < length; ) {
125     // 3. Swap those revenue (along with surplus) to RewardToken Token
126     _swapTokenToRewardToken(
127       tokens[i],
128       IERC20Upgradeable(tokens[i]).balanceOf(address(this))
129     );
130
131     unchecked {
132       i++;
133     }
134   }
135
136   // At this point, we got a portion of reward tokens.
137   // 4. Feed reward to both rewarders
138   _feedRewardToRewarders(feedingExpiredAt);
139 }
```

5.16. Insufficient roundDepth Input Validation

ID	IDX-016
Target	PoolOracle
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Medium **Impact:** High The attacker could easily perform the arbitrage action against the liquidity pool to gain the profit from the price gap. This results in loss of liquidity pool value. **Likelihood:** Low This issue only occurs when the `roundDepth` state is set to 1. However, only the contract owner can set this state.
Status	Resolved The Perp88 team has resolved this issue by adjusting the `_roundDepth` value validation to ensure that the `_roundDepth` value is greater than 2 in order to prevent the price gap on the platform.

5.16.1. Description

In the `PoolOracle` contract, the `roundDepth` state can be set via the `initialize()` and `setRoundDepth()` functions. The `_roundDepth` parameter must not equal to 0 to pass the input validation.

PoolOracle.sol

```
40  function initialize(uint80 _roundDepth) external initializer {
41    OwnableUpgradeable.__Ownable_init();
42
43    if (_roundDepth == 0) revert PoolOracle_BadArguments();
44    roundDepth = _roundDepth;
45  }
```

PoolOracle.sol

```
162 function setRoundDepth(uint80 _roundDepth) external onlyOwner {
163   if (_roundDepth == 0) revert PoolOracle_BadArguments();
164
165   emit SetRoundDepth(roundDepth, _roundDepth);
166   roundDepth = _roundDepth;
167 }
```

However, the `roundDepth` state can be set to 1, resulting in a price gap opportunity for any action on the

platform, such as spot or margin trading. Because the following for loop in the `_getPrice()` function will only be called once, the difference between the current price and the next Oracle price is immediately effective.

PoolOracle.sol

```
47  function _getPrice(address token, bool isUseMaxPrice)
48    internal
49    view
50    returns (uint256)
51  {
52    // SLOAD
53    PriceFeedInfo memory priceFeed = priceFeedInfo[token];
54    if (address(priceFeed.priceFeed) == address(0))
55      revert PoolOracle_PriceFeedNotAvailable();
56
57    uint256 price = 0;
58    int256 _priceCursor = 0;
59    uint256 priceCursor = 0;
60    uint80 latestRoundId = priceFeed.priceFeed.latestRound();
61
62    for (uint80 i = 0; i < roundDepth; i++) {
63      if (i >= latestRoundId) break;
64
65      if (i == 0) {
66        priceCursor = priceFeed.priceFeed.latestAnswer().toUint256();
67      } else {
68        (, _priceCursor, , , ) = priceFeed.priceFeed.getRoundData(
69          latestRoundId - i
70        );
71        priceCursor = _priceCursor.toUint256();
72      }
73
74      if (price == 0) {
75        price = priceCursor;
76        continue;
77      }
78
79      if (isUseMaxPrice && price < priceCursor) {
80        price = priceCursor;
81        continue;
82      }
83
84      if (!isUseMaxPrice && price > priceCursor) {
85        price = priceCursor;
86      }
87    }
88
```

```
89    if (price == 0) revert PoolOracle_UnableFetchPrice();
90
91    price = (price * PRICE_PRECISION) / 10**priceFeed.decimals;
92
93    // Handle strict stable price deviation.
94    if (priceFeed.isStrictStable) {
95      uint256 delta;
96      unchecked {
97        delta = price > ONE_USD ? price - ONE_USD : ONE_USD - price;
98      }
99
100     if (delta <= maxStrictPriceDeviation) return ONE_USD;
101
102     if (isUseMaxPrice && price > ONE_USD) return price;
103
104     if (!isUseMaxPrice && price < ONE_USD) return price;
105
106     return ONE_USD;
107   }
108
109   // Handle spreadBasisPoint
110   if (isUseMaxPrice) return (price * (BPS + priceFeed.spreadBps)) / BPS;
111
112   return (price * (BPS - priceFeed.spreadBps)) / BPS;
113 }
```

For example, in the `LiquidityFacet.swap()` function, the `poolV1ds.oracle.getMinPrice()` and `poolV1ds.oracle.getMaxPrice()` functions are called to get the `tokenIn` and `tokenOut` prices from the Oracle.

LiquidityFacet.sol

```
365 function swap(
366   address account,
367   address tokenIn,
368   address tokenOut,
369   uint256 minAmountOut,
370   address receiver
371 ) external nonReentrant returns (uint256) {
372   SwapLocalVars memory vars;
373
374   // LOAD diamond storage
375   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
376     .poolV1DiamondStorage();
377
378   // Pull Tokens
379   uint256 amountIn = LibPoolV1.pullTokens(tokenIn);
```

```
380
381   if (!LibPoolConfigV1.isSwapEnable()) revert LiquidityFacet_SwapDisabled();
382   if (!LibPoolConfigV1.isAcceptToken(tokenIn))
383     revert LiquidityFacet_BadTokenIn();
384   if (!LibPoolConfigV1.isAcceptToken(tokenOut))
385     revert LiquidityFacet_BadTokenOut();
386   if (tokenIn == tokenOut) revert LiquidityFacet_SameTokenInTokenOut();
387   if (amountIn == 0) revert LiquidityFacet_BadAmount();
388
389   LibPoolV1.realizedFarmPnL(tokenIn);
390   LibPoolV1.realizedFarmPnL(tokenOut);
391
392   FundingRateFacetInterface(address(this)).updateFundingRate(
393     tokenIn,
394     tokenIn
395   );
396   FundingRateFacetInterface(address(this)).updateFundingRate(
397     tokenOut,
398     tokenOut
399   );
400
401   vars.priceIn = poolV1ds.oracle.getMinPrice(tokenIn);
402   vars.priceOut = poolV1ds.oracle.getMaxPrice(tokenOut);
403
404   vars.amountOut = (amountIn * vars.priceIn) / vars.priceOut;
405   vars.amountOut = LibPoolV1.convertTokenDecimals(
406     LibPoolConfigV1.getTokenDecimalsOf(tokenIn),
407     LibPoolConfigV1.getTokenDecimalsOf(tokenOut),
408     vars.amountOut
409   );
410
411   // Adjust USD debt as swap shifted the debt between two assets
412   vars.usdDebt = (amountIn * vars.priceIn) / PRICE_PRECISION;
413   vars.usdDebt = LibPoolV1.convertTokenDecimals(
414     LibPoolConfigV1.getTokenDecimalsOf(tokenIn),
415     USD_DECIMALS,
416     vars.usdDebt
417   );
418
419   uint256 swapFeeBps = GetterFacetInterface(address(this)).getSwapFeeBps(
420     tokenIn,
421     tokenOut,
422     vars.usdDebt
423   );
424   uint256 amountOutAfterFee = _collectSwapFee(
425     tokenOut,
426     poolV1ds.oracle.getMinPrice(tokenOut),
```

