QuillAudits

Audit Report
June, 2021

SPHERIUM

Contents

Introduction

During the period of **April 27th, 2021 to May 1st, 2021** – QuillAudits Team performed security audit for Spherium smart contracts. The code for audit was taken from the following link:

- https://gitlab.com/spherium/uniswap/periphery/-/blob/master/contracts/UniswapV2Migrator.sol
- https://gitlab.com/spherium/uniswap/periphery/-/blob/master/contracts/UniswapV2Router02.sol
- https://gitlab.com/spherium/uniswap/periphery/-/blob/master/contracts/UniswapV2Router01.sol
- https://gitlab.com/spherium/uniswap/core-bsc/-/blob/master/contracts/UniswapV2Pair.sol
- https://gitlab.com/spherium/uniswap/core-bsc/-/blob/master/contracts/UniswapV2Factory.sol
- https://gitlab.com/spherium/uniswap/core-bsc/-/blob/master/contracts/UniswapV2ERC20.sol

Updated Smart Contracts:
Only the issues related to the core-bsc module were addressed.

- https://gitlab.com/spherium/uniswap/core-bsc/-/blob/master/contracts/SpheriumERC20.sol
  [Commit Hash: d4f62d35f1d5924e304a994b775a3fdb543adcfa]

- https://gitlab.com/spherium/uniswap/core-bsc/-/blob/master/contracts/SpheriumFactory.sol
  [Commit Hash: 7680798debc962ec83bac267620372169ebc4827]

- https://gitlab.com/spherium/uniswap/core-bsc/-/blob/master/contracts/SpheriumPair.sol
  [Commit Hash: d4f62d35f1d5924e304a994b775a3fdb543adcfa]

Overview of Contract

Spherium offers a complete suite of financial services comprising a universal wallet, token swap platform, money markets, and inter-blockchain liquidity transfer. Spherium will function as a global financial service provider, extending fundamental financial services to the unbanked.

Spherium Features

- **UNIFIED DeFi LANDSCAPE**
  SWAP + LENDING + BORROWING + STAKING + WALLET

- **BETTER REWARDING MECHANISM**
  Rewards for TRADERS + STAKERS

- **CROSS-CHAIN INTEROPERABILITY**
  Parallel implementation on BSC and Ethereum/Polygon

- **BSC and LAYER 2: LOW GAS FEE & INSTANT TRANSACTIONS**
  Binance Smart Chain is cheap and scalable. On Etherreum, L2 Built using ZK Roll ups and ZK Sync.

Spherium Ecosystem

Spherium offers the following products and services as a part of its Phase I deployment:

- **HyperSwap:**
  - Decentralized Asset Swap based on an automated market-making mechanism.
  - Layer 2 (or Application layer) protocols for mitigating the high transaction cost (Gas fee) while maintaining the security guarantee of layer 1 (or Settlement layer [Ref2]).
  - Cross Chain Swap using platforms such as RENVM.

- **SPH token:** Utility token that will be used for the governance of the Spherium ecosystem, with proposals and voting rights that will together determine its future. Will be used to incentivise early adopters of the Spherium ecosystem.

- **Spherium Wallet:** Decentralised wallet for Spherium products and services.

