Cyberscope

Audit Report

Ginseng Swap

November 2024

Network   Conflux

Source   Deployed Contracts

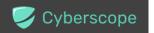Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	Address
GinsengSwapV3Factory	0x393857CFD2A007fF1311F4EBB79Ba6b498f172C7
GinsengSwapV3Pool	0xef93521ddaddfeb332a730379fdb5e19157cfa3e
NFTDescriptor	0x8b9AB5d4876b570fc33656885F536Ce654FF693B
TokenDescriptor	0x0A85E844A819e310651574a17Dfb2880f6E64579
SwapRouter	0x751326d30461E0449d109473c07Fe227a4f7fd9A
Quoter	0xf400834B70f01787f917E76160D5c8882aD5C86F
QuoterV2	0xA2cae42A0c80358Fa4995853001641B1959cedD8
NonFungiblePositionManager	0xc9F81a1A9804F41A5B5b4b20EB8b03Af7f118227
GinsengSwapV3Staker	0xd99E4a3d73381612C6CCa21aB18DaCF23b7FA8BD
TickLens	0x3C531130A0Ed24bda26EC35382730Bb356834D63

Audit Updates

Initial Audit	05 Nov 2024

Source Files

Filename	SHA256
GinsengSwapV3Factory.sol	87ca840ac1655f6046ec9056f34f931ffd81ce890829c4d97bd88257844a0bb3

GinsengSwapV3Pool.sol	03df47be2f1805bcbeb0a3f36da3db05b58b0e12592f456c827270280a5c8d4e
NFTDescriptor.sol	f4e31bf823560fe7356ee454da88b8bda6d7531183d72bf0b7127809e3ee5c23
NonfungibleTokenPositionDescriptor.sol	71932fc4257a091639d0c91b09e44e79e01a268d56aa084055435968d364a4e4
SwapRouter.sol	4a861c528c995b357a7f5f9291f1a23374320467df7d9c7c665e06328e2384d4
Quoter.sol	7c403516b2751421c24e4a4bbe9bed5c93e4a55babbbe683e9822bfef84257cf
QuoterV2.sol	4a6cc77c314ec69ba2a6b114b5696f046d66e7c86002b3cc199c1a6f6b34ae7e
NonfungiblePositionManager.sol	1541eedeedf777ed0dd07c619ca659ed541bdaab86b161413d7af1ee598ac6a7
GinsengSwapV3Staker.sol	52d8abc9ac6469f9839a773df6ceae2421d0a3c6457892268d697e587d6ecd03
TickLens.sol	07180acb53396d27c3dc6d2aa122413b1bb03ed8211615d701ecdb4963554df9

Overview

The GinsengSwap contracts implement a decentralized, permissionless trading and liquidity platform designed to support efficient token swaps and customizable liquidity provision. These contracts enable users to provide liquidity, execute token swaps, and stake positions, leveraging an advanced concentrated liquidity mechanism. With features such as customizable fee tiers, efficient multi-hop routing, and on-chain liquidity position management through NFTs, GinsengSwap aims to offer a secure and flexible decentralized exchange environment.

1. **GinsengSwapV3Factory.sol**

   The GinsengSwapV3Factory contract is the cornerstone of GinsengSwap, responsible for deploying and managing all liquidity pools within the ecosystem. As the central registry, it allows users and other contracts to locate pool addresses based on token pairs, handling initialization, fee structure management, and pool organization for GinsengSwap.

2. **GinsengSwapV3Pool.sol**

   The GinsengSwapV3Pool contract represents the core liquidity pool. It manages liquidity provision and facilitates swaps between token pairs. This contract includes advanced logic for handling concentrated liquidity, maintaining tick and fee calculations to enable users to add liquidity in specific ranges and track pricing accurately.

3. **NFTDescriptor.sol**

   NFTDescriptor is designed to generate descriptive metadata for liquidity positions represented as NFTs, working closely with the NonfungiblePositionManager to provide detailed, human-readable information about each position.

