Three Sigma

Code Audit

Singularity

Singularity

KYB/KYC'd confidential
DeFi infrastructure layer

Disclaimer

Code Audit

Singularity KYB/KYC'd confidential DeFi infrastructure layer

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Singularity KYB/KYC'd confidential DeFi infrastructure layer

Table of Contents

Summary

Code Audit

Singularity KYB/KYC'd confidential DeFi infrastructure layer

Summary

Three Sigma Labs audited Singularity in a 18 person week engagement. The audit was conducted from 26-02-2024 to 19-04-2024 and 05-06-2024 to 15-06-2024.

Protocol Description

Singularity is a KYC/KYB enabled modular confidential DeFi infrastructure integrated with major DeFi protocols so users can leverage existing on-chain liquidity while their on-chain actions remain obfuscated. Singularity primary users are institutional funds or individuals who want to preserve commercial confidentiality on-chain. Singularity uses the UltraPLONK proof system with zero-knowledge circuits based on Noir.

Scope

Code Audit

Singularity KYB/KYC'd confidential DeFi infrastructure layer

Scope

Circuits

Filepath	nLOC
circuits/curve_add_liquidity/src/main.nr	109
circuits/curve_exchange/src/main.nr	43
circuits/curve_multi_exchange/src/main.nr	43
circuits/curve_remove_liquidity/src/main.nr	77
circuits/deposit/src/main.nr	23
circuits/fuzk/src/lib.nr	213
circuits/join_split/src/main.nr	88
circuits/join/src/main.nr	77
circuits/split/src/main.nr	65
circuits/swap/src/main.nr	53
circuits/transfer/src/main.nr	52
circuits/uniswap_collect_fees/src/main.nr	53
circuits/uniswap_lp/src/main.nr	67
circuits/uniswap_remove_liquidity/src/main.nr	60
circuits/uniswap_swap/src/main.nr	41
circuits/withdraw/src/main.nr	39
circuits/zk_lock_asset/src/main.nr	28
circuits/zk_lock_note/src/main.nr	55
circuits/zk_unlock_note/main.mr	55
SUM	1241

Smart Contracts

Filepath	nLOC
contracts/core/base/BaseAssetManager.sol	205
contracts/core/base/BaseAssetPool.sol	39
contracts/core/base/BaseInputBuilder.sol	10
contracts/core/DarkpoolAssetManager.sol	346
contracts/core/DarkpoolInputBuilder.sol	121
contracts/core/FeeManager.sol	110

Filepath	nLOC
contracts/core/KeyringManager.sol	48
contracts/core/MerkleTreeOperator.sol	150
contracts/core/pools/ERC20AssetPool.sol	32
contracts/core/pools/ERC721AssetPool.sol	37
contracts/core/pools/ETHAssetPool.sol	22
contracts/core/RelayerHub.sol	26
contracts/core/VerifierHub.sol	56
contracts/defi/curve/CurveAssetManagerHelper.sol	70
contracts/defi/curve/CurveCPAssetManager.sol	571
contracts/defi/curve/CurveInputBuilder.sol	103
contracts/defi/curve/CurveMultiExchangeAssetManager.sol	203
contracts/defi/curve/CurveSingleExchangeAssetManager.sol	180
contracts/defi/curve/CurveSLPAssetManager.sol	597
contracts/defi/curve/CurveSPPAssetManager.sol	557
contracts/defi/uniswap/libraries/PoolAddress.sol	40
contracts/defi/uniswap/UniswapAssetManager.sol	831
contracts/defi/uniswap/UniswapInputBuilder.sol	107
contracts/staking/StakingAssetManager.sol	343
contracts/staking/StakingOperator.sol	76
contracts/staking/StakingInputBuilder.sol	69
contracts/staking/ZKToken.sol	31
SUM	4980

Assumptions

External libraries such as OpenZeppelin are considered safe.