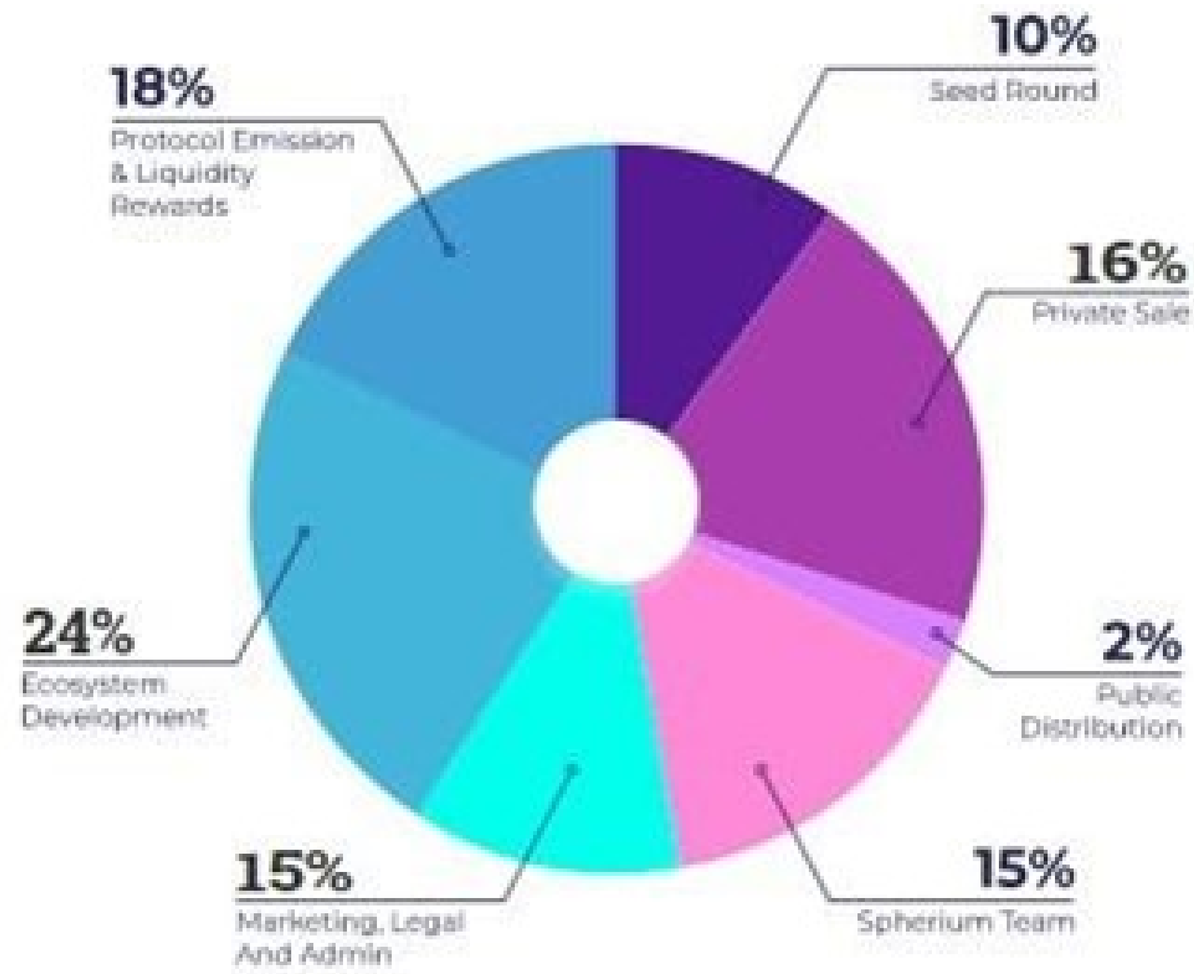
- **SphereComp:** Decentralized money markets where investors can lend or borrow digital assets with interest rates determined by the law of supply and demand

Spherium Token Distribution

Token name: SPHERIUM
Symbol ticker: SPH
Token type: ERC20
Total supply: 100 000 000

The process of the Spherium token allocation will be based on a community-type distribution, whereby each of these stakeholders will play an important part in the ecosystem.

Tokenomics



Details: www.spherium.finance
Pitch Deck: https://docsend.com/view/bgbeenay3cft4dkt
Whitepaper: https://www.spherium.finance/static/media/whitepaper.155bde27.pdf

Scope of Audit

The scope of this audit was to analyse **Spherium** smart contract codebase for quality, security, and correctness. Following is the list of smart contracts included in the scope of this audit:

- UniswapV2Migrator.sol
- UniswapV2Router02.sol
- UniswapV2Router01.sol
- UniswapV2Pair.sol
- UniswapV2Factory.sol
- UniswapV2ERC20.sol

Final Audit Scope:

- SpheriumMigrator.sol
- SpheriumRouter02.sol
- SpheriumRouter01.sol
- SpheriumPair.sol
- SpheriumFactory.sol
- SpheriumERC20.sol

OUT-OF-SCOPE: External contracts, External Oracles, other smart contracts in the repository or imported smart contracts, economic attacks.