4. **NonfungibleTokenPositionDescriptor.sol**

   The NonfungibleTokenPositionDescriptor contract extends the NFTDescriptor functionality by rendering more granular data for each NFT, including fee tiers, token pairs, and other liquidity details, ensuring a clear view of each position for users.

5. **SwapRouter.sol**

   The SwapRouter contract enables token swaps within the GinsengSwap ecosystem. It routes transactions through the appropriate pools, managing both single and multi-hop swaps and enforcing minimum output constraints to ensure efficient and reliable execution for users.

6. **Quoter.sol**

   Quoter is a utility contract that allows users to estimate the output of swap
   transactions without executing them. It's a helpful tool for user interfaces and
   transaction planning, providing insights into swap outcomes for specified routes.

7. **QuoterV2.sol**

   QuoterV2 builds on the original Quoter functionality, using enhanced calculation
   methods for even greater quote accuracy, making it a valuable resource for accurate
   swap estimations.

8. **NonfungiblePositionManager.sol**

   The NonfungiblePositionManager contract handles all user interactions with liquidity
   positions represented as NFTs. Users can mint, modify, and burn their NFTs,
   representing their share of the liquidity within the pools. This contract acts as the
   interface for users to add liquidity, adjust positions, and collect fees, providing a
   streamlined experience for managing GinsengSwap liquidity positions.

9. **TickLens.sol**

   TickLens serves as a utility for accessing tick data across multiple pools efficiently. It
   allows for multi-tick retrieval in a single call, supporting users and interfaces that
   need to visualize or analyze pool states and improve data querying efficiency within
   the GinsengSwap protocol.

10. **GinsengSwapV3Staker.sol**

    The GinsengSwapV3Staker contract is the staking contract, allowing users to stake
    their liquidity positions in exchange for additional rewards. This contract incentivizes
    user participation and liquidity provision on GinsengSwap, enhancing the
    ecosystem's liquidity depth by offering rewards on top of pool earnings.

Findings Breakdown

🔴 Critical	0
🟡 Medium	0
⚪ Minor / Informative	14

Severity	Unresolved	Acknowledged	Resolved	Other
🔴 Critical	0	0	0	0
🟡 Medium	0	0	0	0
⚪ Minor / Informative	14	0	0	0

Diagnostics

🔴 Critical   🟠 Medium   ⚪ Minor / Informative

Severity	Code	Description	Status
⚪	LFB	Liquidity Fee Bypass	Unresolved
⚪	MEM	Missing Error Messages	Unresolved
⚪	MPWM	Missing Pool Whitelist Mechanism	Unresolved
⚪	MSP	Missing Slippage Protection	Unresolved
⚪	PFRI	Potential Front Running Initialization	Unresolved
⚪	PPM	Potential Price Manipulation	Unresolved
⚪	UTO	Unverified Token Order	Unresolved
⚪	L04	Conformance to Solidity Naming Conventions	Unresolved
⚪	L07	Missing Events Arithmetic	Unresolved
⚪	L13	Divide before Multiply Operation	Unresolved
⚪	L14	Uninitialized Variables in Local Scope	Unresolved
⚪	L16	Validate Variable Setters	Unresolved
⚪	L17	Usage of Solidity Assembly	Unresolved
⚪	L18	Multiple Pragma Directives	Unresolved

LFB - Liquidity Fee Bypass

Criticality	Minor / Informative
Location	NonfungiblePositionManager.sol#L202
Status	Unresolved