Methodology

Code Audit

Singularity KYB/KYC'd confidential DeFi infrastructure layer

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Singularity KYB/KYC'd confidential DeFi infrastructure layer

Project Dashboard

Application Summary

Name	Singularity
Commit	4018a576c5f38a2dbf5ef128bc33195f37827273 16dc59f060f16a9d281f5bb1501205fbffa4562a
Language	Solidity and Noir
Platform	Ethereum

Engagement Summary

Timeline	26-02-2024 to 19-04-2024 and 05-06-2024 to 15-06-2024
№ of Auditors	2
Review Time	18 person weeks

Issue Classification	Found	Addressed	Acknowledged
Critical	7	7	0
High	10	10	0
Medium	5	3	2
Low	9	7	2
None	11	11	0

Category Breakdown

Suggestion	8
Documentation	0
Bug	28
Optimization	3
Good Code Practices	3

Code Maturity Evaluation

Code Audit

Singularity KYB/KYC'd confidential DeFi infrastructure layer

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 as well as takes the correct measures in rounding the results of arithmetic operations.
Centralization	Satisfactory. The owner does not have significant control over the funds.
Code Stability	Satisfactory. No new functionality was being added throughout the audit.
Upgradeability	Weak. The protocol is not upgradeable.
Function Composition	Moderate. Functionality was well split into functions, but the codebase has significant code duplication.
Front-Running	Satisfactory. No significant frontrunning opportunities were found.
Monitoring	Satisfactory. Events are correctly emitted for the most significant state changes.
Specification	Satisfactory. The code matched the design specifications.
Testing and Verification	Moderate. Unit tests were present for most functionality but fuzzing and invariant tests could be performed.

Findings

Code Audit

Singularity KYB/KYC'd confidential DeFi infrastructure layer

Findings

3S-SG-C01

In **`StakingAssetManager::lockERC20()`** the resulting note is created with asset instead of **`zkToken`**

Id	3S-SG-C01
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in #ab8b4b9.

Description

Users can transfer tokens to the **`StakingAssetManager`** in return for a note commitment of the corresponding **`zkToken`**. However, **`StakingAssetManager::lockERC20()`** builds the note using **`args.asset`**, which is the asset itself, which will lead to problems (will not be able to unlock).

Recommendation

Build the note using **`zkToken`** instead.

3S-SG-C02

MerkleRoot is not validated in all **`StakingAssetManager`** functions

Id	3S-SG-C02
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in #5520165.

Description

All functions that require a merkle root to verify note inclusion must validate that the merkle root is valid. Otherwise, attackers can forge a merkle root that includes a fake note and steal all assets.

Recommendation

Validate that the provided Merkle Root is valid.

3S-SG-C03

Checks-effects-interations pattern is not always followed, which can be used to drain all tokens

Id	3S-SG-C03
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in #8b300f0.

Description

Here is a writeup about the pattern. Essentially state changes should be handled before interacting with external contracts (for example, when sending **`ETH`** via **`.call("")`**, to avoid reentrancy.

Here is an example of the code not following it. **`_postWithdraw()`** should happen before sending **`ETH`**. So a user can call **`withdrawETH()`**, get execution when the **`ETH`** is sent, use the same note commitment (still not marked used) to swap on uniswap and double spend. This can be done in a loop to drain all tokens.

Recommendation

Ensure that the codebase follows this pattern.

3S-SG-C04

Curve multi exchange does not validate **`assetIn`** and **`assetOut`** against **`route`**

Id	3S-SG-C04
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in #8b300f0.

Description

`CurveMultiExchangeAssetManager::curveMultiExchange()` receives arguments **`assetIn`**, **`assetOut`** and **`route`**. Route is defined as **`Array of [initial token, pool or zap, token, pool or zap, token, ...]`** .

The **`CurveRouter::exchange()`** assumes the **`assetIn`** is **`route[0]`** and **`assetOut`** is the last non null address in **`route`** that represents a token out. This means one of the indexes 2, 4, 6, 8, 10, depending on which one precedes a null.

Thus, knowing that this validation is missing, it's possible to specify an **`assetOut`** of a valuable token, but place in the route a fake coin. By doing so, and introducing a malicious pool to swap, the exchange contract will receive a large amount of fake coins, returning this amount, but the **`CurveMultiExchangeAssetManager`** will think it is **`assetOut`** instead.