Checked Vulnerabilities

We have scanned Spherium smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level
- Address hardcoded
- Using delete for arrays
- Integer overflow/
- underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility
- Using blockhash
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of Spherium smart contracts care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

A combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts 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:

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the Smart contract is structured in a way that will not result in future problems.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Static Analysis
Static Analysis of smart contracts was done to identify contract vulnerabilities. In this step series of automated tools are used to test the security of smart contracts.

Gas Consumption
In this step, we have checked the behaviour of smart contract in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Manticore, Slither.

Issue Categories

High severity issues

Issues that must be fixed before deployment else they can create major issues.

Medium level severity issues

These issues will not create major issues in working but affect the performance of the smart contract.

Low level severity issues

These issues are more suggestions that should be implemented to refine the code in terms of gas, fees, speed and code accuracy

Number of issues per severity

	High	Medium	Low	Informational
Open	0	2	28	8
Closed	0	2	7	0

Findings and Tech Details

A. SpheriumMigrator.sol

Low level severity issues

1. Consider using the latest stable solidity version

   Code Lines: 1
   The smart contract is using an old version of solidity. It is highly recommended to use the latest stable version of solidity.

   Auditors Remarks: **Acknowledged by the developer**.

2. **Coding Style Issues**

   Coding style issues influence code readability and, in some cases, may lead to bugs in future. Smart Contracts have a naming convention, indentation and code layout issues. It’s recommended to use the Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

   Auditors Remarks: **Acknowledged by the developer.**

3. **Order of layout**

   The order of functions as well as the rest of the code layout does not follow solidity style guide.
   Layout contract elements in the following order:
   - Pragma statements
   - Import statements
   - Interfaces
   - Libraries
   - Contracts

   Inside each contract, library or interface, use the following order:
   - Type declarations
   - State variables
   - Events
   - Functions

Please read the following documentation links to understand the correct order:

https://solidity.readthedocs.io/en/v0.6.6/style-guide.html#order-of-layout

Auditors Remarks: Acknowledged by the developer.

4. **Use of different compiler versions of Solidity**

   Different compiler versions of solidity are in use - Version used: ['=0.6.6', '>=0.5.0', '>=0.6.0', '>=0.6.2']
   - =0.6.6 [UniswapV2Migrator.sol]
   - >=0.5.0 [interfaces\IERC20.sol]
   - >=0.5.0 [interfaces\IUniswapV2Migrator.sol]
   - >=0.6.2 [interfaces\IUniswapV2Router01.sol]
   - >=0.5.0 [interfaces\V1\IUniswapV1Exchange.sol]
   - >=0.5.0 [interfaces\V1\IUniswapV1Factory.sol]
   - >=0.6.0 [libraries\TransferHelper.sol]

   It is recommended to a single fixed compiler version in all files including the local OpenZeppelin contracts or Uniswap libraries.

   Auditors Remarks: Fixed

5. **Be explicit about which `uint` the code is using to avoid overflow and underflow issues**

   `uint` is an alias for `uint256`, but using the full form is preferable. Be consistent and use one of the forms. Explicitly declaring data types is recommended to avoid unexpected behaviours and/or errors

   Auditors Remarks: Acknowledged by the developer.

B. **SpheriumRouter01.sol**

Low level severity issues

1. **Consider using the latest stable solidity version**

   Code Lines: 1
   The smart contract is using an old version of solidity. It is highly recommended to use the latest stable version of solidity.

   Auditors Remarks: Acknowledged by the developer.

2. Coding Style Issues

Coding style issues influence code readability and, in some cases, may lead to bugs in future. Smart Contracts have a naming convention, indentation and code layout issues. It's recommended to use the Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

```
UniswapV2Router01.sol
   95:2  error  Line length must be no more than 120 but current length is 128  max-line-length
  176:2  error  Line length must be no more than 120 but current length is 124  max-line-length
  188:2  error  Line length must be no more than 120 but current length is 126  max-line-length
  200:2  error  Line length must be no more than 120 but current length is 126  max-line-length
  226:2  error  Line length must be no more than 120 but current length is 126  max-line-length
  240:2  error  Line length must be no more than 120 but current length is 126  max-line-length
  258:2  error  Line length must be no more than 120 but current length is 130  max-line-length
  265:2  error  Line length must be no more than 120 but current length is 121  max-line-length

✖ 8 problems (8 errors, 0 warnings)
```

Auditors Remarks: Acknowledged by the developer.

