Spherium Hyperswap

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 16th, 2021 - October 1st, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/27/2021	Juned Ansari
0.2	Document Updates	09/28/2021	Juned Ansari
0.3	Document Updates	09/30/2021	Juned Ansari
0.4	Final Draft	09/30/2021	Gabi Urrutia
1.0	Remediation Plan	10/14/2021	Juned Ansari

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Juned Ansari	Halborn	Juned.Ansari@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Spherium engaged Halborn to conduct a security assessment on their Hyperswap smart contracts beginning on September 16th, 2021 and ending October 1st, 2021. This security assessment was scoped to the Hyperswap smart contracts code in Solidity.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure development.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that all Nameless Contract functions are intended.
- Identify potential security issues with the assets in scope.

In summary, Halborn identified several security risks that were mostly accepted by Spherium team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover

flaws in logic, process,and implementation; automated testing techniques help enhance coverage of the Spherium contract solidity code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.

2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE : Hyperswap github repository
The security assessment was scoped to the following smart contract:

Listing 1: Hyperswap-Contract

```
core-bsc/contracts/
periphery-bsc/contracts/
governance/contracts/
rewards/contracts/
```

OUT-OF-SCOPE : External libraries and economics attacks

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	13	5

LIKELIHOOD

IMPACT

(HAL-02) (HAL-03) (HAL-04) (HAL-06)	(HAL-01)			
(HAL-11)	(HAL-05) (HAL-07) (HAL-09) (HAL-10) (HAL-12)			
(HAL-15) (HAL-16) (HAL-17)	(HAL-13) (HAL-14)	(HAL-08)		
(HAL-18) (HAL-19)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - UNCHECKED TRANSFER	Medium	SOLVED - 10/13/2021
HAL02 - MANIPULATION OF INITIAL TOKEN ADDRESSES	Low	NOT APPLICABLE
HAL03 - RE-ENTRANCY PROTECTION	Low	NOT APPLICABLE
HAL04 - MULTIPLE CALLS MAY LEADS TO DENIAL OF SERVICE(DOS)	Low	RISK ACCEPTED
HAL05 - INCOMPATIBILITY WITH INFLATIONARY TOKENS	Low	RISK ACCEPTED
HAL06 - WEAK PSEUDO-RANDOM NUMBER GENERATOR	Low	RISK ACCEPTED
HAL07 - EXTERNAL FUNCTION CALLS WITHIN LOOP	Low	RISK ACCEPTED
HAL08 - IGNORE RETURN VALUES	Low	RISK ACCEPTED
HAL09 - MISSING ZERO-ADDRESS CHECK	Low	RISK ACCEPTED
HAL10 - USAGE OF BLOCK-TIMESTAMP	Low	RISK ACCEPTED
HAL11 - FLOATING PRAGMA	Low	RISK ACCEPTED
HAL12 - OUTDATED DEPENDENCIES	Low	RISK ACCEPTED
HAL13 - PRAGMA VERSION DEPRECATED	Low	RISK ACCEPTED
HAL14 - MULTIPLE PRAGMA DEFINITION	Low	RISK ACCEPTED
HAL15 - USAGE OF STRICT-EQUALITIES	Low	RISK ACCEPTED
HAL16 - USE OF INLINE ASSEMBLY	Informational	ACKNOWLEDGED
HAL17 - PRAGMA TOO RECENT	Informational	ACKNOWLEDGED
HAL18 - REDUNDANT BOOLEAN COMPARISON	Informational	ACKNOWLEDGED
HAL19 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) UNCHECKED TRANSFER - MEDIUM

Description:

In contract SpheriumRouter01.sol and SpheriumRouter02.sol the return values of an external transfer call ISpheriumPair(pair).transferFrom(msg.sender, pair, liquidity) is not checked. It should be noted that token do not revert in case of failure and return false. If one of these tokens is used, a deposit would not revert if the transfer fails, and an attacker could deposit tokens for free.

Code Location:

Listing 2: periphery-bsc/contracts/SpheriumRouter01.sol (Lines 108)

```
 98     function removeLiquidity(
 99         address tokenA,
100         address tokenB,
101         uint liquidity,
102         uint amountAMin,
103         uint amountBMin,
104         address to,
105         uint deadline
106     ) public override ensure(deadline) returns (uint amountA, uint
            amountB) {
107         address pair = SpheriumLibrary.pairFor(factory, tokenA,
                tokenB);
108         ISpheriumPair(pair).transferFrom(msg.sender, pair,
                liquidity); // send liquidity to pair
109         (uint amount0, uint amount1) = ISpheriumPair(pair).burn(to
                );
110         (address token0,) = SpheriumLibrary.sortTokens(tokenA,
                tokenB);
```

Listing 3: periphery-bsc/contracts/SpheriumRouter02.sol (Lines 126)

```
116     function removeLiquidity(
117         address tokenA,
```

```
118         address tokenB,
119         uint liquidity,
120         uint amountAMin,
121         uint amountBMin,
122         address to,
123         uint deadline
124     ) public virtual override ensure(deadline) returns (uint
            amountA, uint amountB) {
125         address pair = SpheriumLibrary.pairFor(factory, tokenA,
                tokenB);
126         ISpheriumPair(pair).transferFrom(msg.sender, pair,
                liquidity); // send liquidity to pair
127         (uint amount0, uint amount1) = ISpheriumPair(pair).burn(to
                );
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to use SafeERC20, or ensure that the transfer return value is checked.

Remediation Plan:

SOLVED: The Spherium team solved the issue in commit 64d87fb122b5d168ee90a324c1a1e664e3019caf.

3.2 (HAL-02) MANIPULATION OF INITIAL TOKEN ADDRESSES - LOW

Description:

During a manual review, it was observed that core-bsc contract SpheriumPair.sol allows a factory user to initialize token0 and token1 at any time by making an external call. Malicious activity can be done to manipulate the functions within the contract i.e. sync, skim, swap, burn, and mint and the intended operation can be bypassed because all these functions has a dependency on token0 and token1.

Code Location:

Malicious initialization of token0 and token1 at anytime by factory.

Listing 4: core-bsc/contracts/SpheriumPair.sol (Lines 67,69,70)

```
62      constructor() public {
63          factory = msg.sender;
64      }
65
66      // called once by the factory at time of deployment
67      function initialize(address _token0, address _token1) external
            {
68          require(msg.sender == factory, 'Spherium: FORBIDDEN'); //
                sufficient check
69          token0 = _token0;
70          token1 = _token1;
71      }
```

Affected token transfer, mint, balance update, and burn

Listing 5: core-bsc/contracts/SpheriumPair.sol (Lines 113,114,139,140,149,150,151,152,171,172,174,175,194,195,200)

```
111  function mint(address to) external lock returns (uint liquidity)
         {
```

```
112         (uint112 _reserve0, uint112 _reserve1,) = getReserves();
                // gas savings
113         uint balance0 = IERC20(token0).balanceOf(address(this));
114         uint balance1 = IERC20(token1).balanceOf(address(this));
115         uint amount0 = balance0.sub(_reserve0);
116         uint amount1 = balance1.sub(_reserve1);
117
118         bool feeOn = _mintFee(_reserve0, _reserve1);
119         uint _totalSupply = totalSupply; // gas savings, must be
                defined here since totalSupply can update in _mintFee
120         if (_totalSupply == 0) {
121             liquidity = Math.sqrt(amount0.mul(amount1)).sub(
                    MINIMUM_LIQUIDITY);
122            _mint(address(0), MINIMUM_LIQUIDITY); // permanently
                    lock the first MINIMUM_LIQUIDITY tokens
123         } else {
124             liquidity = Math.min(amount0.mul(_totalSupply) /
                    _reserve0, amount1.mul(_totalSupply) / _reserve1);
125         }
126         require(liquidity > 0, 'Spherium:
                INSUFFICIENT_LIQUIDITY_MINTED');
127         _mint(to, liquidity);
128
129         _update(balance0, balance1, _reserve0, _reserve1);
130         if (feeOn) kLast = uint(reserve0).mul(reserve1); //
                reserve0 and reserve1 are up-to-date
131         emit Mint(msg.sender, amount0, amount1);
132     }
133
134     // this low-level function should be called from a contract
            which performs important safety checks
135     function burn(address to) external lock returns (uint amount0,
            uint amount1) {
136         (uint112 _reserve0, uint112 _reserve1,) = getReserves();
                // gas savings
137         address _token0 = token0;
                // gas savings
138         address _token1 = token1;
                // gas savings
139         uint balance0 = IERC20(_token0).balanceOf(address(this));
140         uint balance1 = IERC20(_token1).balanceOf(address(this));
141         uint liquidity = balanceOf[address(this)];
142
143         bool feeOn = _mintFee(_reserve0, _reserve1);
```