The transaction could still revert when transferring **`assetOut`**, as the exchange did not actually send the real **`assetOut`**, but it won't due to issue #12. However, the deposit note is still created and the funds can be stolen.

Here is a poc.

Recommendation

Validate **`assetIn`** and **`assetOut`** against the route.

`assetIn` should be **`route[0]`** and **`assetOut`** should match the last non null token index, as described above.

3S-SG-C05

DarkPoolAssetManager can be drained by looping over **`join()`**, **`joinSplit()`** or **`swap()`** with only some initial amount

Id	3S-SG-C05
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in #8b300f0.

Description

DarkPoolAssetManager::join(), DarkPoolAssetManager::joinSplit() and DarkPoolAssetManager::swap() don't check if the 2 notes are the same. Thus, using only 1 note, it's possible to double the amount of funds available. When used in a loop, this enables draining the whole asset manager.

Here is the poc for **`join()`** and **`joinSplit`** and here for **`swap()`**.

Recommendation

Check if the input notes are the same.

3S-SG-C06

Stuck ETH in Curve exchanges due to sending **`msg.value`** to the exchange instead of **`amountIn`**

Id	3S-SG-C06
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in #8b300f0.

Description

The curve exchanges allow users who deposited to use their note commitments to swap for other assets. Thus, they have already deposited the **`ETH`**, but the exchanges still require the **`amountIn`** to be **`msg.value`**, leading to stuck **`ETH`**.

POC here.

Recommendation

Replace **`IExchange(exchangeContract).exchange{value: msg.value}`** by **`IExchange(exchangeContract).exchange{value: amountIn}`**.

3S-SG-C07

Reusing the same **rho** and **pubKey** in different deposits leads to lost tokens

Id	3S-SG-C07
Classification	Critical
Severity	Critical
Likelihood	Medium
Category	Bug
Status	Addressed in #8b300f0.

Description

The nullifier is formed from **rho** and the schnorr **pubKey**. This means that depositing with different assets and amounts will still have the same nullifier, leading to the inability to move the following deposits with the same **rho** and **pubKey**.

Check the poc here for details.

Recommendation

Either enforce the nullifier uniqueness (via unique **rho** and **pubkey** combination) or use the note as nullifier.

3S-SG-H01

Note footers should not be 0 in **`curveRemoveLiquidity()`** if the corresponding **`assetOut`**s are non null

Id	3S-SG-H01
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed in #8b300f0.

Description

In **`curveRemoveLiquidity()`**, the note footer should not be allowed to be 0 if **`assetOut`** is non null, as this could lead to duplicate note footers.

Recommendation

Revert if the note footer is 0 but the corresponding asset out is not.

3S-SG-H02

`UniswapLiquidityAssetManager::_validateCollectFeesArgs()` validates nullifier instead of note footer

Id	3S-SG-H02
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed in #8b300f0.

Description

`UniswapLiquidityAssetManager::_validateCollectFeesArgs()` allows using the same note footers as it checks for nullifiers, when it should check for note footers.

Recommendation

Check for note footers correctly.

3S-SG-H03

`CurveAssetManagerHelper::_validateAssets()` should check that the number of assets provided is smaller than the maximum of the pool

Id	3S-SG-H03
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed in #8b300f0.

Description

CurveAssetManagerHelper::_validateAssets() allows sending assets with indexes bigger than the maximum allowed of a pool (**`num_coins`**). This could lead to users losing tokens or unexpected behaviours.

Recommendation

Validate that the number of assets sent is at most **`num_coins`**.

3S-SG-H04

All curve params should be signed by the schnorr private key in the proof, or users may be griefed

Id	3S-SG-H04
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed in #8b300f0.

Description

An example of this issue is for example here.

If the last bit of **`isLegacy`** is **`0`**, it checks **`address(this).balance`**, otherwise it **`IERC20(_WETH_ADDRESS).balanceOf(address(this))`**. This means that if for the same pool, the wrong **`isLegacy`** is used, 0 **`outAmounts[i]`** will be recorded.

Recommendation

Either send the curve params in the proof and sign them with the schnorr private key or set a mapping from pool to the correct pool params.

3S-SG-H05

Curve multi exchange can be used to withdraw assets without paying fees

Id	3S-SG-H05
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in #8b300f0.