3. Order of layout

The order of functions as well as the rest of the code layout does not follow solidity style guide.

Layout contract elements in the following order:

- Pragma statements
- Import statements
- Interfaces
- Libraries
- Contracts

Inside each contract, library or interface, use the following order:

- Type declarations
- State variables
- Events
- Functions

Please read the following documentation links to understand the correct order:

https://solidity.readthedocs.io/en/v0.6.6/style-guide.html#order-of-layout

Auditors Remarks: Acknowledged by the developer.

4. **Use of different compiler versions of Solidity**

Different compiler versions of solidity are in use - Version used: ['=0.6.6', '>=0.5.0', '>=0.6.0', '>=0.6.2']

- =0.6.6 [UniswapV2Router01.sol]
- >=0.5.0 [interfaces\IERC20.sol]
- >=0.5.0 [interfaces\IUniswapV2Factory.sol]
- >=0.5.0 [interfaces\IUniswapV2Pair.sol]
- >=0.6.2 [interfaces\IUniswapV2Router01.sol]
- >=0.5.0 [interfaces\IWETH.sol]
- =0.6.6 [libraries\SafeMath.sol]
- >=0.6.0 [libraries\TransferHelper.sol]
- >=0.5.0 [libraries\UniswapV2Library.sol]

It is recommended to a single fixed compiler version in all files including the local OpenZeppelin contracts or Uniswap libraries.

Auditors Remarks: Fixed

5. **Use external function modifier instead of public**

The public functions that are never called by contract should be declared external to save gas. Following functions can be declared external:

quote(uint256,uint256,uint256) [#263-265]
getAmountOut(uint256,uint256,uint256) [#267-269]
getAmountIn(uint256,uint256,uint256) [#271-273]
getAmountsOut(uint256,address[]) [#275-277]
getAmountsIn(uint256,address[]) [#279-281]

Auditors Remarks: Acknowledged by the developer.

6. **External calls inside a loop can lead to Denial-of-Service attack**

Code Lines: 168-177

External calls can fail accidentally or deliberately. To minimize the damage caused by such failures, it is often better to isolate each external call into its own transaction that can be initiated by the recipient of the call. This is especially relevant for payments, where it is better to let users withdraw funds rather than push funds to them automatically

Favor Pull over push external calls - https://consensys.github.io/smart-contract-best-practices/recommendations/#favor-pull-over-push-for-external-calls

Auditors Remarks: Acknowledged by the developer.

7. **Use local variable instead of state variable like length in a loop**

   Code Lines: 169

   For every iteration of for loop - state variables like length of non-memory array will consume extra gas. To reduce the gas consumption, it's advisable to use local variable.
   Links: Gas Limits and loops

   Auditors Remarks: Acknowledged by the developer.

8. **Be explicit about which `uint` the code is using to avoid overflow and underflow issues**

   `uint` is an alias for `uint256`, but using the full form is preferable. Be consistent and use one of the forms. Explicitly declaring data types is recommended to avoid unexpected behaviours and/or errors.

   Auditors Remarks: Acknowledged by the developer.

C. SpheriumRouter02.sol

Medium severity issues

1. **Consider downsizing the contract size**

   In Ethereum hard fork 4.0 – Spurious Dragon - Contract code size limit was set to 24576 bytes to avoid Denial-of-Service attack. The contract throws an warning that the contract size exceeds 24576 bytes. Consider downsizing the contract.
   Remediations:
   Downsizing contract: https://soliditydeveloper.com/max-contract-size
   Spurious Dragon: https://blog.ethereum.org/2016/11/18/hard-fork-no-4-spurious-dragon/
   EIP-170: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-170.md

   Auditors Remarks: Acknowledged by the developer.

Low level severity issues

2. **Consider using latest stable solidity version**

   Code Lines: 1
   The smart contract is using an old version of solidity. It is highly recommended to use the latest stable version of solidity.

   Auditors Remarks: Acknowledged by the developer.

3. **Coding Style Issues**

   Coding style issues influence code readability and, in some cases, may lead to bugs in future. Smart Contracts have naming convention, indentation and code layout issues. It's recommended to use Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

   Auditors Remarks: Acknowledged by the developer.