```
144         uint _totalSupply = totalSupply; // gas savings, must be
                defined here since totalSupply can update in _mintFee
145         amount0 = liquidity.mul(balance0) / _totalSupply; // using
                balances ensures pro-rata distribution
146         amount1 = liquidity.mul(balance1) / _totalSupply; // using
                balances ensures pro-rata distribution
147         require(amount0 > 0 && amount1 > 0, 'Spherium:
                INSUFFICIENT_LIQUIDITY_BURNED');
148         _burn(address(this), liquidity);
149         _safeTransfer(_token0, to, amount0);
150         _safeTransfer(_token1, to, amount1);
151         balance0 = IERC20(_token0).balanceOf(address(this));
152         balance1 = IERC20(_token1).balanceOf(address(this));
153
154         _update(balance0, balance1, _reserve0, _reserve1);
155         if (feeOn) kLast = uint(reserve0).mul(reserve1); //
                reserve0 and reserve1 are up-to-date
156         emit Burn(msg.sender, amount0, amount1, to);
157     }
158
159     // this low-level function should be called from a contract
            which performs important safety checks
160     function swap(uint amount0Out, uint amount1Out, address to,
            bytes calldata data) external lock {
161         require(amount0Out > 0 || amount1Out > 0, 'Spherium:
                INSUFFICIENT_OUTPUT_AMOUNT');
162         (uint112 _reserve0, uint112 _reserve1,) = getReserves();
                // gas savings
163         require(amount0Out < _reserve0 && amount1Out < _reserve1,
                'Spherium: INSUFFICIENT_LIQUIDITY');
164
165         uint balance0;
166         uint balance1;
167         { // scope for _token{0,1}, avoids stack too deep errors
168         address _token0 = token0;
169         address _token1 = token1;
170         require(to != _token0 && to != _token1, 'Spherium:
                INVALID_TO');
171         if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out)
                ; // optimistically transfer tokens
172         if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out)
                ; // optimistically transfer tokens
173         if (data.length > 0) ISpheriumCallee(to).spheriumCall(msg.
                sender, amount0Out, amount1Out, data);
```

```
174         balance0 = IERC20(_token0).balanceOf(address(this));
175         balance1 = IERC20(_token1).balanceOf(address(this));
176         }
177         uint amount0In = balance0 > _reserve0 - amount0Out ?
                balance0 - (_reserve0 - amount0Out) : 0;
178         uint amount1In = balance1 > _reserve1 - amount1Out ?
                balance1 - (_reserve1 - amount1Out) : 0;
179         require(amount0In > 0 || amount1In > 0, 'Spherium:
                INSUFFICIENT_INPUT_AMOUNT');
180         { // scope for reserve{0,1}Adjusted, avoids stack too deep
                errors
181         uint balance0Adjusted = balance0.mul(1000).sub(amount0In.
                mul(3));
182         uint balance1Adjusted = balance1.mul(1000).sub(amount1In.
                mul(3));
183         require(balance0Adjusted.mul(balance1Adjusted) >= uint(
                _reserve0).mul(_reserve1).mul(1000**2), 'Spherium: K');
184         }
185
186         _update(balance0, balance1, _reserve0, _reserve1);
187         emit Swap(msg.sender, amount0In, amount1In, amount0Out,
                amount1Out, to);
188     }
189
190     // force balances to match reserves
191     function skim(address to) external lock {
192         address _token0 = token0; // gas savings
193         address _token1 = token1; // gas savings
194         _safeTransfer(_token0, to, IERC20(_token0).balanceOf(
                address(this)).sub(reserve0));
195         _safeTransfer(_token1, to, IERC20(_token1).balanceOf(
                address(this)).sub(reserve1));
196     }
197
198     // force reserves to match balances
199     function sync() external lock {
200         _update(IERC20(token0).balanceOf(address(this)), IERC20(
                token1).balanceOf(address(this)), reserve0, reserve1);
201     }
202 }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to move the initialization of token0 and token1 to constructor() so that it can be called once at the time of deployment by the contract owner. In case to continue with the initialize function, it is recommended to declare the initialize function as internal and the function call should be done within constructor().

Remediation Plan:

NOT APPLICABLE: The Spherium team claims that factory contract can initialize the token0 and token1, but it cannot change the token0 and token1 variables for the second time.

3.3 (HAL-03) MISSING RE-ENTRANCY PROTECTION - LOW

Description:

It was identified that core-bsc Contracts are missing nonReentrant guard. In contract SpheriumPair.sol, functions burn, swap, and skim are missing nonReentrant guard. Also, in skim function, read of persistent state following external call is identified, and in burn, swap function, state variables written after the call, making it vulnerable to a Reentrancy attack.

To protect against cross-function reentrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the function with a recursive call. OpenZeppelin has it's own mutex implementation called ReentrancyGuard which provides a modifier to any function called "nonReentrant" that guards the function with a mutex against the Reentrancy attacks.

Code Location:

Listing 6: core-bsc/contracts/SpheriumPair.sol (Lines 194,195)

```
191     function skim(address to) external lock {
192         address _token0 = token0; // gas savings
193         address _token1 = token1; // gas savings
194         _safeTransfer(_token0, to, IERC20(_token0).balanceOf(
                address(this)).sub(reserve0));
195         _safeTransfer(_token1, to, IERC20(_token1).balanceOf(
                address(this)).sub(reserve1));
196     }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to change the code to follow the checks-effects-interactions pattern and use ReentrancyGuard through the nonReentrant modifier.

Remediation Plan:

NOT APPLICABLE: The Spherium team claims that due to their use of lock modifier in pair contract suffice reentrancy protection.

3.4 (HAL-04) MULTIPLE CALLS MAY LEADS TO DENIAL OF SERVICE(DOS) - LOW

Description:

In core-bsc contract SpheriumPair.sol multiple calls are executed in the same transaction. This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently and it may leads to DOS. This might be caused intentionally by a malicious call.

Code Location:

Listing 7: core-bsc/contracts/SpheriumPair.sol (Lines 194,195,200)

```
191     function skim(address to) external lock {
192         address _token0 = token0; // gas savings
193         address _token1 = token1; // gas savings
194         _safeTransfer(_token0, to, IERC20(_token0).balanceOf(
                address(this)).sub(reserve0));
195         _safeTransfer(_token1, to, IERC20(_token1).balanceOf(
                address(this)).sub(reserve1));
196     }
197
198     // force reserves to match balances
199     function sync() external lock {
200         _update(IERC20(token0).balanceOf(address(this)), IERC20(
                token1).balanceOf(address(this)), reserve0, reserve1);
201     }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

If possible, refactoring the code such that each transaction only executes one external calls or make sure that all calls can be trusted (i.e. they're part of your own codebase).

Remediation Plan:

RISK ACCEPTED: The Spherium team accepts the risk.

3.5 (HAL-05) INCOMPATIBILITY WITH INFLATIONARY TOKENS - LOW

Description:

In multiple functions periphery-bsc uses Uniswap TransferHelper safeTransferFrom and safeTransfer to handle the token transfers. These functions call transferFrom and transfer internally in the token contract to actually execute the transfer. However, since the actual amount transferred i.e. the delta of previous (before transfer) and current (after transfer) balance is not verified, a malicious user may list a custom ERC20 token with the transferFrom or transfer function modified in such a way that it e.g. does not transfer any tokens at all and the attacker is still going to have their liquidity pool tokens minted anyway. In this case both tokens are set in the constructor by the creator of the contract, so they are trusted, but it would be still a good practice to perform this check.

Code Location:

Listing 8: periphery-bsc/contracts/SpheriumRouter01.sol

```
1 TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA)
      ;
2 TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB)
      ;
3 TransferHelper.safeTransferFrom(token, msg.sender, pair,
      amountToken);
4 TransferHelper.safeTransferFrom(path[0], msg.sender,
      SpheriumLibrary.pairFor(factory, path[0], path[1]), amounts[0])
      ;
5 TransferHelper.safeTransfer(token, to, amountToken);
```

Listing 9: periphery-bsc/contracts/SpheriumRouter02.sol

```
1 TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA)
      ;
```

```
2 TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB)
      ;
3 TransferHelper.safeTransferFrom(token, msg.sender, pair,
      amountToken);
4 TransferHelper.safeTransferFrom(path[0], msg.sender,
      SpheriumLibrary.pairFor(factory, path[0], path[1]), amounts[0])
      ;
5 TransferHelper.safeTransferFrom( path[0], msg.sender,
      SpheriumLibrary.pairFor(factory, path[0], path[1]), amountIn);
6 TransferHelper.safeTransfer(token, to, amountToken);
7 TransferHelper.safeTransfer(token, to, IERC20(token).balanceOf(
      address(this)));
```

Recommendations:

Whenever tokens are transferred, the delta of the previous (before transfer) and current (after transfer) token balance should be verified to match the user-declared token amount.