```
427       vars.amountOut,
428       swapFeeBps,
429       account,
430       LiquidityAction.SWAP
431     );
432
433     LibPoolV1.increasePoolLiquidity(tokenIn, amountIn);
434     LibPoolV1.increaseUsdDebt(tokenIn, vars.usdDebt);
435
436     LibPoolV1.decreasePoolLiquidity(tokenOut, vars.amountOut);
437     LibPoolV1.decreaseUsdDebt(tokenOut, vars.usdDebt);
438
439     // Buffer check
440     if (
441       poolV1ds.liquidityOf[tokenOut] <
442       LibPoolConfigV1.getTokenBufferLiquidityOf(tokenOut)
443     ) revert LiquidityFacet_LiquidityBuffer();
444
445     // Slippage check
446     if (amountOutAfterFee < minAmountOut) revert LiquidityFacet_Slippage();
447
448     // Transfer amount out.
449     LibPoolV1.tokenOut(tokenOut, receiver, amountOutAfterFee);
450     emit Swap(
451       receiver,
452       tokenIn,
453       tokenOut,
454       amountIn,
455       vars.amountOut,
456       amountOutAfterFee,
457       swapFeeBps
458     );
459
460     return amountOutAfterFee;
461   }
```

The attacker could use this issue to gain profit from the price gap by monitoring the next Oracle price setting transaction for a profitable trading opportunity. Resulting in a loss of value for pool liquidity.

5.16.2. Remediation

Inspex suggests implementing the proper input validation to prevent price gap on the platform. The `_roundDepth` value must not be less than 2.

PoolOracle.sol

```
40  function initialize(uint80 _roundDepth) external initializer {
41    OwnableUpgradeable.__Ownable_init();
```

```
42
43    if (_roundDepth < 2) revert PoolOracle_BadArguments();
44    roundDepth = _roundDepth;
45  }
```

PoolOracle.sol

```
162 function setRoundDepth(uint80 _roundDepth) external onlyOwner {
163   if (_roundDepth < 2) revert PoolOracle_BadArguments();
164
165   emit SetRoundDepth(roundDepth, _roundDepth);
166   roundDepth = _roundDepth;
167 }
```

5.17. Design Flaw in Fixed Rate Token Swap

ID	IDX-017
Target	PoolOracle
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** High The attacker could perform the arbitrage swap between the DEX and Perp88's liquidity pool to gain the profit from the fixed rate swap. All profitable assets in the pool will be swapped out. **Likelihood:** Low The arbitrage opportunity occurs when the difference between the price on DEX and the platform's oracle is large enough for an arbitrager to be able to leverage for the profit.
Status	Resolved * The Perp88 team has mitigated this issue by implementing various security measures, such as the usage of the Chainlink price oracle, tax fee charging as a virtual price impact depending on the PLP token weights, and a liquidity cooldown on PLP deposit and withdrawal. These security measures could prevent attackers from exploiting this feature.

5.17.1. Description

The platform uses an oracle as a reference for the price of each asset in $USD. When a user performs a swap from the `LiquidityFacet` contract, the price of the swapped tokens will only depend on the prices from the Oracle of the platform in lines 401 and 402.

Therefore, if the price from the oracle deviates from the market price, there will be an opportunity for arbitraging on the platform. Then, the price-deviated token will fluctuate into the platform to drain another token from the platform, which causes an acute shortage of liquidity and a loss of the total value of the pool.

LiquidityFacet.sol

```
365 function swap(
366   address account,
367   address tokenIn,
368   address tokenOut,
369   uint256 minAmountOut,
370   address receiver
371 ) external nonReentrant returns (uint256) {
372   SwapLocalVars memory vars;
373
```

```
374   // LOAD diamond storage
375   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
376     .poolV1DiamondStorage();
377
378   // Pull Tokens
379   uint256 amountIn = LibPoolV1.pullTokens(tokenIn);
380
381   if (!LibPoolConfigV1.isSwapEnable()) revert LiquidityFacet_SwapDisabled();
382   if (!LibPoolConfigV1.isAcceptToken(tokenIn))
383     revert LiquidityFacet_BadTokenIn();
384   if (!LibPoolConfigV1.isAcceptToken(tokenOut))
385     revert LiquidityFacet_BadTokenOut();
386   if (tokenIn == tokenOut) revert LiquidityFacet_SameTokenInTokenOut();
387   if (amountIn == 0) revert LiquidityFacet_BadAmount();
388
389   LibPoolV1.realizedFarmPnL(tokenIn);
390   LibPoolV1.realizedFarmPnL(tokenOut);
391
392   FundingRateFacetInterface(address(this)).updateFundingRate(
393     tokenIn,
394     tokenIn
395   );
396   FundingRateFacetInterface(address(this)).updateFundingRate(
397     tokenOut,
398     tokenOut
399   );
400
401   vars.priceIn = poolV1ds.oracle.getMinPrice(tokenIn);
402   vars.priceOut = poolV1ds.oracle.getMaxPrice(tokenOut);
403
404   vars.amountOut = (amountIn * vars.priceIn) / vars.priceOut;
405   vars.amountOut = LibPoolV1.convertTokenDecimals(
406     LibPoolConfigV1.getTokenDecimalsOf(tokenIn),
407     LibPoolConfigV1.getTokenDecimalsOf(tokenOut),
408     vars.amountOut
409   );
410
411   // Adjust USD debt as swap shifted the debt between two assets
412   vars.usdDebt = (amountIn * vars.priceIn) / PRICE_PRECISION;
413   vars.usdDebt = LibPoolV1.convertTokenDecimals(
414     LibPoolConfigV1.getTokenDecimalsOf(tokenIn),
415     USD_DECIMALS,
416     vars.usdDebt
417   );
418
419   uint256 swapFeeBps = GetterFacetInterface(address(this)).getSwapFeeBps(
420     tokenIn,
```