Description

The contract allows users to avoid paying the Liquidity Provider (LP) fee by locking a minimal amount of tokens initially and then calling the `increaseLiquidity` function to add more liquidity without incurring the fee. Currently, the LP fee is intended to collect a portion of the tokens designated to be locked when the `lock` function is called. However, this fee can be sidestepped by first locking a minimal amount of tokens and then subsequently increasing liquidity using the `increaseLiquidity` function, which does not apply the LP fee. Additionally, users can bypass the fee entirely by directly calling `increaseLiquidity` on the `NonFungiblePositionManager` contract, which also lacks the fee collection mechanism. This issue effectively allows users to bypass the fee structure, reducing revenue that would otherwise be collected from LP operations and potentially impacting the intended fee distribution model.

```
function increaseLiquidity(IncreaseLiquidityParams calldata params)
        external
        payable
        override
        checkDeadline(params.deadline)
        returns (
            uint128 liquidity,
            uint256 amount0,
            uint256 amount1
        )
    {
        Position storage position = _positions[params.tokenId];

        PoolAddress.PoolKey memory poolKey =
_poolIdToPoolKey[position.poolId];

        IGinsengSwapV3Pool pool;
        (liquidity, amount0, amount1, pool) = addLiquidity(
            AddLiquidityParams({
                token0: poolKey.token0,
                token1: poolKey.token1,
                fee: poolKey.fee,
                tickLower: position.tickLower,
                tickUpper: position.tickUpper,
                amount0Desired: params.amount0Desired,
                amount1Desired: params.amount1Desired,
                amount0Min: params.amount0Min,
                amount1Min: params.amount1Min,
                recipient: address(this)
            })
        );

        ...
        emit IncreaseLiquidity(params.tokenId, liquidity, amount0,
amount1);
    }
```

Recommendation

It is recommended to reassess the logic governing LP fee assessment and collection. Implementing fee collection upon any liquidity-increasing action, including calls to `increaseLiquidity` , would help ensure the fee is consistently applied, regardless of how users choose to interact with the LP. Alternatively, the fee logic can be integrated directly within the increase functionality to capture fees during liquidity augmentation, preventing users from circumventing it through GinsengV3's native position manager.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	SwapRouter.sol#L67,204 QuoterV2.sol#L52,74 Quoter.sol#L47,63 NonfungiblePositionManager.sol#L194,269,273,320 GinsengSwapV3Pool.sol#L113,143,153,188,197,464,838 GinsengSwapV3Factory.sol#L40,42,44,45,55,62,63,67,68
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(amount0Delta > 0 || amount1Delta > 0)
require(amountOutReceived == amountOut)
require(amountReceived == amountOutCached)
require(_exists(tokenId))
require(params.liquidity > 0)
require(positionLiquidity >= params.liquidity)
require(params.amount0Max > 0 || params.amount1Max > 0)
require(msg.sender == IGinsengSwapV3Factory(factory).owner())
require(success && data.length >= 32)
require(initializedLower)
require(initializedUpper)
require(amount > 0)


...
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MPWM - Missing Pool Whitelist Mechanism

Criticality	Minor / Informative
Location	GinsengSwapV3Factory.sol#L35
Status	Unresolved

Description

The contract currently allows the creation and use of any liquidity pool for a given token pair, regardless of the pool's Total Value Locked (TVL). Since there can be multiple pools for the same token pair, each with a different fee tier, but smaller, low-liquidity pools are more susceptible to price manipulation. As such, relying on data from these low-TVL pools could lead to inaccurate pricing and potentially manipulated transactions, exposing users to financial risks. Without a mechanism to filter out pools with insufficient liquidity, the contract cannot guarantee the reliability of data drawn from these pools, potentially undermining the integrity of pricing.

```solidity
    function createPool(
        address tokenA,
        address tokenB,
        uint24 fee
    ) external override noDelegateCall returns (address pool) {
        require(tokenA != tokenB);
        (address token0, address token1) = tokenA < tokenB ? (tokenA,
tokenB) : (tokenB, tokenA);
        require(token0 != address(0));
        int24 tickSpacing = feeAmountTickSpacing[fee];
        require(tickSpacing != 0);
        require(getPool[token0][token1][fee] == address(0));
        pool = deploy(address(this), token0, token1, fee, tickSpacing);
        getPool[token0][token1][fee] = pool;
        // populate mapping in the reverse direction, deliberate choice
to avoid the cost of comparing addresses
        getPool[token1][token0][fee] = pool;
        emit PoolCreated(token0, token1, fee, tickSpacing, pool);
    }
```

Recommendation

It is recommended to implement a whitelist mechanism that filters liquidity pools based on minimum TVL or other relevant criteria to select only pools with substantial liquidity. This approach would reduce exposure to low-TVL pools, enhancing the security and accuracy of price data used within the contract. This mechanism would allow project teams to restrict price data sources to pools with adequate liquidity, thus minimizing the risk of price manipulation and protecting users from unintended losses.