Remediation Plan:

RISK ACCEPTED: The Spherium team accepts the risk.

3.6 (HAL-06) WEAK PSEUDO-RANDOM NUMBER GENERATOR - LOW

Description:

During a manual review, we noticed the use of now in core-bsc SpheriumPair.sol and use of block.timestamp in periphery-bsc SpheriumOracleLibrary.sol. Contract SpheriumPair.sol function _update and contract library SpheriumOracleLibrary.sol function currentBlockTimestamp uses a weak pseudo random number generator due to a modulo on now and blockTimestamp respectively i.e. uint32(now % 2 ** 32) and uint32(block.timestamp % 2 ** 32). The contract developers should be aware that this does not mean current time. now is an alias for block.timestamp. Miners can influence the value of block.timestamp to perform Maximal Extractable Value (MEV) attacks. The use of now creates a risk that time manipulation can be performed to manipulate price oracles. Miners can modify the timestamp by up to 900 seconds.

Code Location:

Listing 10: core-bsc/contracts/SpheriumPair.sol (Lines 76)

```
74     function _update(uint balance0, uint balance1, uint112
           _reserve0, uint112 _reserve1) private {
75         require(balance0 <= uint112(-1) && balance1 <= uint112(-1)
               , 'Spherium: OVERFLOW');
76         uint32 blockTimestamp = uint32(now % 2**32);
77         uint32 timeElapsed = blockTimestamp - blockTimestampLast;
               // overflow is desired
```

Listing 11: periphery-bsc/contracts/libraries/SpheriumOracleLibrary.sol (Lines 14)

```
13     function currentBlockTimestamp() internal view returns (uint32
           ) {
14         return uint32(block.timestamp % 2 ** 32);
15     }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

Do not use now or blockTimestamp as a source of randomness. Use block.number instead of block.timestamp or now to reduce the risk of MEV attacks.

Remediation Plan:

RISK ACCEPTED: The Spherium team accepts the risk.

3.7 (HAL-07) EXTERNAL FUNCTION CALLS WITHIN LOOP - LOW

Description:

Calls inside a loop increase Gas usage or might lead to a denial-of-service attack. In some of the functions discovered there is a for loop on variable `i` that iterates up to the `path.length` array length. If this integer is evaluated at extremely large numbers this can cause a DoS.

Code Location:

Listing 12: periphery-bsc/contracts/SpheriumRouter01.sol (Lines 175)

```
168     function _swap(uint[] memory amounts, address[] memory path,
            address _to) private {
169         for (uint i; i < path.length - 1; i++) {
170             (address input, address output) = (path[i], path[i +
                    1]);
171             (address token0,) = SpheriumLibrary.sortTokens(input,
                    output);
172             uint amountOut = amounts[i + 1];
173             (uint amount0Out, uint amount1Out) = input == token0 ?
                     (uint(0), amountOut) : (amountOut, uint(0));
174             address to = i < path.length - 2 ? SpheriumLibrary.
                    pairFor(factory, output, path[i + 2]) : _to;
175             ISpheriumPair(SpheriumLibrary.pairFor(factory, input,
                    output)).swap(amount0Out, amount1Out, to, new bytes
                    (0));
176         }
177     }
```

Listing 13: periphery-bsc/contracts/SpheriumRouter02.sol (Lines 232,233,234)

```
225     function _swap(uint[] memory amounts, address[] memory path,
            address _to) internal virtual {
226         for (uint i; i < path.length - 1; i++) {
```

```
227             (address input, address output) = (path[i], path[i +
                    1]);
228             (address token0,) = SpheriumLibrary.sortTokens(input,
                    output);
229             uint amountOut = amounts[i + 1];
230             (uint amount0Out, uint amount1Out) = input == token0 ?
                     (uint(0), amountOut) : (amountOut, uint(0));
231             address to = i < path.length - 2 ? SpheriumLibrary.
                    pairFor(factory, output, path[i + 2]) : _to;
232             ISpheriumPair(SpheriumLibrary.pairFor(factory, input,
                    output)).swap(
233                 amount0Out, amount1Out, to, new bytes(0)
234             );
235         }
```

Listing 14: periphery-bsc/contracts/SpheriumRouter02.sol (Lines 349,351,356)

```
341     function _swapSupportingFeeOnTransferTokens(address[] memory
            path, address _to) internal virtual {
342         for (uint i; i < path.length - 1; i++) {
343             (address input, address output) = (path[i], path[i +
                    1]);
344             (address token0,) = SpheriumLibrary.sortTokens(input,
                    output);
345             ISpheriumPair pair = ISpheriumPair(SpheriumLibrary.
                    pairFor(factory, input, output));
346             uint amountInput;
347             uint amountOutput;
348             { // scope to avoid stack too deep errors
349             (uint reserve0, uint reserve1,) = pair.getReserves();
350             (uint reserveInput, uint reserveOutput) = input ==
                    token0 ? (reserve0, reserve1) : (reserve1, reserve0
                    );
351             amountInput = IERC20(input).balanceOf(address(pair)).
                    sub(reserveInput);
352             amountOutput = SpheriumLibrary.getAmountOut(
                    amountInput, reserveInput, reserveOutput);
353             }
354             (uint amount0Out, uint amount1Out) = input == token0 ?
                     (uint(0), amountOutput) : (amountOutput, uint(0));
355             address to = i < path.length - 2 ? SpheriumLibrary.
                    pairFor(factory, output, path[i + 2]) : _to;
```

```
356                 pair.swap(amount0Out, amount1Out, to, new bytes(0));
357             }
358         }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

If possible, use pull over push strategy for external calls.

Reference:

External Calls Recommendation

Remediation Plan:

RISK ACCEPTED: The Spherium team accepts the risk.

3.8 (HAL-08) IGNORE RETURN VALUES - LOW

Description:

The return value of an external call is not stored in a local or state variable. In contract SpheriumRouter01.sol and SpheriumRouter02.sol, there are instances where external methods are being called and return value are being ignored.

It was observed that contract SpheriumRouter01.sol and SpheriumRouter02.sol function _addLiquidity ignores return value by ISpheriumFactory(factory).createPair(tokenA, tokenB).

Code Location:

Listing 15: periphery-bsc/contracts/SpheriumRouter01.sol (Lines 39)

```
29     function _addLiquidity(
30         address tokenA,
31         address tokenB,
32         uint amountADesired,
33         uint amountBDesired,
34         uint amountAMin,
35         uint amountBMin
36     ) private returns (uint amountA, uint amountB) {
37         // create the pair if it doesn't exist yet
38         if (ISpheriumFactory(factory).getPair(tokenA, tokenB) ==
               address(0)) {
39             ISpheriumFactory(factory).createPair(tokenA, tokenB);
40         }
```

Listing 16: periphery-bsc/contracts/SpheriumRouter02.sol (Lines 56)

```
46     function _addLiquidity(
47         address tokenA,
48         address tokenB,
49         uint amountADesired,
50         uint amountBDesired,
```

```
51          uint amountAMin,
52          uint amountBMin
53      ) internal virtual returns (uint amountA, uint amountB) {
54          // create the pair if it doesn't exist yet
55          if (ISpheriumFactory(factory).getPair(tokenA, tokenB) ==
                address(0)) {
56              ISpheriumFactory(factory).createPair(tokenA, tokenB);
57          }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Add return value check to avoid unexpected crash of the contract. Return value check will help in handling the exceptions better way.

Remediation Plan:

RISK ACCEPTED: The Spherium team accepts the risk.

3.9 (HAL-09) MISSING ZERO-ADDRESS CHECK - LOW

Description:

There are multiple instances found where Address validation is missing. Lack of zero address validation has been found when assigning user supplied address values to state variables directly.
In core-bsc contract SpheriumFactory.sol function setFeeTo(address) lacks a zero-check on _feeTo, function setFeeToSetter(address) lacks a zero-check on _feeToSetter, and constructor constructor(address) lacks a zero-check on _feeToSetter. In contract SpheriumPair.sol function initialize(address,address) lacks a zero-check on _token0 and _token1.
In periphery-bsc contracts SpheriumRouter01.sol and SpheriumRouter02.sol constructor lacks a zero-check on _factory and _WETH.
In governance contract SpheriumToken-TGE.sol function createLGEWhitelist lacks a zero-check on pairAddress.

Code Location:

Listing 17: core-bsc/contracts/SpheriumFactory.sol (Lines 19)