Description

`CurveMultiExchangeAssetManager::exchange()` allows sending any **`route`**, which is of type **`Array of [initial token, pool or zap, token, pool or zap, token, ...]`**. The **`pool or zap`** chosen indexes are not validated, which means any contract can be specified. Thus, users can create a contract with the same interface as **`pool or zap`** and receive the **`amountIn`** there. Then, only send back a very small amount of token out to bypass the fee manager check in **`calculateFee()`** of the **`amount`** being bigger than **`serviceCharge + relayerRefund`**. The service fees will be very small and the user will get the funds right away.

The poc for issue #13 contains an example of a malicious **`swap`** contract.

Recommendation

Apply the fees when releasing **`assetIn`** instead, this way it can not be gamed.

3S-SG-H06

Anyone can deposit to **`DarkpoolAssetManager`** as the **`owner`** can be freely chosen without any implication

Id	3S-SG-H06
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in #8b300f0.

Description

DarkpoolAssetManager::depositETH() and DarkpoolAssetManager::depositERC20() allow choosing any owner, while the **`msg.sender`** still receives the deposit.

Recommendation

The owner should be the **`msg.sender`** or the owner's signature should be validated (so it can be relayed).

3S-SG-H07

Attackers can include other users nullifiers to make their funds stuck when adding liquidity to curve

Id	3S-SG-H07
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed in #8b300f0.

Description

CurveLiquidityAssetManager::curveAddLiquidity() always spends the nullifiers, even if the amounts used are 0. In the circuit, if the amount is 0, it does not validate the nullifier against the user signature, making it possible to include nullifiers from other users in the same transaction and losing their funds forever.

Here is the poc.

Recommendation

In **`CurveAddLiquidityAssetManager::_addLiquidity()`** skip _postWithdraw() if the amount is **`0`**.

3S-SG-H08

`_addLiquidity()` slippage is incorrectly set

Id	3S-SG-H08
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in #8b300f0.

Description

The slippage is computed as:

```
mintAmount = _args.isLegacy & 12 == 12 ?
    IPools(_args.pool).calc_token_amount(curveInputAmounts)
    : IPools(_args.pool).calc_token_amount(curveInputAmounts, true);
mintAmount = (mintAmount / 100) * 95;
```

This gets the **`mintAmount`** value post price manipulation, rendering the slippage protection useless. Also, hardcoding a parameter of **`95%`** slippage is not ideal.

Here is a similar finding.

Recommendation

There are 2 solutions to this problem:

1. Send the minimum mint amount as an argument.

2. Use an offchain oracle such as Chainlink.

3S-SG-H09

Anyone can frontrun a relayer interaction with the same arguments but a much higher/lower relayer fee

Id	3S-SG-H09
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed in #99f0220, #02e5da2.

Description

The relayer fee used throughout the codebase is not enforced anywhere, such that it's possible to frontrun a legit transaction and set a much lower/higher relayer fee, harming the relayer/user.

Recommendation

There are several ways to tackle this issue:

1. Assert that the relayer is the **`msg.sender`**.

2. Place a cap on the relayer fee.

3. The relayer gas fee could be part of the message that the user signs.

3S-SG-H10

`DarkpoolAssetManager::Split()` into 2 equal amounts leads to lost funds

Id	3S-SG-H10
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed in #8b300f0.

Description

When using DarkpoolAssetManager::Split(), there is no check to make sure that the 2 resulting notes are not equal, making it impossible to use the second note after the first one as the nullifier will be used by then.

Check the poc here.

Recommendation

In `DarkpoolAssetManager::Split()` add a `_noteOut1 != _noteOut2` check.

3S-SG-M01

Service fees should depend on asset

Id	3S-SG-M01
Classification	Medium
Severity	Medium
Likelihood	High
Category	Suggestion
Status	Acknowledged

Description

Service fees are fixed, but they should depend on the asset, as their price differs significantly. Users are likely to arbitrage this.

Recommendation

Set a mapping for service fees.

3S-SG-M02

`CurveAddLiquidityAssetManager::curveAddLiquidity()` does not deal correctly with `isLegacy = 0b10` and `ETH`

Id	3S-SG-M02
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #8b300f0.