MSP - Missing Slippage Protection

Criticality	Minor / Informative
Location	GinsengSwapV3Pool.sol#L596
Status	Unresolved

Description

The contract is currently missing a mechanism for setting slippage tolerance parameters, exposing users to potential sandwich attacks and unfavorable price fluctuations during swap operations. When interacting with external protocols, the code should implement slippage protection based on the specific use case and allow users to adjust slippage tolerance to match their risk preferences. In the swap function, the `sqrtPriceLimitX96` parameter sets the bounds for the acceptable price range, which can serve as an effective safeguard against extreme price swings. Without user-configurable slippage tolerance, users may face significant losses in volatile conditions.

```solidity
function swap(
        address recipient,
        bool zeroForOne,
        int256 amountSpecified,
        uint160 sqrtPriceLimitX96,
        bytes calldata data
    ) external override noDelegateCall returns (int256 amount0, int256
amount1) {
        require(amountSpecified != 0, 'AS');

        Slot0 memory slot0Start = slot0;

        require(slot0Start.unlocked, 'LOK');
        require(
            zeroForOne
                ? sqrtPriceLimitX96 < slot0Start.sqrtPriceX96 &&
sqrtPriceLimitX96 > TickMath.MIN_SQRT_RATIO
                : sqrtPriceLimitX96 > slot0Start.sqrtPriceX96 &&
sqrtPriceLimitX96 < TickMath.MAX_SQRT_RATIO,
            'SPL'
        );
        ...
```

Recommendation

It is recommended to enable user-defined slippage tolerance by allowing them to set the `sqrtPriceLimitX96` parameter within the swap function. This addition would provide users with more control over the acceptable price range during swaps, offering greater protection against high slippage and potential sandwich attacks. Properly implementing this feature could greatly enhance user safety by preventing unexpected losses due to price volatility.

PFRI - Potential Front Running Initialization

Criticality	Minor / Informative
Location	GinsengSwapV3Pool.sol#L271
Status	Unresolved

Description

The contract is vulnerable to a front-running risk on the `GinsengSwapV3Pool.initialize` function, allowing an attacker to set an arbitrary initial price and profit from subsequent deposits. Since there are no access controls on `initialize`, any user can call this function upon pool deployment, regardless of intention. An attacker could front-run the pool initializer's transaction to set an unfavorable initial price, enabling them to exploit the liquidity provider's deposits at an unfair rate. For instance, if the intended initial price is 1:1 for two assets, the attacker could set a price of 1:10, profiting by swapping tokens at the manipulated rate after the legitimate deposit is made. This lack of restriction compromises the integrity of the initial liquidity provision and exposes early liquidity providers to substantial losses.

```
function initialize(uint160 sqrtPriceX96) external override {
        require(slot0.sqrtPriceX96 == 0, 'AI');

        int24 tick = TickMath.getTickAtSqrtRatio(sqrtPriceX96);

        (uint16 cardinality, uint16 cardinalityNext) =
observations.initialize(_blockTimestamp());

        slot0 = Slot0({
            sqrtPriceX96: sqrtPriceX96,
            tick: tick,
            observationIndex: 0,
            observationCardinality: cardinality,
            observationCardinalityNext: cardinalityNext,
            feeProtocol: 0,
            unlocked: true
        });

        emit Initialize(sqrtPriceX96, tick);
    }
```

Recommendation

It is recommended to implement measures to secure the initialization process. In the short term, consider moving the price-setting operation to the constructor, adding access controls to `initialize`, or at the very least, ensuring that documentation clearly warns about the risks of unprotected initialization. In the long term, avoid initializing the pool outside of the constructor, if possible, or implement robust safeguards and thorough documentation to mitigate the risks associated with external initialization.