```
18      constructor(address _feeToSetter) public {
19          feeToSetter = _feeToSetter;
20      }
```

Listing 18: core-bsc/contracts/SpheriumFactory.sol (Lines 46)

```
44      function setFeeTo(address _feeTo) external {
45          require(msg.sender == feeToSetter, 'Spherium: FORBIDDEN');
46          feeTo = _feeTo;
47      }
```

Listing 19: core-bsc/contracts/SpheriumFactory.sol (Lines 51)

```
49      function setFeeToSetter(address _feeToSetter) external {
50          require(msg.sender == feeToSetter, 'Spherium: FORBIDDEN');
```

```
51          feeToSetter = _feeToSetter;
52      }
```

Listing 20: core-bsc/contracts/SpheriumPair.sol (Lines 69,70)

```
67      function initialize(address _token0, address _token1) external
            {
68          require(msg.sender == factory, 'Spherium: FORBIDDEN'); //
                sufficient check
69          token0 = _token0;
70          token1 = _token1;
71      }
```

Listing 21: periphery-bsc/contracts/SpheriumRouter01.sol (Lines 20,21)

```
19      constructor(address _factory, address _WETH) public {
20          factory = _factory;
21          WETH = _WETH;
22      }
```

Listing 22: periphery-bsc/contracts/SpheriumRouter02.sol (Lines 32,33)

```
31      constructor(address _factory, address _WETH) public {
32          factory = _factory;
33          WETH = _WETH;
34      }
```

Listing 23: governance/contracts/SpheriumToken-TGE.sol (Lines 389)

```
382     function createLGEWhitelist(
383         address pairAddress,
384         uint256[] calldata durations,
385         uint256[] calldata amountsMax
386     ) external onlyWhitelister() {
387         require(durations.length == amountsMax.length, "Invalid
                whitelist(s)");
388
389         _lgePairAddress = pairAddress;
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Although administrative restrictions are imposed to this function due to the OpenZeppelin RBAC it is better to add proper address validation when assigning a value to a variable from user supplied inputs.

Remediation Plan:

RISK ACCEPTED: The Spherium team accepts the risk.

3.10 (HAL-10) USAGE OF BLOCK-TIMESTAMP - LOW

Description:

During a manual review, usage of now in core-bsc SpheriumERC20.sol, SpheriumPair.sol ,and usage of block.timestamp in governance contract SpheriumToken-TGE.sol and SpheriumToken.sol were observed. The contract developers should be aware that this does not mean current time. now is an alias for block.timestamp. The value of block.timestamp can be influenced by miners to a certain degree, so the testers should be warned that this may have some risk if miners collude on time manipulation to influence the price oracles. Miners can influence the timestamp by a tolerance of 900 seconds.

Code Location:

Listing 24: core-bsc/contracts/SpheriumERC20.sol (Lines 81)

```
80     function permit(address owner, address spender, uint value,
           uint deadline, uint8 v, bytes32 r, bytes32 s) external {
81         require(deadline >= now, 'SpheriumV2: EXPIRED');
```

Listing 25: core-bsc/contracts/SpheriumPair.sol (Lines 78)

```
74     function _update(uint balance0, uint balance1, uint112
           _reserve0, uint112 _reserve1) private {
75         require(balance0 <= uint112(-1) && balance1 <= uint112(-1)
               , 'Spherium: OVERFLOW');
76         uint32 blockTimestamp = uint32(now % 2**32);
77         uint32 timeElapsed = blockTimestamp - blockTimestampLast;
               // overflow is desired
78         if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
79                         // * never overflows, and + overflow is
                               desired
80             price0CumulativeLast += uint(UQ112x112.encode(
                   _reserve1).uqdiv(_reserve0)) * timeElapsed;
```

```
81              price1CumulativeLast += uint(UQ112x112.encode(
                    _reserve0).uqdiv(_reserve1)) * timeElapsed;
82          }
```

Listing 26: governance/contracts/SpheriumToken-TGE.sol (Lines 456,463)

```
444     function getLGEWhitelistRound()
445         public
446         view
447         returns (
448             uint256,
449             uint256,
450             uint256,
451             uint256,
452             bool,
453             uint256
454         )
455     {
456         if (_lgeTimestamp > 0) {
457             uint256 wlCloseTimestampLast = _lgeTimestamp;
458
459             for (uint256 i = 0; i < _lgeWhitelistRounds.length; i
                    ++) {
460                 WhitelistRound storage wlRound =
                        _lgeWhitelistRounds[i];
461
462                 wlCloseTimestampLast = wlCloseTimestampLast +
                        wlRound.duration;
463                 if (block.timestamp <= wlCloseTimestampLast)
464                     return (
```

Listing 27: governance/contracts/SpheriumToken-TGE.sol (Lines 490,498)

```
483     function _applyLGEWhitelist(
484         address sender,
485         address recipient,
486         uint256 amount
487     ) internal {
488         if (_lgePairAddress == address(0) || _lgeWhitelistRounds.
                length == 0) return;
489
490         if (_lgeTimestamp == 0 && sender != _lgePairAddress &&
                recipient == _lgePairAddress && amount > 0)
```

```
491             _lgeTimestamp = block.timestamp;
492
493         if (sender == _lgePairAddress && recipient !=
                _lgePairAddress) {
494             //buying
495
496             (uint256 wlRoundNumber, , , , , ) =
                    getLGEWhitelistRound();
497
498             if (wlRoundNumber > 0) {
499                 WhitelistRound storage wlRound =
                        _lgeWhitelistRounds[wlRoundNumber - 1];
```

Listing 28: governance/contracts/SpheriumToken.sol (Lines 147)

```
144         address signatory = ecrecover(digest, v, r, s);
145         require(signatory != address(0), "SPHRI::delegateBySig:
                invalid signature");
146         require(nonce == nonces[signatory]++, "SPHRI::
                delegateBySig: invalid nonce");
147         require(block.timestamp <= expiry, "SPHRI::delegateBySig:
                signature expired");
148         return _delegate(signatory, delegatee);
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Use block.number instead of block.timestamp or now to reduce the risk of MEV attacks. Check if the timescale of the project occurs across years, days and months rather than seconds. If possible, it is recommended to use Oracles.

Remediation Plan:

RISK ACCEPTED: The Spherium team accepts the risk.

3.11 (HAL-11) FLOATING PRAGMA - LOW

Description:

Hyperswap governance contract SpheriumToken-TGE.sol uses the floating pragma ^0.8.0. Contract should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version too new which has not been extensively tested.

Code Location:

Listing 29: governance/contracts/SpheriumToken-TGE.sol (Lines 1)

```
1 pragma solidity ^0.8.0;
2 // SPDX-License-Identifier: MIT
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

Consider locking the pragma version with known bugs for the compiler version. When possible, do not use floating pragma in the final live deployment. Specifying a fixed compiler version ensures that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Remediation Plan:

RISK ACCEPTED: The Spherium team accepts the risk.

3.12 (HAL-12) OUTDATED DEPENDENCIES - LOW

Description:

It was noticed that the 4.1.0 version of openzepplin-contracts is used in governance smart contracts. However, the latest version of those libraries is 4.3.2, which fixes a vulnerability in UUPSUpgradeable.
It was also observed that the outdated versions 3.10 and 2.5.0 of openzepplin-contracts are used in periphery-bsc, rewards, and core-bsc smart contracts. There are no vulnerabilities in these version, but it is a security best practice to keep all libraries up-to-date.

Code Location:

Listing 30: governance/package.json (Lines 20)

```
19  "devDependencies": {
20    "@openzeppelin/contracts": "^4.1.0",
21    "@truffle/hdwallet-provider": "^1.1.0",
22    "big-number": "^2.0.0",
23    "chai": "^4.2.0",
```

Listing 31: rewards/package.json (Lines 13)

```
12  "dependencies": {
13    "@openzeppelin/contracts": "^3.1.0",
14    "@truffle/hdwallet-provider": "1.0.18",
15    "@uniswap/v2-core": "1.0.0",
16    "dotenv": "^8.2.0"
17  },
```

Listing 32: periphery-bsc/package.json (Lines 18)

```
17  "dependencies": {
18    "@openzeppelin/contracts": "^3.1.0",
19    "@truffle/hdwallet-provider": "1.0.18",
20    "@uniswap/v2-core": "1.0.0",
```

```
21      "dotenv": "^8.2.0"
22    },
```