4. **Order of layout**

   The order of functions as well as the rest of the code layout does not follow solidity style guide.
   Layout contract elements in the following order:
   - Pragma statements
   - Import statements
   - Interfaces
   - Libraries
   - Contracts

   Inside each contract, library or interface, use the following order:
   - Type declarations
   - State variables
   - Events
   - Functions

   Please read the following documentation links to understand the correct order:
   https://solidity.readthedocs.io/en/v0.6.6/style-guide.html#order-of-layout

   Auditors Remarks: Acknowledged by the developer.

5. **Use of different compiler versions of Solidity**

Different compiler versions of solidity are in use - - Version used: ['=0.6.6', '>=0.5.0', '>=0.6.0', '>=0.6.2']

- =0.6.6 [UniswapV2Router02.sol]
- >=0.5.0 [interfaces\IERC20.sol]
- >=0.5.0 [interfaces\IUniswapV2Factory.sol]
- >=0.5.0 [interfaces\IUniswapV2Pair.sol]
- >=0.6.2 [interfaces\IUniswapV2Router01.sol]
- >=0.6.2 [interfaces\IUniswapV2Router02.sol]
- >=0.5.0 [interfaces\IWETH.sol]
- =0.6.6 [libraries\SafeMath.sol]
- >=0.6.0 [libraries\TransferHelper.sol]
- >=0.5.0 [libraries\UniswapV2Library.sol]

It is recommended to a single fixed compiler version in all files including the local OpenZeppelin contracts or Uniswap libraries.

Auditors Remarks: Fixed

6. **Use external function modifier instead of public**

The public functions that are never called by contract should be declared external to save gas. Following functions can be declared external:

- quote(uint256,uint256,uint256)
- getAmountOut(uint256,uint256,uint256)
- getAmountIn(uint256,uint256,uint256)
- getAmountsOut(uint256,address[])
- getAmountsIn(uint256,address[])

Auditors Remarks: Acknowledged by the developer.

7. **External calls inside a loop can lead to Denial-of-Service attack**

Code Lines: 212-222

External calls can fail accidentally or deliberately. To minimize the damage caused by such failures, it is often better to isolate each external call into its own transaction that can be initiated by the recipient of the call. This is especially relevant for payments, where it is better to let users withdraw funds rather than push funds to them automatically.

Favor Pull over push external calls - https://consensys.github.io/smart-contract-best-practices/recommendations/#favor-pull-over-push-for-external-calls

Auditors Remarks: Acknowledged by the developer.

8. **Use local variable instead of state variable like length in a loop**

   Code Lines: 213, 322
   For every iteration of for loop - state variables like length of non-memory array will consume extra gas. To reduce the gas consumption, it's advisable to use local variable.
   Links: Gas Limits and loops

   Auditors Remarks: Acknowledged by the developer.

9. **Be explicit about which `uint` the code is using to avoid overflow and underflow issues**

   `uint` is an alias for `uint256`, but using the full form is preferable. Be consistent and use one of the forms. Explicitly declaring data types is recommended to avoid unexpected behaviours and/or errors.

   Auditors Remarks: Acknowledged by the developer.

D. SpheriumERC20.sol

Low level severity issues

1. **Consider using the latest stable solidity version**

   Code Lines: 1
   The smart contract is using an old version of solidity. It is highly recommended to use the latest stable version of solidity.

   Auditors Remarks: Acknowledged by the developer.

2. Coding Style Issues

Coding style issues influence code readability and, in some cases, may lead to bugs in future. Smart Contracts have a naming convention, indentation and code layout issues. It's recommended to use the Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

Auditors Remarks: Acknowledged by the developer.

3. Order of layout

The order of functions as well as the rest of the code layout does not follow solidity style guide.
Layout contract elements in the following order:

- Pragma statements
- Import statements
- Interfaces
- Libraries
- Contracts

Inside each contract, library or interface, use the following order:

- Type declarations
- State variables
- Events
- Functions

Please read the following documentation links to understand the correct order:

https://solidity.readthedocs.io/en/v0.6.6/style-guide.html#order-of-layout

Auditors Remarks: Acknowledged by the developer.