PPM - Potential Price Manipulation

Criticality	Minor / Informative
Location	GinsengSwapV3Pool.sol#L236
Status	Unresolved

Description

The contract is exposed to a risk of price manipulation by using `Slot0` to directly fetch `sqrtPriceX96` for Oracle pricing. This approach makes it vulnerable to short-term price manipulation by attackers who could execute targeted swaps to adjust the liquidity pool's price to their advantage. Without protections, such as averaging, sudden price fluctuations can lead to inaccurate pricing data that affects dependent functions, especially those relying on stable, reliable prices for calculations.

```solidity
function observe(uint32[] calldata secondsAgos)
        external
        view
        override
        noDelegateCall
        returns (int56[] memory tickCumulatives, uint160[] memory
secondsPerLiquidityCumulativeX128s)
    {
        return
            observations.observe(
                _blockTimestamp(),
                secondsAgos,
                slot0.tick,
                slot0.observationIndex,
                liquidity,
                slot0.observationCardinality
            );
    }
```

Recommendation

It is recommended to implement a Time-Weighted Average Price (TWAP) mechanism for Oracle pricing to mitigate this risk. By averaging prices over time, TWAP reduces the impact of temporary price fluctuations, making manipulation more challenging. Integrating TWAP into the pricing mechanism can provide more consistent and trustworthy price data, thus enhancing the contract's resistance to manipulation.

UTO - Unverified Token Order

Criticality	Minor / Informative
Location	interfaces/IGinsengSwapV3Factory.sol#L41
Status	Unresolved

Description

The contract is currently designed to determine the base and quote tokens (`token0` and `token1`) by sorting token addresses in lexicographical order. However, this approach may introduce potential issues in cross-chain deployments where the same token might have different addresses on different blockchains. Such discrepancies could inadvertently swap the roles of `token0` and `token1` , altering the base-quote relationship. This inconsistency may impact price calculations and lead to incorrect price displays, as a token assigned to `token0` on one chain could become `token1` on another. Ensuring consistent token ordering across different environments is crucial for reliable price data and trading logic.

```solidity
function createPool(
    address tokenA,
    address tokenB,
    uint24 fee
) external override noDelegateCall returns (address pool) {
    require(tokenA != tokenB);
    (address token0, address token1) = tokenA < tokenB ? (tokenA,
tokenB) : (tokenB, tokenA);

    ...
    }
```

Recommendation

It is recommended to verify token order within the contract logic to account for cross-chain address variations. A mechanism should be implemented to standardize the identification of `token0` and `token1` across chains, ensuring that the base-quote relationship remains consistent regardless of token address variations. This verification would help maintain reliable pricing and trading functionality, minimizing logic issues stemming from cross-chain token address discrepancies.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	SwapRouter.sol#L62,65 QuoterV2.sol#L47 Quoter.sol#L42 NonfungibleTokenPositionDescriptor.sol#L28 NFTDescriptor.sol#L37 libraries/NFTDescriptor.sol#L37 GinsengSwapV3Factory.sol#L54
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function GinsengSwapV3SwapCallback(
        int256 amount0Delta,
        int256 amount1Delta,
        bytes calldata _data
    ) external override {
        require(amount0Delta > 0 || amount1Delta > 0); // swaps entirely
within 0-liquidity regions are not supported
...
            } else {
                amountInCached = amountToPay;
                tokenIn = tokenOut; // swap in/out because exact output
swaps are reversed
                pay(tokenIn, data.payer, msg.sender, amountToPay);
            }
        }
    }