Description

If `isETH`, but `args.isLegacy` is 1, it wraps to `weth` and allows pool, see here.

if `args.islegacy` is `0b10` and the asset is `ETH`, it does not send `ETH` to the pool, here.

Additionally, it seems that it expects the ether balance to increase, which is unexpected, `mintAmount = address(this).balance - initAmount;`, as can be seen here.

Recommendation

It seems that the `mintAmount` would always be `IERC20(_args.lpToken).balanceOf(address(this)) - initAmount`, where `initAmount = IERC20(_args.lpToken).balanceOf(address(this));`, but a pool example should be given to confirm.

3S-SG-M03

Uniswap asset managers are missing slippage checks

Id	3S-SG-M03
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed in #8b300f0.

Description

All Uniswap interactions are missing a deadline argument and are not setting the following slippage protection arguments in **`UniswapLiquidityAssetManager`**:

- **`amount0Min`** and **`amount1Min`** in **`MintParams`**.

- **`amount0Min`** and **`amount1Min`** in **`DecreaseLiquidityParams`**.

`UniswapSwapAssetManager` sets the **`minAmountOut`** value, but is not signed by the user in the circuit, so it could be gamed.

Recommendation

These arguments should be part of the proof and the user should sign them to prevent slippage.

3S-SG-M04

No support for fee on transfer tokens

Id	3S-SG-M04
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

Fee on transfer tokens are not correctly dealt, as the **`transferFrom()`** call is expected to transfer exactly the requested amount. However, this is not the case for tokens that charge a fee on transfer.

Recommendation

To support these tokens, check the balance before calling **`safeTransferFrom()`** and compare with the new balance, getting the actual transferred amount.

3S-SG-M05

Some **ETH** transfers don't revert if they fail

Id	3S-SG-M05
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #8b300f0.

Description

ETH transfers should revert if they fail. Although this issue alone will not lead to exploits, it increases the attack surface.

Recommendation

Revert if the transfers fails in:

- Curve remove liquidity.

- Curve multi exchange.

- Curve single exchange.

3S-SG-L01

Decimals in **`ZKToken`** are not set to the underlying decimals, which will likely harm **`tvl`** calculations in aggregators

Id	3S-SG-L01
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in #37bc421, #dc4f4a4.

Description

`ZKToken`s are minted 1:1 to the underlying assets, so should inherit the same decimals. For example, if **`USDT`** is the underlying asset, and 100 **`USDT`** are locked, it will mint **`100e6`** **`ZKToken`**s. As the decimals are **`1e18`** in the OpenZeppelin implementation, this will equal much less TVL.

Recommendation

Change the constructor to set the decimals according to the underlying asset. Keep in mind that decimals are not part of the standard, so it may be required to manually set the decimals to the right value.

3S-SG-L02

`StakingOperator::setCollateralToken()` will cause issues if it changes tokens relations that have already been set

Id	3S-SG-L02
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #6e589cb, #dc4f4a4.

Description

Tokens are locked and unlocked, minting and burning **`zkToken`**s in the process, respectively. Thus, if **`zkToken`**s have been minted, but then the mappings between original and collateral tokens are changed, it will burn other **`zkToken`**s, leading to issues.

Recommendation

Either place an explicit check in the function to ensure the mappings have not been already created or keep this in mind.

3S-SG-L03

Curve pools should be whitelisted as some of them may not be **100%** compatible

Id	3S-SG-L03
Classification	Low
Severity	Medium
Likelihood	Low
Category	Suggestion
Status	Acknowledged

Description

Curve pools vary significantly between one another, which could lead to unexpected behavior. For example, it may be possible to add liquidity to a certain curve pool, but not remove it, due to incompatibilities, leading to lost funds.

Recommendation

Whitelist curve pools that have been tested to work.

3S-SG-L04

Missing event in `VerifierHub::setVerifier()`

Id	3S-SG-L04
Classification	Low
Severity	Low
Likelihood	High
Category	Suggestion
Status	Addressed in #8b300f0.

Description

VerifierHub::setVerifier() should emit an event when a **`verifier`** is set.