```
421       tokenOut,
422       vars.usdDebt
423     );
424     uint256 amountOutAfterFee = _collectSwapFee(
425       tokenOut,
426       poolV1ds.oracle.getMinPrice(tokenOut),
427       vars.amountOut,
428       swapFeeBps,
429       account,
430       LiquidityAction.SWAP
431     );
432
433     LibPoolV1.increasePoolLiquidity(tokenIn, amountIn);
434     LibPoolV1.increaseUsdDebt(tokenIn, vars.usdDebt);
435
436     LibPoolV1.decreasePoolLiquidity(tokenOut, vars.amountOut);
437     LibPoolV1.decreaseUsdDebt(tokenOut, vars.usdDebt);
438
439     // Buffer check
440     if (
441       poolV1ds.liquidityOf[tokenOut] <
442       LibPoolConfigV1.getTokenBufferLiquidityOf(tokenOut)
443     ) revert LiquidityFacet_LiquidityBuffer();
444
445     // Slippage check
446     if (amountOutAfterFee < minAmountOut) revert LiquidityFacet_Slippage();
447
448     // Transfer amount out.
449     LibPoolV1.tokenOut(tokenOut, receiver, amountOutAfterFee);
450     emit Swap(
451       receiver,
452       tokenIn,
453       tokenOut,
454       amountIn,
455       vars.amountOut,
456       amountOutAfterFee,
457       swapFeeBps
458     );
459
460     return amountOutAfterFee;
461   }
```

5.17.2. Remediation

Inspex suggests implementing the mechanism to prevent the arbitrager from profitable action against the liquidity pool, such as applying the cooldown on the add/remove liquidity, allowing only the platform itself to perform the swap at the fixed price rate.

5.18. Arbitrary Resetting PLP Transfer Cooldown

ID	IDX-018
Target	LiquidityFacet
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Low **Impact:** Low The attacker can call the `addLiquidity()` function with the victim's address as a receiver to reset the PLP token transfer cooldown on the target. However, the victim can still use the PLP token with the platform as normal since the platform address should be whitelisted from the transfer cooldown. **Likelihood:** Medium There is no restriction on the `addLiquidity()` function, but the attacker needs to add the actual liquidity for the victim to exploit this attack.
Status	**Acknowledged** The Perp88 team has clarified that this feature is used for providing more composability to other projects and for various use cases.

5.18.1. Description

In the `PLP` contract, the `mint()` function will set the transfer cooldown of the receiver to the current `block.timestamp + liquidityCooldown` as shown below.

PLP.sol

```
56 function mint(address to, uint256 amount) public onlyMinter {
57   cooldown[to] = block.timestamp + liquidityCooldown;
58   _mint(to, amount);
59 }
```

If the cooldown timestamp has not yet passed, the user cannot transfer the PLP tokens as checked in the `_beforeTokenTransfer()` function.

PLP.sol

```
56 function _beforeTokenTransfer(
57   address from,
58   address to,
59   uint256 amount
60 ) internal override {
61   if (whitelist[from] || whitelist[to]) return;
```

```
62
63    uint256 cooldownExpireAt = cooldown[from];
64    if (amount > 0 && block.timestamp < cooldownExpireAt)
65      revert PLP_Cooldown(cooldownExpireAt);
66  }
```

However, the user can supply any receiver address while calling the `addLiquidity()` function. Then the PLP token will be minted to the receiver's address at line 197.

LiquidityFacet.sol

```
167 function addLiquidity(
168   address account,
169   address token,
170   address receiver
171 ) external nonReentrant allowed(account) returns (uint256) {
172   // LOAD poolV1 diamond storage
173   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
174     .poolV1DiamondStorage();
175
176   // Pull tokens
177   uint256 amount = LibPoolV1.pullTokens(token);
178
179   // Check
180   if (!LibPoolConfigV1.isAcceptToken(token)) revert LiquidityFacet_BadToken();
181   if (amount == 0) revert LiquidityFacet_BadAmount();
182
183   LibPoolV1.realizedFarmPnL(token);
184
185   uint256 aum = GetterFacetInterface(address(this)).getAumE18(true);
186   uint256 lpSupply = poolV1ds.plp.totalSupply();
187
188   uint256 usdDebt = _join(
189     token,
190     amount,
191     receiver,
192     account,
193     LiquidityAction.ADD_LIQUIDITY
194   );
195   uint256 mintAmount = aum == 0 ? usdDebt : (usdDebt * lpSupply) / aum;
196
197   poolV1ds.plp.mint(receiver, mintAmount);
198
199   emit AddLiquidity(
200     account,
201     token,
202     amount,
203     aum,
```

```
204       lpSupply,
205       usdDebt,
206       mintAmount
207     );
208
209     return mintAmount;
210   }
```

This allows anyone to use the `addLiquidity()` function to reset the PLP transfer cooldown of any address.

5.18.2. Remediation

Inspex suggests allowing the users to add liquidity for themselves only to prevent this attack. For example, removing the receiver parameter in the `addLiquidity()` function and minting the PLP token to the account instead.

LiquidityFacet.sol

```
167 function addLiquidity(
168   address account,
169   address token
170 ) external nonReentrant allowed(account) returns (uint256) {
171   // LOAD poolV1 diamond storage
172   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
173     .poolV1DiamondStorage();
174
175   // Pull tokens
176   uint256 amount = LibPoolV1.pullTokens(token);
177
178   // Check
179   if (!LibPoolConfigV1.isAcceptToken(token)) revert LiquidityFacet_BadToken();
180   if (amount == 0) revert LiquidityFacet_BadAmount();
181
182   LibPoolV1.realizedFarmPnL(token);
183
184   uint256 aum = GetterFacetInterface(address(this)).getAumE18(true);
185   uint256 lpSupply = poolV1ds.plp.totalSupply();
186
187   uint256 usdDebt = _join(
188     token,
189     amount,
190     account,
191     account,
192     LiquidityAction.ADD_LIQUIDITY
193   );
194   uint256 mintAmount = aum == 0 ? usdDebt : (usdDebt * lpSupply) / aum;
195
196   poolV1ds.plp.mint(account, mintAmount);
```