Listing 33: periphery-bsc/package.json (Lines 52)

```
49    "dependencies": {
50      "@truffle/hdwallet-provider": "1.0.18",
51      "dotenv": "^8.2.0",
52      "@openzeppelin/contracts": "^2.5.0"
53    }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Even though UUPSUpgradeable is not used directly within governance contracts, it is always important to keep all libraries up-to-date.

References:

Open Zeppelin Advisory
UUPS Implementation Workaround

Remediation Plan:

RISK ACCEPTED: The Spherium team accepts the risk.

3.13 (HAL-13) PRAGMA VERSION DEPRECATED - LOW

Description:

In the Hyperswap periphery-bsc contracts the current pragma version in use for the contracts is pragma 0.6.0and 0.6.6. While this version is still functional, and some security issues safely implemented by mitigating contracts with other utility contracts such as SafeMath.sol, the risk to the long-term sustainability and integrity of the solidity code increases.

Code Location:

Listing 34

```
1     pragma solidity =0.6.6 (contracts/SpheriumRouter01.sol#1)
2     pragma solidity =0.6.6 (contracts/interfaces/IERC20.sol#1)
3     pragma solidity =0.6.6 (contracts/interfaces/ISpheriumFactory.
          sol#1)
4     pragma solidity =0.6.6 (contracts/interfaces/ISpheriumPair.sol
          #1)
5     pragma solidity =0.6.6 (contracts/interfaces/ISpheriumRouter01
          .sol#1)
6     pragma solidity =0.6.6 (contracts/interfaces/IWETH.sol#1)
7     pragma solidity =0.6.6 (contracts/libraries/SafeMath.sol#1)
8     pragma solidity =0.6.6 (contracts/libraries/SpheriumLibrary.
          sol#1)
9     pragma solidity >=0.6.0 (contracts/libraries/TransferHelper.
          sol#3)
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

If possible, consider using the latest stable pragma version that has been thoroughly tested to prevent potential undiscovered vulnerabilities such as pragma between 0.6.12 - 0.7.6.

Remediation Plan:

RISK ACCEPTED: The Spherium team accepts the risk.

3.14 (HAL-14) MULTIPLE PRAGMA DEFINITION - LOW

Description:

In the Hyperswap periphery-bsc contracts, different Pragma version(0.6.6 and 0.6.0) is defined. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma, for example, either an outdated pragma version that might introduce bugs that affect the contract system negatively or a recently released pragma version which has not been extensively tested.

Code Location:

Listing 35: periphery-bsc/contracts/

```
1 pragma solidity =0.6.6 (contracts/SpheriumRouter01.sol#1)
2 pragma solidity =0.6.6 (contracts/interfaces/IERC20.sol#1)
3 pragma solidity =0.6.6 (contracts/interfaces/ISpheriumFactory.sol
    #1)
4 pragma solidity =0.6.6 (contracts/interfaces/ISpheriumPair.sol#1)
5 pragma solidity =0.6.6 (contracts/interfaces/ISpheriumRouter01.sol
    #1)
6 pragma solidity =0.6.6 (contracts/interfaces/IWETH.sol#1)
7 pragma solidity =0.6.6 (contracts/libraries/SafeMath.sol#1)
8 pragma solidity =0.6.6 (contracts/libraries/SpheriumLibrary.sol#1)
9 pragma solidity >=0.6.0 (contracts/libraries/TransferHelper.sol#3)
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

Consider lock and use single pragma version known bugs for the compiler version.

Remediation Plan:

RISK ACCEPTED: The Spherium team accepts the risk.

3.15 (HAL-15) USAGE OF STRICT-EQUALITIES - INFORMATIONAL

Description:

Avoid checking for strict equality, use of strict equalities can be easily manipulated by an attacker via selfdestruct() or by mining.

Code Location:

Listing 36: core-bsc/contracts/SpheriumPair.sol (Lines 120)

```
118         bool feeOn = _mintFee(_reserve0, _reserve1);
119         uint _totalSupply = totalSupply; // gas savings, must be
                defined here since totalSupply can update in _mintFee
120         if (_totalSupply == 0) {
121             liquidity = Math.sqrt(amount0.mul(amount1)).sub(
                    MINIMUM_LIQUIDITY);
122            _mint(address(0), MINIMUM_LIQUIDITY); // permanently
                   lock the first MINIMUM_LIQUIDITY tokens
```

Listing 37: governance/contracts/SpheriumToken-TGE.sol (Lines 490)

```
483     function _applyLGEWhitelist(
484         address sender,
485         address recipient,
486         uint256 amount
487     ) internal {
488         if (_lgePairAddress == address(0) || _lgeWhitelistRounds.
                length == 0) return;
489
490         if (_lgeTimestamp == 0 && sender != _lgePairAddress &&
                recipient == _lgePairAddress && amount > 0)
491             _lgeTimestamp = block.timestamp;
```

Listing 38: governance/contracts/SpheriumToken.sol (Lines 252)

```
242     function _writeCheckpoint(
243         address delegatee,
```

```
244         uint32 nCheckpoints,
245         uint256 oldVotes,
246         uint256 newVotes
247     )
248         internal
249     {
250         uint32 blockNumber = safe32(block.number, "SPHRI::
                _writeCheckpoint: block number exceeds 32 bits");
251
252         if (nCheckpoints > 0 && checkpoints[delegatee][
                nCheckpoints - 1].fromBlock == blockNumber) {
253             checkpoints[delegatee][nCheckpoints - 1].votes =
                    newVotes;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

While these sections of code use it for totalSupply, and time validation. Don’t use strict equality to determine if an account has enough Ether or tokens.

Remediation Plan:

RISK ACCEPTED: The Spherium team accepts the risk.

3.16 (HAL-16) USE OF INLINE ASSEMBLY - INFORMATIONAL

Description:

Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This discards several important safety features in Solidity. Inline assembly is used in core-bsc SpheriumERC20.sol, SpheriumFactory.sol, and governance spheriumToken.sol contracts.

Code Location:

Listing 39: core-bsc/contracts/SpheriumERC20.sol (Lines 25,26,27)

```
25          assembly {
26              chainId := chainid
27          }
```

Listing 40: core-bsc/contracts/SpheriumFactory.sol (Lines 33,34,35)

```
33          assembly {
34              pair := create2(0, add(bytecode, 32), mload(bytecode),
                    salt)
35          }
```

Listing 41: governance/contracts/spheriumToken.sol (Lines 269)

```
267     function getChainId() internal view returns (uint) {
268         uint256 chainId;
269         assembly { chainId := chainid() }
270         return chainId;
271     }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

When possible, do not use inline assembly because it is a manner to access to the EVM (Ethereum Virtual Machine) at a low level. An attacker could bypass many important safety features of Solidity.

Remediation Plan:

ACKNOWLEDGED: The Spherium team accepts the risk.

3.17 (HAL-17) PRAGMA TOO RECENT - INFORMATIONAL

Description:

Hyperswap in-scope governance Contract uses one of the latest pragma version (0.8.0) which was released on December 16, 2020. The latest pragma version (0.8.7) was released in August 2021. Many pragma versions have been lately released, going from version 0.7.x to the recently released version 0.8.x. in just 6 months.

Reference: https://github.com/ethereum/solidity/releases

In the Solitidy Github repository, there is a json file where are all bugs finding in the different compiler versions. It should be noted that pragma 0.6.12 and 0.7.6 are widely used by Solidity developers and have been extensively tested in many security audits.

Reference: https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json

Code Location:

Listing 42: governance/contracts/SpheriumToken-TGE.sol (Lines 1)

```
1 pragma solidity ^0.8.0;
2 // SPDX-License-Identifier: MIT
```

Listing 43: governance/contracts/SpheriumToken.sol (Lines 2)

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.0;
```

Risk Level:

Likelihood - 1

Impact - 2

Recommendations:

If possible, consider using the latest stable pragma version that has been thoroughly tested to prevent potential undiscovered vulnerabilities such as pragma between 0.6.12 - 0.7.6.

Remediation Plan:

ACKNOWLEDGED: The Spherium team acknowledged the issue.

3.18 (HAL-18) REDUNDANT BOOLEAN COMPARISON - INFORMATIONAL

Description:

In the solidity language, Boolean constants can be used directly and do not need to be compare to true or false. In the periphery-bsc contracts, boolean constants are compared with false.