...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	SwapRouter.sol#L84
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
amountInCached = amountToPay
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	NFTDescriptor.sol#L252,254 libraries/NFTDescriptor.sol#L252,254
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
tick == (TickMath.MAX_TICK / tickSpacing) * tickSpacing)
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	TickLens.sol#L23 QuoterV2.sol#L153,229 Quoter.sol#L104,150 NFTDescriptor.sol#L266,345,379,391 libraries/NFTDescriptor.sol#L266,345,379,391 GinsengSwapV3Pool.sol#L392,393,478,479
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```solidity
uint256 numberOfPopulatedTicks
bytes memory reason
bool extraDigit;
DecimalStringParams memory params;
uint8 numSigfigs;
bool flippedLower
bool flippedUpper
uint256 balance0Before
uint256 balance1Before
...
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	NonfungibleTokenPositionDescriptor.sol#L33 NonfungiblePositionManager.sol#L80 GinsengSwapV3Factory.sol#L57
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
WETH9 = _WETH9
_tokenDescriptor = _tokenDescriptor_
owner = _owner
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	QuoterV2.sol#L65,97 Quoter.sol#L56,76
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            let ptr := mload(0x40)
            mstore(ptr, amountReceived)
            mstore(add(ptr, 0x20), sqrtPriceX96After)
            mstore(add(ptr, 0x40), tickAfter)
            revert(ptr, 96)
        }

assembly {
            reason := add(reason, 0x04)
        }

...
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	TickLens.sol#L2,3 SwapRouter.sol#L2,3 QuoterV2.sol#L2,3 Quoter.sol#L2,3 NonfungibleTokenPositionDescriptor.sol#L2,3 NonfungiblePositionManager.sol#L2,3 NFTDescriptor.sol#L2,3 GinsengSwapV3Staker.sol#L2,3 libraries/NFTDescriptor.sol#L2,3 interfaces/ITickLens.sol#L2,3 interfaces/ISwapRouter.sol#L2,3 interfaces/IQuoterV2.sol#L2,3 interfaces/IQuoter.sol#L2,3 interfaces/INonfungiblePositionManager.sol#L2,3 interfaces/IGinsengSwapV3Staker.sol#L2,3
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```solidity
pragma solidity >=0.5.0;
pragma abicoder v2;
pragma solidity =0.7.6;
pragma solidity >=0.7.0;
pragma solidity >=0.7.5;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Muta bility	Modifiers
TickLens	Implementation	ITickLens		
	getPopulatedTicksInWord	Public		-
SwapRouter	Implementation	ISwapRouter, PeripheryImmutableSta te, PeripheryValidation, PeripheryPaymentsWith Fee, Multicall, SelfPermit		
		Public	✓	PeripheryImmutabl eState
	getPool	Private		
	GinsengSwapV3SwapCall back	External	✓	-
	exactInputInternal	Private	✓	
	exactInputSingle	External	Paya ble	checkDeadline
	exactInput	External	Paya ble	checkDeadline
	exactOutputInternal	Private	✓	
	exactOutputSingle	External	Paya ble	checkDeadline
	exactOutput	External	Paya ble	checkDeadline

QuoterV2	Implementation	IQuoterV2, IGinsengSwapV3Swap Callback, PeripheryImmutableSta te		
		Public	✓	PeripheryImmutabl eState
	getPool	Private		
	GinsengSwapV3SwapCall back	External		-
	parseRevertReason	Private		
	handleRevert	Private		
	quoteExactInputSingle	Public	✓	-
	quoteExactInput	Public	✓	-
	quoteExactOutputSingle	Public	✓	-
	quoteExactOutput	Public	✓	-
Quoter	Implementation	IQuoter, IGinsengSwapV3Swap Callback, PeripheryImmutableSta te		
		Public	✓	PeripheryImmutabl eState
	getPool	Private		
	GinsengSwapV3SwapCall back	External		-
	parseRevertReason	Private		
	quoteExactInputSingle	Public	✓	-
	quoteExactInput	External	✓	-
	quoteExactOutputSingle	Public	✓	-
	quoteExactOutput	External	✓	-