```
197
198   emit AddLiquidity(
199     account,
200     token,
201     amount,
202     aum,
203     lpSupply,
204     usdDebt,
205     mintAmount
206   );
207
208   return mintAmount;
209 }
```

5.19. Improper Parameter Control of Calculation Parameter

ID	IDX-019
Target	FeedableRewarder WFeedableRewarder
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Low **Impact:** Low The reward rate can be changed to any desired amount. The users can increase the reward rate to a high amount, allowing everyone to claim their entire reward in a single block. **Likelihood:** Medium The functions that can manipulate the reward cannot be called by anyone directly, but only by the feeder role, which can be set by the contract owner.
Status	Resolved The Perp88 team has resolved this issue by implementing constraint values for the duration value, assuring users that the feeding period will never be less than or greater than the constraint values.

5.19.1. Description

The `FeedableRewarder` and `WFeedableRewarder` contracts are rewarder contracts, in which there is a feeder who has a responsibility to feed the reward into the contracts. The rate of the reward emission per time unit is stored in the `rewardRate` state of each rewarder contract. Normally, the value of the `rewardRate` state will have the whole reward amount distributed completely at the time on the `rewardRateExpiredAt` state.

The `_feed()` functions in `FeedableRewarder` and `WFeedableRewarder` contracts can indirectly set their `rewardRate` state. For example, on the `FeedableRewarder` contract, in line 179, the `rewardRate` state is calculated from the `leftOverReward` divided by the value of the `duration` variable, which is one of the functions' parameters.

If the `duration` parameter is set to `1`, it means that the whole reward amount (`leftOverReward` + `feedAmount`) will be distributed within the next 1 second but the user's share amount is still the same.

FeedableRewarder.sol

```
152 function _feed(uint256 feedAmount, uint256 duration) internal {
153   uint256 totalShare = _totalShare();
```

```
154   _forceUpdateRewardCalculationParams(totalShare);
155
156   {
157     // Transfer token, with decay check
158     uint256 balanceBefore = IERC20Upgradeable(rewardToken).balanceOf(
159       address(this)
160     );
161     IERC20Upgradeable(rewardToken).safeTransferFrom(
162       msg.sender,
163       address(this),
164       feedAmount
165     );
166
167     if (
168       IERC20Upgradeable(rewardToken).balanceOf(address(this)) -
169         balanceBefore !=
170       feedAmount
171     ) revert FeedableRewarderError_FeedAmountDecayed();
172   }
173
174   uint256 leftOverReward = rewardRateExpiredAt > block.timestamp
175     ? (rewardRateExpiredAt - block.timestamp) * rewardRate
176     : 0;
177   uint256 totalRewardAmount = leftOverReward + feedAmount;
178
179   rewardRate = totalRewardAmount / duration;
180   rewardRateExpiredAt = block.timestamp + duration;
181
182   emit LogFeed(feedAmount, rewardRate, rewardRateExpiredAt);
183 }
```

The `_feed()` function is an internal function that can only be called through the `feed()` or `feedWithExpiredAt()` functions. However, these functions have the `onlyFeeder` modifier. So, it depends on the contract owner which address has the `feeder` role.

FeedableRewarder.sol

```
137 function feed(uint256 feedAmount, uint256 duration) external onlyFeeder {
138   _feed(feedAmount, duration);
139 }
140
141 function feedWithExpiredAt(uint256 feedAmount, uint256 expiredAt)
142   external
143   onlyFeeder
144 {
145   _feed(feedAmount, expiredAt - block.timestamp);
146 }
```

5.19.2. Remediation

Inspex suggests having constraints for the reward `duration` parameter to guarantee the users that the feeding period will never be less or more than the constraint values.

FeedableRewarder.sol

```
152 uint256 public constant MINIMUM_PERIOD = 7 days;    // This is only an example
    value. You can adjust it to suit the business.
153 uint256 public constant MAXIMUM_PERIOD = 365 days;  // This is only an example
    value. You can adjust it to suit the business.
154
155 error FeedableRewarderError_InvalidDuration();
156
157 function _feed(uint256 feedAmount, uint256 duration) internal {
158   if (duration < MINIMUM_PERIOD || duration > MAXIMUM_PERIOD) revert
    FeedableRewarderError_InvalidDuration();
159   uint256 totalShare = _totalShare();
160   _forceUpdateRewardCalculationParams(totalShare);
161
162   {
163     // Transfer token, with decay check
164     uint256 balanceBefore = IERC20Upgradeable(rewardToken).balanceOf(
165       address(this)
166     );
167     IERC20Upgradeable(rewardToken).safeTransferFrom(
168       msg.sender,
169       address(this),
170       feedAmount
171     );
172
173     if (
174       IERC20Upgradeable(rewardToken).balanceOf(address(this)) -
175         balanceBefore !=
176       feedAmount
177     ) revert FeedableRewarderError_FeedAmountDecayed();
178   }
179
180   uint256 leftOverReward = rewardRateExpiredAt > block.timestamp
181     ? (rewardRateExpiredAt - block.timestamp) * rewardRate
182     : 0;
183   uint256 totalRewardAmount = leftOverReward + feedAmount;
184
185   rewardRate = totalRewardAmount / duration;
186   rewardRateExpiredAt = block.timestamp + duration;
187
188   emit LogFeed(feedAmount, rewardRate, rewardRateExpiredAt);
189 }
```

It goes the same as the `WFeedableRewarder` contract.

WFeedableRewarder.sol

```
158 uint256 public constant MINIMUM_PERIOD = 7 days;    // This is only an example
    value. You can adjust it to suit the business.
159 uint256 public constant MAXIMUM_PERIOD = 365 days;  // This is only an example
    value. You can adjust it to suit the business.
160
161 error WFeedableRewarderError_InvalidDuration();
162
163 function _feed(uint256 feedAmount, uint256 duration) internal {
164   if (duration < MINIMUM_PERIOD || duration > MAXIMUM_PERIOD) revert
    WFeedableRewarderError_InvalidDuration();
165   uint256 totalShare = _totalShare();
166   _forceUpdateRewardCalculationParams(totalShare);
167
168   {
169     // Transfer token, with decay check
170     uint256 balanceBefore = IERC20Upgradeable(rewardToken).balanceOf(
171       address(this)
172     );
173     IERC20Upgradeable(rewardToken).safeTransferFrom(
174       msg.sender,
175       address(this),
176       feedAmount
177     );
178
179     if (
180       IERC20Upgradeable(rewardToken).balanceOf(address(this)) -
181         balanceBefore !=
182       feedAmount
183     ) revert WFeedableRewarderError_FeedAmountDecayed();
184   }
185
186   uint256 leftOverReward = rewardRateExpiredAt > block.timestamp
187     ? (rewardRateExpiredAt - block.timestamp) * rewardRate
188     : 0;
189   uint256 totalRewardAmount = leftOverReward + feedAmount;
190
191   rewardRate = totalRewardAmount / duration;
192   rewardRateExpiredAt = block.timestamp + duration;
193
194   emit LogFeed(feedAmount, rewardRate, rewardRateExpiredAt);
195 }
```

5.20. Incorrect Order Book Swap Rate

ID	IDX-020
Target	OrderBook
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Low **Impact:** Low When the three paths swap order is executed, the swap rate will always be worse than the trigger rate. **Likelihood:** Medium This issue only occurs when the swap order that was created with three paths is triggered and executed.
Status	Resolved The Perp88 team has resolved this issue by adding the minimum and maximum price of token B in the `currentRatio` value calculation to adjust the price validation in the `validateSwapOrderPriceWithTriggerAboveThreshold()` function for three path swap orders.

5.20.1. Description

The `createSwapOrder()` function allows users to create a swap order from token A to token B that will be triggered when the token price hits the provided `_triggerRatio` (the price of token B / price of token A). The swap order can be created with 2-3 token paths as shown in line 322.

OrderBook.sol