Recommendation

Emit events for all relevant state changes.

3S-SG-L05

Setting note commitments, nullifiers and note footers used should revert if they are already set to prevent exploits

Id	3S-SG-L05
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #8b300f0.

Description

Setting note commitments, nullifiers and note footers should never happen after they are already set, in which case the transaction should revert, as it is some clear attempt of an exploit.

Recommendation

Revert if they are already set.

3S-SG-L06

ERC20AssetPool and **ERC721AssetPool** should have the **nonReentrant** modifier as **ERC721** and some tokens have callbacks

Id	3S-SG-L06
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #8b300f0.

Description

ERC721 implements a callback when transferring tokens.

Some tokens, such as **ERC777** implement callbacks when transferring.

Both these tokens could lead to reentrancy.

Recommendation

Implement the **nonReentrant** modifier for **ERC20AssetPool** and **ERC721AssetPool**.

3S-SG-L07

Generating numbers smaller than **P** by doing **% P** might be vulnerable

Id	3S-SG-L07
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #8b300f0.

Description

For example, in BaseInputBuilder::_bytifyToNoir()

```
_bytifyToNoir(address value) internal view returns (bytes32) {
    return bytes32(uint256(keccak256(abi.encode(value))) % _primeField);
}
```

1) the hashes are not unique, num A and num B = A + P will yield the same **%**

2) doing **% P** introduces a bias in the hash where smaller values are more frequent. This makes it easier to brute force solutions

Thus, it may be possible (not sure about the required power) to specify a different address that leads to the same number supplief in the proof

See this writeup, Bug: Generating prime field elements.

Possible attack scenario: withdraw with a different address, leading to lost funds.

Recommendation

Use the **Mimc254** contract to generate safer hashes.

3S-SG-L08

Uniswap collect fees should skip collecting fees of one of the tokens if the amount is 0

Id	3S-SG-L08
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Addressed in #8b300f0.

Description

Uniswap positions may collect fees in only of the tokens, leading to 0 fees in the other. This will cause the fee manager to revert in calculateFee().

Recomendation

Skip transferring the funds if the amount is **`0`**.

3S-SG-L09

`MerkleTreeOperator::getMerklePath()` will revert due to OOG after enough elements

Id	3S-SG-L09
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

`MerkleTreeOperator::getMerklePath()` searches the tree in linear time for the index of the requested **`_noteCommitment`**. Thus, it will run out of gas or timeout when using rpc providers when enough leaves are added.

Recommendation

Store the index of each leaf in a mapping.

3S-SG-N01

`stakingAssetManager` in `ZKToken` may be immutable as it is never changed

Id	3S-SG-N01
Classification	None
Category	Optimization
Status	Addressed in #bff8628.

Description

`stakingAssetManager` in `ZKToken` can be made immutable as it is set in the constructor and never changed.

Recommendation

Set `stakingAssetManager` to immutable to save gas on reads.

3S-SG-N02

`StakingOperator::_setUnlockWindow()` checks if the times are negative, which is impossible

Id	3S-SG-N02
Classification	None
Category	Optimization
Status	Addressed in #2e77492.

Description

In **`StakingOperator::_setUnlockWindow()`**, **`unlockWindowStart`** and **`unlockWindowDuration`** are **`uint256`**, so they can not be negative. However, the code checks for negative values, which is a waste of gas.

Recommendation

Replace both checks by equalities to **`0`**.

3S-SG-N03

`StakingOperator` does not set **`isUnlockWindowActive`** to true in the constructor if the flag passed is true

Id	3S-SG-N03
Classification	None
Category	Suggestion
Status	Addressed in #5ad81c5.

Description

The constructor of **`StakingOperator`** is

```
constructor(
    uint256 unlockWindowStart,
    uint256 unlockWindowDuration,
    bool isUnlockWindowActive_,
    address owner
) Ownable(owner) {
    if (isUnlockWindowActive_) {
        _setUnlockWindow(unlockWindowStart, unlockWindowDuration);
    }
}
```

As can be seen, it receives a flag **`isUnlockWindowActive_`**, but never states the state variable **`isUnlockWindowActive`**. This may be intended, but flagging it because it is not clear.