NonfungibleTokenPositionDescriptor	Implementation	INonfungibleTokenPositionDescriptor		
		Public	✓	-
	nativeCurrencyLabel	Public		-
	tokenURI	External		-
	flipRatio	Public		-
	tokenRatioPriority	Public		-
NonfungiblePositionManager	Implementation	INonfungiblePositionManager, Multicall, ERC721Permit, PeripheryImmutableState, PoolInitializer, LiquidityManagement, PeripheryValidation, SelfPermit		
		Public	✓	ERC721Permit PeripheryImmutableState
	positions	External		-
	cachePoolKey	Private	✓	
	mint	External	Payable	checkDeadline
	tokenURI	Public		-
	baseURI	Public		-
	increaseLiquidity	External	Payable	checkDeadline
	decreaseLiquidity	External	Payable	isAuthorizedForToken checkDeadline
	collect	External	Payable	isAuthorizedForToken

	burn	External	Paya ble	isAuthorizedForTok en
	_getAndIncrementNonce	Internal	✓	
	getApproved	Public		-
	_approve	Internal	✓	
NFTDescriptor	Library			
	constructTokenURI	Public		-
	escapeQuotes	Internal		
	generateDescriptionPartO ne	Private		
	generateDescriptionPartTw o	Private		
	generateName	Private		
	generateDecimalString	Private		
	tickToDecimalString	Internal		
	sigfigsRounded	Private		
	adjustForDecimalPrecision	Private		
	abs	Private		
	fixedPointToDecimalString	Internal		
	feeToPercentString	Internal		
	addressToString	Internal		
	generateSVGImage	Internal		
	overRange	Private		
	scale	Private		
	tokenToColorHex	Internal		

	getCircleCoord	Internal		
	sliceTokenHex	Internal		
GinsengSwapV3Staker	Implementation	IGinsengSwapV3Staker, Multicall		
	stakes	Public		-
		Public	✓	-
	createIncentive	External	✓	-
	endIncentive	External	✓	-
	onERC721Received	External	✓	-
	transferDeposit	External	✓	-
	withdrawToken	External	✓	-
	stakeToken	External	✓	-
	unstakeToken	External	✓	-
	claimReward	External	✓	-
	getRewardInfo	External		-
	_stakeToken	Private	✓	
GinsengSwapV3Pool	Implementation	IGinsengSwapV3Pool, NoDelegateCall		
		Public	✓	-
	checkTicks	Private		
	_blockTimestamp	Internal		
	balance0	Private		
	balance1	Private		

	snapshotCumulativesInside	External		noDelegateCall
	observe	External		noDelegateCall
	increaseObservationCardinalityNext	External	✓	lock noDelegateCall
	initialize	External	✓	-
	_modifyPosition	Private	✓	noDelegateCall
	_updatePosition	Private	✓	
	mint	External	✓	lock
	collect	External	✓	lock
	burn	External	✓	lock
	swap	External	✓	noDelegateCall
	flash	External	✓	lock noDelegateCall
	setFeeProtocol	External	✓	lock onlyFactoryOwner
	collectProtocol	External	✓	lock onlyFactoryOwner
GinsengV3Factory	Implementation	IGinsengSwapV3Factory, GinsengSwapV3PoolDeployer, NoDelegateCall		
		Public	✓	-
	createPool	External	✓	noDelegateCall
	setOwner	External	✓	-
	enableFeeAmount	Public	✓	-

Inheritance Graph

For the detailed Inheritance Graph image, please refer to the link provided below:

📄 Inheritance Graph_Ginseng Swap.png

Flow Graph

For the detailed Flow Graph image, please refer to the link provided below:

📄 Flow Graph_Ginseng Swap.png

Summary

The GinsengSwap contracts implement a robust decentralized exchange mechanism, supporting high-efficiency token swaps and concentrated liquidity provision. This audit investigates potential security vulnerabilities, assesses the accuracy of business logic, and explores potential improvements to enhance the stability and functionality of GinsengSwap's ecosystem.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io