4. **Use of different compiler versions of Solidity**

   Different compiler versions of solidity are in use - Version used: ['=0.5.16', '>=0.5.0']
   - =0.5.16 [UniswapV2ERC20.sol]
   - >=0.5.0 [interfaces\IUniswapV2ERC20.sol]
   - =0.5.16 [libraries\SafeMath.sol]

   It is recommended to a single fixed compiler version in all files including the local OpenZeppelin contracts or Uniswap libraries.

   Auditors Remarks: Fixed

5. **Consider removing the dead code or commented code from the contract**

   Code Lines: 168-177

   Auditors Remarks: Fixed

6. **Be explicit about which `uint` the code is using to avoid overflow and underflow issues**

   `uint` is an alias for `uint256`, but using the full form is preferable. Be consistent and use one of the forms. Explicitly declaring data types is recommended to avoid unexpected behaviours and/or errors.

   Auditors Remarks: Acknowledged by the developer.

Informational

7. **Use of block.timestamp should be avoided**

   Code Lines: 82
   Do not use block.timestamp, now or blockhash as a source of randomness. Malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. Contracts often need access to the current timestamp to trigger time-dependent events. As Ethereum is decentralized, nodes can synchronize time only to some degree. It is risky to use block.timestamp, as block.timestamp can be manipulated by
   miners. Therefore block.timestamp is recommended to be supplemented with some other strategy in the case of high-value/risk applications.

Remediations:
https://consensys.github.io/smart-contract-best-practices/recommendations/#timestamp-dependence
https://consensys.github.io/smart-contract-best-practices/recommendations/#avoid-using-blocknumber-as-a-timestamp

8. Approve function of ERC-20 is vulnerable

A security issue called “Multiple Withdrawal Attack” - originates from two methods in the ERC20 standard for approving and transferring tokens. The use of these functions in an adverse environment (e.g., front-running) could result in more tokens being spent than what was intended. This issue is still open on the GitHub and several solutions have been made to mitigate it.
For more details on how to resolve the multiple withdrawal attack check the following document for details:
https://drive.google.com/file/d/1skR4BpZ0VBSQICIC_eqRBgGnACf2li5X/view?usp=sharing
https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit
https://blog.smartdec.net/erc20-approve-issue-in-simple-words-a41aaf47bca6
https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729

9. The use of assembly is error-prone and should be avoided.

Code Lines: 26
Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This bypasses several important safety features and checks of Solidity. You should only use it for tasks that need it, and only if you are confident with using it.0x vulnerability was caused by assembly code put there to optimize space. For security critical applications assembly is not a good idea. Assembly code is harder to read/audit, and it’s also just harder to get right.
Examples of how inline assembly can be exploited:
https://samczsun.com/the-0x-vulnerability-explained/
https://medium.com/consensys-diligence/return-data-length-validation-a-bug-we-missed-4b7bbea8e9ab
https://blog.0xproject.com/post-mortem-0x-v2-0-exchange-vulnerability-763015399578

E. SpheriumFactory.sol

Medium severity issues

1. Reentrancy

   Code Lines:
   createPair(address,address) [#23-38]

   One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. It's recommended that the calls to external functions/events should happen after any changes to state variables in your contract so your contract is not vulnerable to a reentrancy exploit. When control is transferred to recipient, care must be taken to not create reentrancy vulnerabilities. OpenZeppelin has its own mutex implementation you can use called ReentrancyGuard. This library provides a modifier you can apply to any function called nonReentrant that guards the function with a mutex.

   Consider using ReentrancyGuard or the checks-effects-interactions pattern.

   Following link explains more about Reentrancy and ReentrancyGuard
   https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard
   https://blog.openzeppelin.com/reentrancy-after-istanbul/

   Most Recent DeFi Reentrancy Attack: DFORCE

   Auditors Remarks: Fixed

2. **Function createPair has return type but there is no return statement inside the function**

   Code Lines: 23-38

   The function createPair has return types but the return statement is missing inside the function. It is recommended to add a return statement inside the function.

   Auditors Remarks: Fixed

Low level severity issues

3. **Consider using the latest stable solidity version**

   Code Lines: 1
   The smart contract is using an old version of solidity. It is highly recommended to use the latest stable version of solidity.

   Auditors Remarks: Acknowledged by the developer.