Recommendation

Either place a comment or fix the issue.

3S-SG-N04

Checks effects interactions pattern is not always followed

Id	3S-SG-N04
Classification	None
Category	Suggestion
Status	Addressed in #89d8e20, #189f399.

Description

In **`StakingAssetManager::unlock()`**, fees are forwarded before marking the nullifier as used by calling **`_postWithdraw()`**. Although this is not exploitable because the call is permissioned by the relayer and it's not possible to register the same nullifier more than once, the pattern should always be followed to prevent future issues (if the code changes).

Recommendation

Forward the fees after recording all state changes.

3S-SG-N05

Variables are initialized to **`0`** by default

Id	3S-SG-N05
Classification	None
Category	Good Code Practices
Status	Addressed in #8b300f0.

Description

Some variables such as these are initialized to **`0`**, which is not necessary.

Recommendation

Consider not initializing the variables as it is not required.

3S-SG-N06

`UniswapLiquidityAssetManager::uniswapLiquidityProvision()` could return `tokenId`

Id	3S-SG-N06
Classification	None
Category	Good Code Practices
Status	Addressed in #8b300f0.

Description

UniswapLiquidityAssetManager::uniswapLiquidityProvision() could return the **`tokenId`** for better verbosity.

Recommendation

Return the **`tokenId`**.

3S-SG-N07

Spelling errors throughout the codebase

Id	3S-SG-N07
Classification	None
Category	Good Code Practices
Status	Addressed in #8b300f0.

Description

Some spelling errors can be found in the codebase:

https://github.com/portalgateme/darkpool-v1-zk-contracts-fork/blob/master/contracts/core/base/BaseAssetPool.sol#L17

https://github.com/portalgateme/darkpool-v1-zk-contracts-fork/blob/master/contracts/defi/curve/CurveAssetManagerHelper.sol#L179

https://github.com/portalgateme/darkpool-v1-zk-contracts-fork/blob/master/contracts/defi/curve/CurveAssetManagerHelper.sol#L147

Recommendation

Fix the spelling mistakes.

3S-SG-N08

Unused **`noteCommitment`** parameter in **`UniswapRemoveLiquidityInputs struct`**

Id	3S-SG-N08
Classification	None
Category	Bug
Status	Addressed in #8b300f0.

Description

`UniswapRemoveLiquidityInputs struct` in **`UniswapInputBuilder`** has a parameter named **`positionNoteCommitment`** which is not required.

Recommendation

Remove the mentioned parameter.

3S-SG-N09

Missing proof identifier, which could lead to using the same proof in another method

Id	3S-SG-N09
Classification	None
Category	Suggestion
Status	Addressed in #8b300f0.

Description

There is no identifier in the circuits corresponding to the method that the proof relates to. This means that if 2 circuits have exactly the same inputs, the same proof could be used for more than 1 method. Thus, if a user intended to do some action, when the proof became exposed in the mempool, it could be used to perform another action.

There are no known situations where the same proof can be used, but it is still a possibility.

Recommendation

Each circuit should have an identifier such that the proof can only match to prevent using it in another method.

3S-SG-N10

`UniswapLiquidityAssetManager` registers the note footer twice

Id	3S-SG-N10
Classification	None
Category	Optimization
Status	Addressed in #8b300f0.

Description

The note footers are registered twice.

https://github.com/portalgateme/darkpool-v1-zk-contracts-fork/blob/master/contracts/defi/uniswap/UniswapLiquidityAssetManager.sol#L574-L575

https://github.com/portalgateme/darkpool-v1-zk-contracts-fork/blob/master/contracts/defi/uniswap/UniswapLiquidityAssetManager.sol#L532-L533

Recommendation

Only register once.

3S-SG-N11

`BaseAssetManager` missing **`0`** address checks in the constructor

Id	3S-SG-N11
Classification	None
Category	Suggestion
Status	Addressed in #8b300f0.

Description

`0` address checks in the constructor are a safety check to ensure addresses are correctly set. The **`BaseAssetManager`**, which is inherited by all pool managers, could perform these checks.

Recommendation

Place these checks in the **`BaseAssetManager`**.