Code Location:

Listing 44: periphery-bsc/contracts/SpheriumRouter02.sol (Lines 237)

```
236         if(spheriumTraderRewards != address(0)){
237             if(ISpheriumTraderRewards(spheriumTraderRewards).
                    paused() == false){
238                 ISpheriumTraderRewards(spheriumTraderRewards).
                        recordTrade(amounts, path, _to);
239             }
240         }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended to compare boolean constants directly in the if statement.

Remediation Plan:

ACKNOWLEDGED: The Spherium team acknowledged the issue.

3.19 (HAL-19) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.
Also, methods do not necessarily have to be public if they are only called within the contract-in such case they should be marked internal.

Code Location:

Below are smart contracts and their corresponding functions affected:

"periphery-bsc/contracts/SpheriumRouter01.sol:
getAmountIn(uint256,uint256,uint256) getAmountOut(uint256,uint256,uint256)
getAmountsIn(uint256,address[]) getAmountsOut(uint256,address[])
quote(uint256,uint256,uint256)

"periphery-bsc/contracts/SpheriumRouter02.sol:
getAmountIn(uint256,uint256,uint256) getAmountOut(uint256,uint256,uint256)
getAmountsIn(uint256,address[]) getAmountsOut(uint256,address[])
quote(uint256,uint256,uint256) setSpheriumTraderRewards(address)

governance/contracts/SpheriumToken-TGE.sol:
renounceOwnership() transferOwnership(address)

governance/contracts/spheriumToken.sol:
mint(address,uint256)

rewards/contracts/TraderRewards.sol:
recordTrade(address) setDivisor(uint256) setRewardTokensRemaining(uint256)

setRouter(address) setSphToken(address) withdrawRewardTokens()

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider as much as possible declaring external variables instead of public variables. As for best practice, you should use external if you expect that the function will only be called externally and use public if you need to call the function internally. To sum up, all can access to public functions, external functions only can be accessed externally and internal functions can only be called within the contract.

Remediation Plan:

ACKNOWLEDGED: The Spherium team acknowledged the issue.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

```
INFO:Detectors:
SpheriumERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (contracts/SpheriumERC20.sol#80-92) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(deadline >= now,SpheriumV2: EXPIRED) (contracts/SpheriumERC20.sol#81)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
SpheriumPair._update(uint256,uint256,uint112,uint112) (contracts/SpheriumPair.sol#74-87) uses a weak PRNG: "blockTimestamp = uint32(now % 2 ** 32) (contracts/SpheriumPair.sol#76)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
SpheriumPair._safeTransfer(address,address,uint256) (contracts/SpheriumPair.sol#45-48) uses a dangerous strict equality:
        - require(bool,string)(success && (data.length == 0 || abi.decode(data,(bool))),Spherium: TRANSFER_FAILED) (contracts/SpheriumPair.sol#47)
SpheriumPair.mint(address) (contracts/SpheriumPair.sol#111-132) uses a dangerous strict equality:
        - _totalSupply == 0 (contracts/SpheriumPair.sol#120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in SpheriumPair.burn(address) (contracts/SpheriumPair.sol#135-157):
        External calls:
        - _safeTransfer(_token0,to,amount0) (contracts/SpheriumPair.sol#149)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
        - _safeTransfer(_token1,to,amount1) (contracts/SpheriumPair.sol#150)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
        State variables written after the call(s):
        - _update(balance0,balance1,_reserve0,_reserve1) (contracts/SpheriumPair.sol#154)
                - blockTimestampLast = blockTimestamp (contracts/SpheriumPair.sol#85)
        - kLast = uint256(reserve0).mul(reserve1) (contracts/SpheriumPair.sol#155)
        - _update(balance0,balance1,_reserve0,_reserve1) (contracts/SpheriumPair.sol#154)
                - reserve0 = uint112(balance0) (contracts/SpheriumPair.sol#83)
        - _update(balance0,balance1,_reserve0,_reserve1) (contracts/SpheriumPair.sol#154)
                - reserve1 = uint112(balance1) (contracts/SpheriumPair.sol#84)
Reentrancy in SpheriumFactory.createPair(address,address) (contracts/SpheriumFactory.sol#26-42):
        External calls:
        - ISpheriumPair(pair).initialize(token0,token1) (contracts/SpheriumFactory.sol#36)
        State variables written after the call(s):
        - getPair[token0][token1] = pair (contracts/SpheriumFactory.sol#37)
        - getPair[token1][token0] = pair (contracts/SpheriumFactory.sol#38)
Reentrancy in SpheriumPair.swap(uint256,uint256,address,bytes) (contracts/SpheriumPair.sol#160-188):
        External calls:
        - _safeTransfer(_token0,to,amount0Out) (contracts/SpheriumPair.sol#171)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
        - _safeTransfer(_token1,to,amount1Out) (contracts/SpheriumPair.sol#172)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
        - ISpheriumCallee(to).spheriumCall(msg.sender,amount0Out,amount1Out,data) (contracts/SpheriumPair.sol#173)
        State variables written after the call(s):
        - _update(balance0,balance1,_reserve0,_reserve1) (contracts/SpheriumPair.sol#186)
                - blockTimestampLast = blockTimestamp (contracts/SpheriumPair.sol#85)
        - _update(balance0,balance1,_reserve0,_reserve1) (contracts/SpheriumPair.sol#186)
                - reserve0 = uint112(balance0) (contracts/SpheriumPair.sol#83)
        - _update(balance0,balance1,_reserve0,_reserve1) (contracts/SpheriumPair.sol#186)
                - reserve1 = uint112(balance1) (contracts/SpheriumPair.sol#84)
SpheriumFactory.constructor(address)._feeToSetter (contracts/SpheriumFactory.sol#18) lacks a zero-check on :
                - feeToSetter = _feeToSetter (contracts/SpheriumFactory.sol#19)
SpheriumFactory.setFeeTo(address)._feeTo (contracts/SpheriumFactory.sol#44) lacks a zero-check on :
                - feeTo = _feeTo (contracts/SpheriumFactory.sol#46)
SpheriumFactory.setFeeToSetter(address)._feeToSetter (contracts/SpheriumFactory.sol#49) lacks a zero-check on :
                - feeToSetter = _feeToSetter (contracts/SpheriumFactory.sol#51)
SpheriumPair.initialize(address,address)._token0 (contracts/SpheriumPair.sol#67) lacks a zero-check on :
                - token0 = _token0 (contracts/SpheriumPair.sol#69)
SpheriumPair.initialize(address,address)._token1 (contracts/SpheriumPair.sol#67) lacks a zero-check on :
                - token1 = _token1 (contracts/SpheriumPair.sol#70)
```

AUTOMATED TESTING