4. **Coding Style Issues**

   Coding style issues influence code readability and, in some cases, may lead to bugs in future. Smart Contracts have a naming convention, indentation and code layout issues. It's recommended to use the Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

   Auditors Remarks: Acknowledged by the developer.

5. **Use of different compiler versions of Solidity**

   Different compiler versions of solidity are in use - Version used: ['=0.5.16', '>=0.5.0']
   - =0.5.16 [UniswapV2ERC20.sol]
   - =0.5.16 [UniswapV2Factory.sol]
   - =0.5.16 [UniswapV2Pair.sol]
   - >=0.5.0 [interfaces\IERC20.sol]
   - >=0.5.0 [interfaces\IUniswapV2Callee.sol]
   - >=0.5.0 [interfaces\IUniswapV2ERC20.sol]
   - >=0.5.0 [interfaces\IUniswapV2Factory.sol]
   - >=0.5.0 [interfaces\IUniswapV2Pair.sol]
   - =0.5.16 [libraries\Math.sol]
   - =0.5.16 [libraries\SafeMath.sol]
   - =0.5.16 [libraries\UQ112x112.sol]

   It is recommended to a single fixed compiler version in all files including the local OpenZeppelin contracts or Uniswap libraries.

   Auditors Remarks: Fixed

Informational

6. The use of assembly is error-prone and should be avoided.

Code Lines: 30
Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This bypasses several important safety features and checks of Solidity. You should only use it for tasks that need it, and only if you are confident with using it.0x vulnerability was caused by assembly code put there to optimize space. For security critical applications assembly is not a good idea. Assembly code is harder to read/audit, and it's also just harder to get right.
Examples of how inline assembly can be exploited:
https://samczsun.com/the-0x-vulnerability-explained/
https://medium.com/consensys-diligence/return-data-length-validation-a-bug-we-missed-4b7bbea8e9ab
https://blog.0xproject.com/post-mortem-0x-v2-0-exchange-vulnerability-763015399578

7. Overpowered owner

Code Lines: 40, 45
The contract is tightly coupled to the owner, making some functions callable only by the owner's address. This poses a serious risk: if the private key of the owner gets compromised, then an attacker can gain control over the contract.

F. SpheriumPair.sol

Medium severity issues

1. Reentrancy

   Code Lines:
   burn(address) [#134-156]
   swap(uint256,uint256,address,bytes) [#159-187]

   One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. It's recommended that the calls to external functions/events should happen after any changes to state variables in your contract so your contract is not vulnerable to a reentrancy exploit. When control is transferred to recipient, care must be taken to not create reentrancy vulnerabilities. OpenZeppelin has its own mutex implementation you can use called ReentrancyGuard. This library provides a modifier you can apply to any function called nonReentrant that guards the function with a mutex.
   Consider using ReentrancyGuard or the checks-effects-interactions pattern.

   Following link explains more about Reentrancy and ReentrancyGuard
   https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard
   https://blog.openzeppelin.com/reentrancy-after-istanbul/

   Most Recent DeFi Reentrancy Attack: DFORCE

   Auditors Remarks: Acknowledged by the developer.

Low level severity issues

2. **Consider using the latest stable solidity version**

   Code Lines: 1
   The smart contract is using an old version of solidity. It is highly recommended to use the latest stable version of solidity.

   Auditors Remarks: Acknowledged by the developer.

3. Coding Style Issues

Coding style issues influence code readability and, in some cases, may lead to bugs in future. Smart Contracts have a naming convention, indentation and code layout issues. It's recommended to use the Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

Auditors Remarks: Acknowledged by the developer.

4. Order of layout

The order of functions as well as the rest of the code layout does not follow solidity style guide.
Layout contract elements in the following order:

- Pragma statements
- Import statements
- Interfaces
- Libraries
- Contracts

Inside each contract, library or interface, use the following order:

- Type declarations
- State variables
- Events
- Functions

Please read the following documentation links to understand the correct order:

https://solidity.readthedocs.io/en/v0.6.6/style-guide.html#order-of-layout

Auditors Remarks: Acknowledged by the developer.