```
312 function createSwapOrder(
313   address[] memory _path,
314   uint256 _amountIn,
315   uint256 _minOut,
316   uint256 _triggerRatio, // tokenB / tokenA
317   bool _triggerAboveThreshold,
318   uint256 _executionFee,
319   bool _shouldWrap,
320   bool _shouldUnwrap
321 ) external payable nonReentrant {
322   if (_path.length != 2 && _path.length != 3) revert InvalidPathLength();
323   if (_path[0] == _path[_path.length - 1]) revert InvalidPath();
324   if (_amountIn == 0) revert InvalidAmountIn();
325   if (_executionFee < minExecutionFee) revert InsufficientExecutionFee();
```

```
326
327   // always need this call because of mandatory executionFee user has to
      transfer in BNB
328   _transferInETH();
329
330   if (_shouldWrap) {
331     if (_path[0] != weth) revert OnlyNativeShouldWrap();
332     if (msg.value != _executionFee + _amountIn)
333       revert IncorrectValueTransfer();
334   } else {
335     if (msg.value != _executionFee) revert IncorrectValueTransfer();
336     IERC20Upgradeable(_path[0]).safeTransferFrom(
337       msg.sender,
338       address(this),
339       _amountIn
340     );
341   }
342
343   _createSwapOrder(
344     msg.sender,
345     _path,
346     _amountIn,
347     _minOut,
348     _triggerRatio,
349     _triggerAboveThreshold,
350     _shouldUnwrap,
351     _executionFee
352   );
353 }
```

When the price hits the trigger, the `executeSwapOrder()` function will be called by the whitelisted address to process the swap order. The `validateSwapOrderPriceWithTriggerAboveThreshold()` function is also called in line 479 to validate that the current price matches the criteria.

OrderBook.sol

```
467 function executeSwapOrder(
468   address _account,
469   uint256 _orderIndex,
470   address payable _feeReceiver
471 ) external nonReentrant whitelisted {
472   SwapOrder memory order = swapOrders[_account][_orderIndex];
473   if (order.account == address(0)) revert NonExistentOrder();
474
475   if (order.triggerAboveThreshold) {
476     // gas optimisation
477     // order.minAmount should prevent wrong price execution in case of simple
```

```
limit order
    if (
      !validateSwapOrderPriceWithTriggerAboveThreshold(
        order.path,
        order.triggerRatio
      )
    ) revert InvalidPriceForExecution();
  }

  delete swapOrders[_account][_orderIndex];

  IERC20Upgradeable(order.path[0]).safeTransfer(pool, order.amountIn);

  uint256 _amountOut;
  if (order.path[order.path.length - 1] == weth && order.shouldUnwrap) {
    _amountOut = _swap(
      order.account,
      order.path,
      order.minOut,
      address(this)
    );
    _transferOutETH(_amountOut, payable(order.account));
  } else {
    _amountOut = _swap(
      order.account,
      order.path,
      order.minOut,
      order.account
    );
  }

  // pay executor
  _transferOutETH(order.executionFee, _feeReceiver);

  emit ExecuteSwapOrder(
    _account,
    _orderIndex,
    order.path,
    order.amountIn,
    order.minOut,
    order.triggerRatio,
    order.triggerAboveThreshold,
    order.shouldUnwrap,
    order.executionFee,
    _amountOut
  );
}
```

The `_swap()` function handles the swapping logic with three paths by calling `_vaultSwap()` function twice. The min source token price and the max destination token price will be used to calculate the amount out of the token for each `_vaultSwap()` function call.

OrderBook.sol

```
1056 function _swap(
1057   address _account,
1058   address[] memory _path,
1059   uint256 _minOut,
1060   address _receiver
1061 ) private returns (uint256) {
1062   if (_path.length == 2) {
1063     return _vaultSwap(_account, _path[0], _path[1], _minOut, _receiver);
1064   }
1065   if (_path.length == 3) {
1066     uint256 midOut = _vaultSwap(
1067       _account,
1068       _path[0],
1069       _path[1],
1070       0,
1071       address(this)
1072     );
1073     IERC20Upgradeable(_path[1]).safeTransfer(pool, midOut);
1074     return _vaultSwap(_account, _path[1], _path[2], _minOut, _receiver);
1075   }
1076
1077   revert("OrderBook: invalid _path.length");
1078 }
```

However, in the `validateSwapOrderPriceWithTriggerAboveThreshold()` function there is only a check between the source and the destination token prices even for the three paths swap order as shown in lines 427-437.

OrderBook.sol

```
418 function validateSwapOrderPriceWithTriggerAboveThreshold(
419   address[] memory _path,
420   uint256 _triggerRatio
421 ) public view returns (bool) {
422   if (_path.length != 2 && _path.length != 3) revert InvalidPathLength();
423
424   // limit orders don't need this validation because minOut is enough
425   // so this validation handles scenarios for stop orders only
426   // when a user wants to swap when a price of tokenB increases relative to
    tokenA
427   address tokenA = _path[0];
428   address tokenB = _path[_path.length - 1];
```