```
Reentrancy in SpheriumPair.burn(address) (contracts/SpheriumPair.sol#135-157):
        External calls:
        - _safeTransfer(_token0,to,amount0) (contracts/SpheriumPair.sol#149)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
        - _safeTransfer(_token1,to,amount1) (contracts/SpheriumPair.sol#150)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
        State variables written after the call(s):
        - _update(balance0,balance1,_reserve0,_reserve1) (contracts/SpheriumPair.sol#154)
                - price0CumulativeLast += uint256(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed (contracts/SpheriumPair.sol#80)
        - _update(balance0,balance1,_reserve0,_reserve1) (contracts/SpheriumPair.sol#154)
                - price1CumulativeLast += uint256(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed (contracts/SpheriumPair.sol#81)
Reentrancy in SpheriumFactory.createPair(address,address) (contracts/SpheriumFactory.sol#26-42):
        External calls:
        - ISpheriumPair(pair).initialize(token0,token1) (contracts/SpheriumFactory.sol#36)
        State variables written after the call(s):
        - allPairs.push(pair) (contracts/SpheriumFactory.sol#39)
Reentrancy in SpheriumPair.swap(uint256,uint256,address,bytes) (contracts/SpheriumPair.sol#160-188):
        External calls:
        - _safeTransfer(_token0,to,amount0Out) (contracts/SpheriumPair.sol#171)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
        - _safeTransfer(_token1,to,amount1Out) (contracts/SpheriumPair.sol#172)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
        - ISpheriumCallee(to).spheriumCall(msg.sender,amount0Out,amount1Out,data) (contracts/SpheriumPair.sol#173)
        State variables written after the call(s):
        - _update(balance0,balance1,_reserve0,_reserve1) (contracts/SpheriumPair.sol#186)
                - price0CumulativeLast += uint256(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed (contracts/SpheriumPair.sol#80)
        - _update(balance0,balance1,_reserve0,_reserve1) (contracts/SpheriumPair.sol#186)
                - price1CumulativeLast += uint256(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed (contracts/SpheriumPair.sol#81)
Reentrancy in SpheriumPair.burn(address) (contracts/SpheriumPair.sol#135-157):
        External calls:
        - _safeTransfer(_token0,to,amount0) (contracts/SpheriumPair.sol#149)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
        - _safeTransfer(_token1,to,amount1) (contracts/SpheriumPair.sol#150)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
        Event emitted after the call(s):
        - Burn(msg.sender,amount0,amount1,to) (contracts/SpheriumPair.sol#156)
        - Sync(reserve0,reserve1) (contracts/SpheriumPair.sol#86)
                - _update(balance0,balance1,_reserve0,_reserve1) (contracts/SpheriumPair.sol#154)
Reentrancy in SpheriumFactory.createPair(address,address) (contracts/SpheriumFactory.sol#26-42):
        External calls:
        - ISpheriumPair(pair).initialize(token0,token1) (contracts/SpheriumFactory.sol#36)
        Event emitted after the call(s):
        - PairCreated(token0,token1,pair,allPairs.length) (contracts/SpheriumFactory.sol#40)
Reentrancy in SpheriumPair.swap(uint256,uint256,address,bytes) (contracts/SpheriumPair.sol#160-188):
        External calls:
        - _safeTransfer(_token0,to,amount0Out) (contracts/SpheriumPair.sol#171)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
        - _safeTransfer(_token1,to,amount1Out) (contracts/SpheriumPair.sol#172)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
        - ISpheriumCallee(to).spheriumCall(msg.sender,amount0Out,amount1Out,data) (contracts/SpheriumPair.sol#173)
        Event emitted after the call(s):
        - Swap(msg.sender,amount0In,amount1In,amount0Out,amount1Out,to) (contracts/SpheriumPair.sol#187)
        - Sync(reserve0,reserve1) (contracts/SpheriumPair.sol#86)
                - _update(balance0,balance1,_reserve0,_reserve1) (contracts/SpheriumPair.sol#186)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
SpheriumERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (contracts/SpheriumERC20.sol#80-92) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(deadline >= now,SpheriumV2: EXPIRED) (contracts/SpheriumERC20.sol#81)
SpheriumPair._update(uint256,uint256,uint112,uint112) (contracts/SpheriumPair.sol#74-87) uses timestamp for comparisons
        Dangerous comparisons:
        - timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0 (contracts/SpheriumPair.sol#78)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
SpheriumPair._update(uint256,uint256,uint112,uint112) (contracts/SpheriumPair.sol#74-87) uses a weak PRNG: "blockTimestamp = uint32(now % 2 ** 32) (contracts/SpheriumPair.sol#76)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
SpheriumERC20.constructor() (contracts/SpheriumERC20.sol#23-37) uses assembly
        - INLINE ASM (contracts/SpheriumERC20.sol#25-27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
SafeMath.mul(uint256,uint256) (contracts/libraries/SafeMath.sol#14-16) is never used and should be removed
SpheriumERC20._burn(address,uint256) (contracts/SpheriumERC20.sol#45-49) is never used and should be removed
SpheriumERC20._mint(address,uint256) (contracts/SpheriumERC20.sol#39-43) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
SpheriumERC20.constructor() (contracts/SpheriumERC20.sol#23-37) uses assembly
        - INLINE ASM (contracts/SpheriumERC20.sol#25-27)
SpheriumFactory.createPair(address,address) (contracts/SpheriumFactory.sol#26-42) uses assembly
        - INLINE ASM (contracts/SpheriumFactory.sol#33-35)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Low level call in SpheriumPair._safeTransfer(address,address,uint256) (contracts/SpheriumPair.sol#45-48):
        - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
SpheriumFactory.createPair(address,address) (contracts/SpheriumFactory.sol#26-42) uses literals with too many digits:
        - bytecode = type(address)(SpheriumPair).creationCode (contracts/SpheriumFactory.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
SpheriumERC20.constructor() (contracts/SpheriumERC20.sol#23-37) uses assembly
        - INLINE ASM (contracts/SpheriumERC20.sol#25-27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Low level call in SpheriumPair._safeTransfer(address,address,uint256) (contracts/SpheriumPair.sol#45-48):
        - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (contracts/SpheriumPair.sol#46)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Slither:contracts/ analyzed (35 contracts with 18 detectors), 10 result(s) found
```

AUTOMATED TESTING

```
name() should be declared external:
        - MockERC20.name() (contracts/test/mockToken.sol#64-66)
symbol() should be declared external:
        - MockERC20.symbol() (contracts/test/mockToken.sol#72-74)
decimals() should be declared external:
        - MockERC20.decimals() (contracts/test/mockToken.sol#89-91)
totalSupply() should be declared external:
        - MockERC20.totalSupply() (contracts/test/mockToken.sol#96-98)
balanceOf(address) should be declared external:
        - MockERC20.balanceOf(address) (contracts/test/mockToken.sol#103-105)
transfer(address,uint256) should be declared external:
        - MockERC20.transfer(address,uint256) (contracts/test/mockToken.sol#115-118)
allowance(address,address) should be declared external:
        - MockERC20.allowance(address,address) (contracts/test/mockToken.sol#123-125)
approve(address,uint256) should be declared external:
        - MockERC20.approve(address,uint256) (contracts/test/mockToken.sol#134-137)
transferFrom(address,address,uint256) should be declared external:
        - MockERC20.transferFrom(address,address,uint256) (contracts/test/mockToken.sol#152-156)
increaseAllowance(address,uint256) should be declared external:
        - MockERC20.increaseAllowance(address,uint256) (contracts/test/mockToken.sol#170-173)
decreaseAllowance(address,uint256) should be declared external:
        - MockERC20.decreaseAllowance(address,uint256) (contracts/test/mockToken.sol#189-192)
```

```
LGEWhitelisted._applyLGEWhitelist(address,address,uint256) (contracts/SpheriumToken-TGE.sol#483-512) uses a dangerous strict equality:
        - _lgeTimestamp == 0 && sender != _lgePairAddress && recipient == _lgePairAddress && amount > 0 (contracts/SpheriumToken-TGE.sol#490)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
BEP20TokenWhitelisted.allowance(address,address).owner (contracts/SpheriumToken-TGE.sol#596) shadows:
        - Ownable.owner() (contracts/SpheriumToken-TGE.sol#147-149) (function)
BEP20TokenWhitelisted._approve(address,address,uint256).owner (contracts/SpheriumToken-TGE.sol#704) shadows:
        - Ownable.owner() (contracts/SpheriumToken-TGE.sol#147-149) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
LGEWhitelisted.createLGEWhitelist(address,uint256[],uint256[]).pairAddress (contracts/SpheriumToken-TGE.sol#383) lacks a zero-check on :
                - _lgePairAddress = pairAddress (contracts/SpheriumToken-TGE.sol#389)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
LGEWhitelisted.getLGEWhitelistRound() (contracts/SpheriumToken-TGE.sol#444-476) uses timestamp for comparisons
        Dangerous comparisons:
        - _lgeTimestamp > 0 (contracts/SpheriumToken-TGE.sol#456)
        - block.timestamp <= wlCloseTimestampLast (contracts/SpheriumToken-TGE.sol#463)
LGEWhitelisted._applyLGEWhitelist(address,address,uint256) (contracts/SpheriumToken-TGE.sol#483-512) uses timestamp for comparisons
        Dangerous comparisons:
        - _lgeTimestamp == 0 && sender != _lgePairAddress && recipient == _lgePairAddress && amount > 0 (contracts/SpheriumToken-TGE.sol#490)
        - wlRoundNumber > 0 (contracts/SpheriumToken-TGE.sol#498)
```

```
INFO:Detectors:
Pragma version0.8.0 (contracts/SpheriumToken.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.0 (contracts/libraries/SafeMath.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
spheriumToken.constructor() (contracts/SpheriumToken.sol#47) uses literals with too many digits:
        - ERC20Capped(100000000 * (10 ** uint256(18))) (contracts/SpheriumToken.sol#47)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
mint(address,uint256) should be declared external:
        - spheriumToken.mint(address,uint256) (contracts/SpheriumToken.sol#54-57)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

```
INFO:Detectors:
SpheriumOracleLibrary.currentCumulativePrices(address) (contracts/libraries/SpheriumOracleLibrary.sol#18-36) uses timestamp for comparisons
        Dangerous comparisons:
        - blockTimestampLast != blockTimestamp (contracts/libraries/SpheriumOracleLibrary.sol#27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
INFO:Detectors:
Low level call in TransferHelper.safeApprove(address,address,uint256) (contracts/libraries/TransferHelper.sol#7-18):
        - (success,data) = token.call(abi.encodeWithSelector(0x095ea7b3,to,value)) (contracts/libraries/TransferHelper.sol#13)
Low level call in TransferHelper.safeTransfer(address,address,uint256) (contracts/libraries/TransferHelper.sol#20-31):
        - (success,data) = token.call(abi.encodeWithSelector(0xa9059cbb,to,value)) (contracts/libraries/TransferHelper.sol#26)
Low level call in TransferHelper.safeTransferFrom(address,address,address,uint256) (contracts/libraries/TransferHelper.sol#33-45):
        - (success,data) = token.call(abi.encodeWithSelector(0x23b872dd,from,to,value)) (contracts/libraries/TransferHelper.sol#40)
Low level call in TransferHelper.safeTransferETH(address,uint256) (contracts/libraries/TransferHelper.sol#47-50):
        - (success) = to.call{value: value}(new bytes(0)) (contracts/libraries/TransferHelper.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
setSpheriumTraderRewards(address) should be declared external:
        - SpheriumRouter02.setSpheriumTraderRewards(address) (contracts/SpheriumRouter02.sol#41-43)
quote(uint256,uint256,uint256) should be declared external:
        - SpheriumRouter02.quote(uint256,uint256,uint256) (contracts/SpheriumRouter02.sol#423-425)
getAmountOut(uint256,uint256,uint256) should be declared external:
        - SpheriumRouter02.getAmountOut(uint256,uint256,uint256) (contracts/SpheriumRouter02.sol#427-435)
getAmountIn(uint256,uint256,uint256) should be declared external:
        - SpheriumRouter02.getAmountIn(uint256,uint256,uint256) (contracts/SpheriumRouter02.sol#437-445)
getAmountsOut(uint256,address[]) should be declared external:
        - SpheriumRouter02.getAmountsOut(uint256,address[]) (contracts/SpheriumRouter02.sol#447-455)
getAmountsIn(uint256,address[]) should be declared external:
        - SpheriumRouter02.getAmountsIn(uint256,address[]) (contracts/SpheriumRouter02.sol#457-465)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

```
INFO:Detectors:
SpheriumRouter01.removeLiquidity(address,address,uint256,uint256,uint256,address,uint256) (contracts/SpheriumRouter01.sol#98-114) ignores return value by ISpheriumPair(pair).transferFrom(msg.sender,pair,liquidity) (contracts/SpheriumRouter01.sol#108)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
SpheriumRouter01._swap(uint256[],address[],address).i (contracts/SpheriumRouter01.sol#169) is a local variable never initialized
SpheriumLibrary.getAmountsOut(address,uint256,address[]).i (contracts/libraries/SpheriumLibrary.sol#66) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
SpheriumRouter01._addLiquidity(address,address,uint256,uint256,uint256,uint256) (contracts/SpheriumRouter01.sol#29-56) ignores return value by ISpheriumFactory(factory).createPair(tokenA,tokenB) (contracts/SpheriumRouter01.sol#39)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
SpheriumRouter01.constructor(address,address)._factory (contracts/SpheriumRouter01.sol#19) lacks a zero-check on :
        - factory = _factory (contracts/SpheriumRouter01.sol#20)
SpheriumRouter01.constructor(address,address)._WETH (contracts/SpheriumRouter01.sol#19) lacks a zero-check on :
        - WETH = _WETH (contracts/SpheriumRouter01.sol#21)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
SpheriumRouter01._swap(uint256[],address[],address) (contracts/SpheriumRouter01.sol#168-177) has external calls inside a loop: ISpheriumPair(SpheriumLibrary.pairFor(factory,input,output)).swap(amount0Out,amount1Out,to,new bytes(0)) (contracts/SpheriumRouter01.sol#175)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
SpheriumRouter02._swap(uint256[],address[],address) (contracts/SpheriumRouter02.sol#225-241) has external calls inside a loop: ISpheriumPair(SpheriumLibrary.pairFor(factory,input,output)).swap(amount0Out,amount1Out,to,new bytes(0)) (contracts/SpheriumRouter02.sol#232-234)
SpheriumRouter02._swapSupportingFeeOnTransferTokens(address[],address) (contracts/SpheriumRouter02.sol#341-358) has external calls inside a loop: (reserve0,reserve1) = pair.getReserves() (contracts/SpheriumRouter02.sol#349)
SpheriumRouter02._swapSupportingFeeOnTransferTokens(address[],address) (contracts/SpheriumRouter02.sol#341-358) has external calls inside a loop: amountInput = IERC20(input).balanceOf(address(pair)).sub(reserveInput) (contracts/SpheriumRouter02.sol#351)
SpheriumRouter02._swapSupportingFeeOnTransferTokens(address[],address) (contracts/SpheriumRouter02.sol#341-358) has external calls inside a loop: pair.swap(amount0Out,amount1Out,to,new bytes(0)) (contracts/SpheriumRouter02.sol#356)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
SpheriumLibrary.getAmountsOut(address,uint256,address[]).i (contracts/libraries/SpheriumLibrary.sol#66) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
Pragma version=0.6.6 (contracts/interfaces/ISpheriumFactory.sol#1) allows old versions
Pragma version=0.6.6 (contracts/interfaces/ISpheriumPair.sol#1) allows old versions
Pragma version=0.6.6 (contracts/libraries/Babylonian.sol#3) allows old versions
Pragma version=0.6.6 (contracts/libraries/FullMath.sol#2) allows old versions
Pragma version=0.6.6 (contracts/libraries/SafeMath.sol#1) allows old versions
Pragma version=0.6.6 (contracts/libraries/SpheriumLibrary.sol#1) allows old versions
Pragma version=0.6.6 (contracts/libraries/SpheriumLiquidityMathLibrary.sol#1) allows old versions
solc-0.6.6 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
recordTrade(address) should be declared external:
        - TraderRewards.recordTrade(address) (contracts/TraderRewards.sol#78-84)
withdrawRewardTokens() should be declared external:
        - TraderRewards.withdrawRewardTokens() (contracts/TraderRewards.sol#91-98)
setSphToken(address) should be declared external:
        - TraderRewards.setSphToken(address) (contracts/TraderRewards.sol#105-110)
setRouter(address) should be declared external:
        - TraderRewards.setRouter(address) (contracts/TraderRewards.sol#117-122)
setDivisor(uint256) should be declared external:
        - TraderRewards.setDivisor(uint256) (contracts/TraderRewards.sol#128-133)
setRewardTokensRemaining(uint256) should be declared external:
        - TraderRewards.setRewardTokensRemaining(uint256) (contracts/TraderRewards.sol#139-144)
Reentrancy in TraderRewards.safeSPHTransfer(address,uint256) (contracts/TraderRewards.sol#154-166):
        External calls:
        - success = sphToken.transfer(to,sphBal) (contracts/TraderRewards.sol#159)
        Event emitted after the call(s):
        - LogSafeSPHTransfer(to,sphBal) (contracts/TraderRewards.sol#160)
Reentrancy in TraderRewards.safeSPHTransfer(address,uint256) (contracts/TraderRewards.sol#154-166):
        External calls:
        - success = sphToken.transfer(to,amount) (contracts/TraderRewards.sol#163)
        Event emitted after the call(s):
        - LogSafeSPHTransfer(to,amount) (contracts/TraderRewards.sol#164)
Reentrancy in TraderRewards.withdrawRewardTokens() (contracts/TraderRewards.sol#91-98):
        External calls:
        - (success,amountTransferred) = safeSPHTransfer(msg.sender,amount) (contracts/TraderRewards.sol#95)
                - success = sphToken.transfer(to,sphBal) (contracts/TraderRewards.sol#159)
                - success = sphToken.transfer(to,amount) (contracts/TraderRewards.sol#163)
        Event emitted after the call(s):
        - LogWithdrawal(msg.sender,amountTransferred) (contracts/TraderRewards.sol#97)
```

According to the test results, some of the findings found by these tools were considered as false positives while some of these findings were real security concerns. All relevant findings were reviewed by the auditors and relevant findings addressed on the report as security concerns.

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. Only security-related findings are shown below.

Results:

SpheriumPair.sol, SpheriumERC20.sol

Report for SpheriumERC20.sol
https://dashboard.mythx.io/#/console/analyses/a1852878-9ba5-40c9-92f4-62d315ef8bc0
https://dashboard.mythx.io/#/console/analyses/53c0ba6c-ed32-4ed7-9444-867538b3df8d
https://dashboard.mythx.io/#/console/analyses/439f9515-3b0f-4fbc-8b11-989c1d71bc70

Line	SWC Title	Severity	Short Description
81	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.

Report for SpheriumPair.sol
https://dashboard.mythx.io/#/console/analyses/a1852878-9ba5-40c9-92f4-62d315ef8bc0

Line	SWC Title	Severity	Short Description
36	(SWC-107) Reentrancy	Low	Write to persistent state following external call
46	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
78	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.
195	(SWC-107) Reentrancy	Low	Read of persistent state following external call
200	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

SpheriumToken.sol, SpheriumToken-TGE.sol

Report for contracts/SpheriumToken-TGE.sol
https://dashboard.mythx.io/#/console/analyses/2feec77b-23b8-4c2b-bdfa-c5dff54ff18d

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.

Report for contracts/SpheriumToken.sol
https://dashboard.mythx.io/#/console/analyses/11a10d2a-4a4c-4e4c-b40e-b4a05a38498f

Line	SWC Title	Severity	Short Description
177	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
250	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

All relevant valid findings were founded in the manual code review.

THANK YOU FOR CHOOSING

// HALBORN