```
429   uint256 tokenAPrice;
430   uint256 tokenBPrice;
431
432   tokenAPrice = poolOracle.getMinPrice(tokenA);
433   tokenBPrice = poolOracle.getMaxPrice(tokenB);
434
435   uint256 currentRatio = (tokenBPrice * PRICE_PRECISION) / tokenAPrice;
436
437   bool isValid = currentRatio > _triggerRatio;
438   return isValid;
439 }
```

This results in a bad swap rate for the three paths swap order when the min and max prices of the second token are different. Since the second token will be both the destination token of the first swap and the source token of the second swap.

5.20.2. Remediation

Inspex suggests implementing the proper price validation in the `validateSwapOrderPriceWithTriggerAboveThreshold()` function for three paths swap order. For example, when swapping from token A > token B > token C, the min and max price of token B should be included in the `currentRatio` value calculation as shown in lines 437-452:

OrderBook.sol

```
418 function validateSwapOrderPriceWithTriggerAboveThreshold(
419   address[] memory _path,
420   uint256 _triggerRatio
421 ) public view returns (bool) {
422   if (_path.length != 2 && _path.length != 3) revert InvalidPathLength();
423
424   // limit orders don't need this validation because minOut is enough
425   // so this validation handles scenarios for stop orders only
426   // when a user wants to swap when a price of tokenB increases relative to
    tokenA
427   if (_path.length == 2) {
428     address tokenA = _path[0];
429     address tokenB = _path[_path.length - 1];
430     uint256 tokenAPrice;
431     uint256 tokenBPrice;
432
433     tokenAPrice = poolOracle.getMinPrice(tokenA);
434     tokenBPrice = poolOracle.getMaxPrice(tokenB);
435
436     uint256 currentRatio = (tokenBPrice * PRICE_PRECISION) / tokenAPrice;
437   } else {
438     address tokenA = _path[0];
```

```
439         address tokenB = _path[1];
440         address tokenC = _path[2];
441         uint256 tokenAPrice;
442         uint256 tokenBMinPrice;
443         uint256 tokenBMaxPrice;
444         uint256 tokenCPrice;
445 
446         tokenAPrice = poolOracle.getMinPrice(tokenA);
447         tokenBMinPrice = poolOracle.getMinPrice(tokenB);
448         tokenBMaxPrice = poolOracle.getMaxPrice(tokenB);
449         tokenCPrice = poolOracle.getMaxPrice(tokenC);
450 
451         uint256 currentRatio = (tokenCPrice * tokenBMaxPrice * PRICE_PRECISION) /
    (tokenAPrice * tokenBMinPrice);
452       }
453       bool isValid = currentRatio > _triggerRatio;
454       return isValid;
455     }
```

5.21. Reward Loss When Using onWithdraw() Function

ID	IDX-021
Target	AdHocMintRewarder
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Low **Impact:** Medium The reward will be reset when the `onWithdraw()` function is called without harvesting the reward for the user. **Likelihood:** Low This issue only occurs when a staking contract calls the `onWithdraw()` function without first calling the `onHarvest()` function to harvest the user reward.
Status	Resolved * The Perp88 team has clarified that, according to the Perp88 business requirements, the `PLPStaking` contract will never add the `AdhocMintRewarder` contract as the rewarder, so this issue will not happen.

5.21.1. Description

Normally, when the users harvest their reward, the `onHarvest()` function will be called. After the reward tokens are minted and transferred to the users, the `userAccRewards` will be set to 0 as shown in line 88.

AdHocMintRewarder.sol

```
80  function onHarvest(address user, address receiver)
81    external
82    onlyStakingContract
83  {
84    uint256 pendingRewardAmount = _pendingReward(user);
85
86    // Reset user reward accumulation.
87    // The next action will start accum reward from zero again.
88    userAccRewards[user] = 0;
89    userLastRewards[user] = block.timestamp.toUint64();
90
91    if (pendingRewardAmount != 0) {
92      _harvestToken(receiver, pendingRewardAmount);
93    }
94
95    emit LogHarvest(user, pendingRewardAmount);
96  }
```

The reward amount is calculated from accumulated reward since the last action sum with the jotted rewards from the past (`calculateUserAccReward(user)` + `userAccRewards[user]` in the `_pendingReward()`) function.

AdHocMintRewarder.sol

```
102 function _pendingReward(address user) internal view returns (uint256) {
103   // (accumulated reward since the last action) + (jotted reward from the past)
104   return _calculateUserAccReward(user) + userAccRewards[user];
105 }
```

However, the user's reward is reset when the `onWithdraw()` function in the `AdHocMintRewarder` contract is executed after withdrawing staked tokens from the platform. The user's reward is reset in line 75 by the `onWithdraw()` function without being harvested. As a result, users lose their jotted past rewards.

AdHocMintRewarder.sol

```
69 function onWithdraw(address user, uint256 shareAmount)
70   external
71   onlyStakingContract
72 {
73   // Reset user reward
74   // The rule is whenever withdraw occurs, no matter the size, reward
   calculation should restart.
75   userAccRewards[user] = 0;
76   userLastRewards[user] = block.timestamp.toUint64();
77   emit LogOnWithdraw(user, shareAmount);
78 }
```

5.21.2. Remediation

Inspex suggests accumulating the user reward when the user withdraws tokens from the platform instead and resetting it only when the user harvests the reward in order to prevent losing the user accumulated reward.

AdHocMintRewarder.sol

```
69 function onWithdraw(address user, uint256 shareAmount)
70   external
71   onlyStakingContract
72 {
73   userAccRewards[user] += _calculateUserAccReward(user);
74   userLastRewards[user] = block.timestamp.toUint64();
75   emit LogOnWithdraw(user, shareAmount);
76 }
```

5.22. Insufficient Logging for Privileged Functions

ID	IDX-022
Target	Compounder PLPStaking FarmFacet FeedableRewarder WFeedableRewarder
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Privileged functions' executions cannot be monitored easily by the users. **Likelihood:** Low It is not likely that the execution of the privileged functions will be a malicious action.
Status	Resolved The Perp88 team has resolved this issue by adding the event emitting to the listed functions.

5.22.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts on the platform.

For example, the `onlyOwner` role can call the `setStrategyOf()` function to add a new strategy or changing strategy to the `poolConfigDs.pendingStrategyOf[token]` at line 67, and no events are emitted.

FarmFacet.sol

```
54  function setStrategyOf(address token, StrategyInterface newStrategy)
55    external
56    onlyOwner
57  {
58    // Load PoolConfig Diamond storage
59    LibPoolConfigV1.PoolConfigV1DiamondStorage
60      storage poolConfigDs = LibPoolConfigV1.poolConfigV1DiamondStorage();
61
62    LibPoolConfigV1.StrategyData memory strategyData = poolConfigDs
63      .strategyDataOf[token];
64    StrategyInterface pendingStrategy = poolConfigDs.pendingStrategyOf[token];
```

```
65    if (strategyData.startTimestamp == 0 || pendingStrategy != newStrategy) {
66      // When adding new strategy or changing strategy
67      poolConfigDs.pendingStrategyOf[token] = newStrategy;
68      strategyData.startTimestamp = uint64(block.timestamp + STRATEGY_DELAY);
69    } else {
70      // When committing a new strategy
71      if (
72        strategyData.startTimestamp == 0 ||
73        block.timestamp < strategyData.startTimestamp
74      ) revert FarmFacet_TooEarlyToCommitStrategy();
75      if (address(poolConfigDs.strategyOf[token]) != address(0)) {
76        // If there is previous strategy, we need to withdraw all funds from it
77        int256 balanceChange = poolConfigDs.strategyOf[token].exit(
78          strategyData.principle
79        );
80        // Update totalOf[token] to sync physical balance with pool state
81        LibPoolV1.updateTotalOf(token);
82        // Realized profits/losses
83        if (balanceChange > 0) {
84          uint256 profit = uint256(balanceChange);
85          LibPoolV1.increasePoolLiquidity(token, profit);
86
87          emit StrategyRealizedProfit(token, profit);
88        } else if (balanceChange < 0) {
89          uint256 loss = uint256(-balanceChange);
90          LibPoolV1.decreasePoolLiquidity(token, loss);
91
92          emit StrategyRealizedLoss(token, loss);
93        }
94
95        emit StrategyDivest(token, strategyData.principle);
96      }
97      // Commit new strategy
98      poolConfigDs.strategyOf[token] = newStrategy;
99      strategyData.startTimestamp = 0;
100     strategyData.principle = 0;
101     poolConfigDs.pendingStrategyOf[token] = StrategyInterface(address(0));
102
103     emit SetStrategy(token, newStrategy);
104   }
105   poolConfigDs.strategyDataOf[token] = strategyData;
106 }
```

The privileged functions without sufficient logging are as follows:

File	Contract	Function	Modifier
Compounder.sol (L: 36)	Compounder	addToken()	onlyOwner
Compounder.sol (L: 51)	Compounder	removeToken()	onlyOwner
Compounder.sol (L: 69)	Compounder	setCompoundToken()	onlyOwner
PLPStaking.sol (L: 39)	PLPStaking	addStakingToken()	onlyOwner
PLPStaking.sol (L: 53)	PLPStaking	addRewarder()	onlyOwner
PLPStaking.sol (L: 76)	PLPStaking	setCompounder()	onlyOwner
FarmFacet.sol (L: 54)	FarmFacet	setStrategyOf()	onlyOwner
FeedableRewarder.sol (L: 148)	FeedableRewarder	setFeeder()	onlyOwner
WFeedableRewarder.sol (L: 154)	WFeedableRewarder	setFeeder()	onlyOwner

5.22.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

FarmFacet.sol

```
54  event SetPendingStrategyOf(StrategyInterface newStrategy);
55  function setStrategyOf(address token, StrategyInterface newStrategy)
56    external
57    onlyOwner
58  {
59    // Load PoolConfig Diamond storage
60    LibPoolConfigV1.PoolConfigV1DiamondStorage
61      storage poolConfigDs = LibPoolConfigV1.poolConfigV1DiamondStorage();
62
63    LibPoolConfigV1.StrategyData memory strategyData = poolConfigDs
64      .strategyDataOf[token];
65    StrategyInterface pendingStrategy = poolConfigDs.pendingStrategyOf[token];
66    if (strategyData.startTimestamp == 0 || pendingStrategy != newStrategy) {
67      // When adding new strategy or changing strategy
68      poolConfigDs.pendingStrategyOf[token] = newStrategy;
69      strategyData.startTimestamp = uint64(block.timestamp + STRATEGY_DELAY);
70      emit SetPendingStrategyOf(newStrategy);
71    } else {
72      // When committing a new strategy
73      if (
74        strategyData.startTimestamp == 0 ||
75        block.timestamp < strategyData.startTimestamp
```

```
76      ) revert FarmFacet_TooEarlyToCommitStrategy();
77      if (address(poolConfigDs.strategyOf[token]) != address(0)) {
78        // If there is previous strategy, we need to withdraw all funds from it
79        int256 balanceChange = poolConfigDs.strategyOf[token].exit(
80          strategyData.principle
81        );
82        // Update totalOf[token] to sync physical balance with pool state
83        LibPoolV1.updateTotalOf(token);
84        // Realized profits/losses
85        if (balanceChange > 0) {
86          uint256 profit = uint256(balanceChange);
87          LibPoolV1.increasePoolLiquidity(token, profit);
88
89          emit StrategyRealizedProfit(token, profit);
90        } else if (balanceChange < 0) {
91          uint256 loss = uint256(-balanceChange);
92          LibPoolV1.decreasePoolLiquidity(token, loss);
93
94          emit StrategyRealizedLoss(token, loss);
95        }
96
97        emit StrategyDivest(token, strategyData.principle);
98      }
99      // Commit new strategy
100     poolConfigDs.strategyOf[token] = newStrategy;
101     strategyData.startTimestamp = 0;
102     strategyData.principle = 0;
103     poolConfigDs.pendingStrategyOf[token] = StrategyInterface(address(0));
104
105     emit SetStrategy(token, newStrategy);
106   }
107   poolConfigDs.strategyDataOf[token] = strategyData;
108 }
```

5.23. Missing Duplication Check

ID	IDX-023
Target	PLPStaking
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Very Low **Impact:** Low Due to the duplicate staking token, the user share value is calculated improperly, which has an impact on the user's cumulative reward. **Likelihood:** Low Only the contract owner could add duplicate staking tokens and rewarders into the contract.
Status	Resolved The Perp88 team has resolved this issue by validating the duplicated before adding the new token and new rewarder.

5.23.1. Description

The `PLPStaking` contract's `addStakingToken()` and `addRewarder()` functions are used to add new staking tokens and rewarders to the pool. However, in both functions, there is no duplication check for the staking token and the rewarder.

PLPStaking.sol

```
39 function addStakingToken(address newToken, address[] memory newRewarders)
40   external
41   onlyOwner
42 {
43   uint256 length = newRewarders.length;
44   for (uint256 i = 0; i < length; ) {
45     _updatePool(newToken, newRewarders[i]);
46
47     unchecked {
48       ++i;
49     }
50   }
51 }
52
53 function addRewarder(address newRewarder, address[] memory newTokens)
54   external
55   onlyOwner
```

```
56  {
57    uint256 length = newTokens.length;
58    for (uint256 i = 0; i < length; ) {
59      _updatePool(newTokens[i], newRewarder);
60
61      unchecked {
62        ++i;
63      }
64    }
65  }
```

In the case that the owner unintentionally adds a duplicate staking token or rewarder to the pool, when the user calls the `deposit()` function to deposit the staking token that has been added twice by the owner, the staking token will be added to both pools.

PLPStaking.sol

```
80  function deposit(
81      address to,
82      address token,
83      uint256 amount
84    ) external {
85      if (!isStakingToken[token]) revert PLPStaking_UnknownStakingToken();
86
87      uint256 length = stakingTokenRewarders[token].length;
88      for (uint256 i = 0; i < length; ) {
89        address rewarder = stakingTokenRewarders[token][i];
90
91        IRewarder(rewarder).onDeposit(to, amount);
92
93        unchecked {
94          ++i;
95        }
96      }
97
98      userTokenAmount[token][to] += amount;
99      IERC20Upgradeable(token).safeTransferFrom(
100       msg.sender,
101       address(this),
102       amount
103     );
104
105     emit LogDeposit(msg.sender, to, token, amount);
106   }
```

The user's cumulative reward must be calculated using the user share value. The `calculateShare()` function used to calculate the share value will add the duplicate token twice, leading to a miscalculation on

the user's cumulative reward.

PLPStaking.sol

```
179 function calculateShare(address rewarder, address user)
180   external
181   view
182   returns (uint256)
183 {
184   address[] memory tokens = rewarderStakingTokens[rewarder];
185   uint256 share = 0;
186   uint256 length = tokens.length;
187   for (uint256 i = 0; i < length; ) {
188     share += userTokenAmount[tokens[i]][user];
189
190     unchecked {
191       ++i;
192     }
193   }
194   return share;
195 }
```

5.23.2. Remediation

Inspex suggests validating the duplicate staking token and rewarder before adding them to the pool. For example, add a mapping to keep track of the pair of the added token and the added rewarder. Then, check the mapping in the `_updatePool()` function to prevent the added token and rewarder from being added again.

PLPStaking.sol

```
67 mapping(address => mapping(address => boolean)) public
   tokenRewarderAvailability;
68 error PLPStaking_DuplicatedTokenRewarder();
69
70 function _updatePool(address newToken, address newRewarder) internal {
71   if (tokenRewarderAvailability[newToken][newRewarder]) revert
   PLPStaking_DuplicatedTokenRewarder();
72   stakingTokenRewarders[newToken].push(newRewarder);
73   rewarderStakingTokens[newRewarder].push(newToken);
74   isStakingToken[newToken] = true;
75   if (!isRewarder[newRewarder]) {
76     isRewarder[newRewarder] = true;
77   }
78   tokenRewarderAvailability[newToken][newRewarder] = true;
79 }
```

5.24. Use of Deprecated Function

ID	IDX-024
Target	PoolOracle
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Perp88 team has resolved this issue by changing the deprecated function to the current available function.

5.24.1. Description

According to the Chainlink's Data Feeds API (https://docs.chain.link/docs/data-feeds/price-feeds/api-reference/), the `latestRound()` and `latestAnswer()` function calls are deprecated. There are several deprecated calls in the `PoolOracle` contract, for example, in the `_getPrice()` function, the deprecated functions are executed at lines 60 and 66.

PoolOracle.sol

```
47 function _getPrice(address token, bool isUseMaxPrice)
48   internal
49   view
50   returns (uint256)
51 {
52   // SLOAD
53   PriceFeedInfo memory priceFeed = priceFeedInfo[token];
54   if (address(priceFeed.priceFeed) == address(0))
55     revert PoolOracle_PriceFeedNotAvailable();
56
57   uint256 price = 0;
58   int256 _priceCursor = 0;
59   uint256 priceCursor = 0;
60   uint80 latestRoundId = priceFeed.priceFeed.latestRound();
61
62   for (uint80 i = 0; i < roundDepth; i++) {
63     if (i >= latestRoundId) break;
64
65     if (i == 0) {
```

```
66        priceCursor = priceFeed.priceFeed.latestAnswer().toUint256();
67      } else {
68        (, _priceCursor, , , ) = priceFeed.priceFeed.getRoundData(
69          latestRoundId - i
70        );
71        priceCursor = _priceCursor.toUint256();
72      }
73
74      if (price == 0) {
75        price = priceCursor;
76        continue;
77      }
78
79      if (isUseMaxPrice && price < priceCursor) {
80        price = priceCursor;
81        continue;
82      }
83
84      if (!isUseMaxPrice && price > priceCursor) {
85        price = priceCursor;
86      }
87    }
88
89    if (price == 0) revert PoolOracle_UnableFetchPrice();
90
91    price = (price * PRICE_PRECISION) / 10**priceFeed.decimals;
92
93    // Handle strict stable price deviation.
94    if (priceFeed.isStrictStable) {
95      uint256 delta;
96      unchecked {
97        delta = price > ONE_USD ? price - ONE_USD : ONE_USD - price;
98      }
99
100     if (delta <= maxStrictPriceDeviation) return ONE_USD;
101
102     if (isUseMaxPrice && price > ONE_USD) return price;
103
104     if (!isUseMaxPrice && price < ONE_USD) return price;
105
106     return ONE_USD;
107   }
108
109   // Handle spreadBasisPoint
110   if (isUseMaxPrice) return (price * (BPS + priceFeed.spreadBps)) / BPS;
111
112   return (price * (BPS - priceFeed.spreadBps)) / BPS;
```

```
113 }
```

The following table contains all targets that contain deprecated calling functions.

File	Contract	Deprecated Function
PoolOracle.sol (L: 60)	PoolOracle	latestRound()
PoolOracle.sol (L: 66)	PoolOracle	latestAnswer()
PoolOracle.sol (L: 152)	PoolOracle	latestAnswer()

However, with the current data feed implementation, these functions can still be used to get the quote price.

5.24.2. Remediation

Inspex suggests using the `latestRoundData()` function instead of the `latestRound()` and `latestAnswer()` functions. It also includes better verification information for further integrated usage.

6. Appendix

6.1. About Inspex

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

inspeX

CYBERSECURITY PROFESSIONAL SERVICE