5. **Use of different compiler versions of Solidity**

   Different compiler versions of solidity are in use - Version used: ['=0.5.16', '>=0.5.0']
   - =0.5.16 [UniswapV2ERC20.sol]
   - =0.5.16 [UniswapV2Pair.sol]
   - >=0.5.0 [interfaces\IERC20.sol]
   - >=0.5.0 [interfaces\IUniswapV2Callee.sol]
   - >=0.5.0 [interfaces\IUniswapV2ERC20.sol]
   - >=0.5.0 [interfaces\IUniswapV2Factory.sol]
   - >=0.5.0 [interfaces\IUniswapV2Pair.sol]
   - =0.5.16 [libraries\Math.sol]
   - =0.5.16 [libraries\SafeMath.sol]
   - =0.5.16 [libraries\UQ112x112.sol]

   It is recommended to a single fixed compiler version in all files including the local OpenZeppelin contracts or Uniswap libraries.

   Auditors Remarks: Fixed

6. **Be explicit about which `uint` the code is using to avoid overflow and underflow issues**

   `uint` is an alias for `uint256`, but using the full form is preferable. Be consistent and use one of the forms. Explicitly declaring data types is recommended to avoid unexpected behaviours and/or errors.

   Auditors Remarks: **Acknowledged by the developer.**

Informational

7. **Use of block.timestamp should be avoided**

   Code Lines: 73-86
   Do not use block.timestamp, now or blockhash as a source of randomness. Malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. Contracts often need access to the current timestamp to trigger time-dependent events. As Ethereum is decentralized, nodes can synchronize time only to some degree. It is risky to use block.timestamp, as block.timestamp can be manipulated by
   miners. Therefore block.timestamp is recommended to be supplemented with some other strategy in the case of high-value/risk applications.

Remediations:
https://consensys.github.io/smart-contract-best-practices/recommendations/#timestamp-dependence
https://consensys.github.io/smart-contract-best-practices/recommendations/#avoid-using-blocknumber-as-a-timestamp

8. Low level calls

Code Lines: 45
Low-level calls do not check for code existence or call success. Use of low-level calls should be avoided as they are error-prone.

9. Gas optimization tips

As the gas costs are increasing it is highly recommended to implement gas optimization techniques to reduce gas consumption.
https://blog.polymath.network/solidity-tips-and-tricks-to-save-gas-and-reduce-bytecode-size-c44580b218e6

Automated Testing

Slither

Slither is an open-source Solidity static analysis framework. This tool provides rich information about Ethereum smart contracts and has the critical properties. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

UniswapV2ERC20

```
INFO:Printers:
Compiled with solc
Number of lines: 134 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 1
Number of contracts: 3 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 6
Number of low issues: 1
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+----------------+-------------+-------+---------------------+--------------+-----------+
|      Name      | # functions |  ERCS |     ERC20 info      | Complex code |  Features |
+----------------+-------------+-------+---------------------+--------------+-----------+
| UniswapV2ERC20 |      23     | ERC20 |     No Minting      |      No      | Ecrecover |
|                |             |       | Approve Race Cond.  |              |  Assembly |
|                |             |       |                     |              |           |
|    SafeMath    |      3      |       |                     |      No      |           |
+----------------+-------------+-------+---------------------+--------------+-----------+
INFO:Slither:UniswapV2ERC20.sol analyzed (3 contracts)
```

UniswapV2Factory

```
INFO:Printers:
Compiled with solc
Number of lines: 518 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 4
Number of contracts: 11 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 22
Number of low issues: 13
Number of medium issues: 5
Number of high issues: 1
ERCs: ERC20

+--------------------+-------------+-------+----------------------+--------------+----------------------+
|        Name        | # functions | ERCS  |      ERC20 info      | Complex code |       Features       |
+--------------------+-------------+-------+----------------------+--------------+----------------------+
| UniswapV2Factory   |      13     |       |                      |      No      | Tokens interaction   |
|                    |             |       |                      |              |      Assembly        |
|  UniswapV2Pair     |      62     | ERC20 |     ∞ Minting        |      No      |      Ecrecover       |
|                    |             |       | Approve Race Cond.   |              | Tokens interaction   |
|                    |             |       |                      |              |      Assembly        |
|       IERC20       |      9      | ERC20 |     No Minting       |      No      |                      |
|                    |             |       | Approve Race Cond.   |              |                      |
|                    |             |       |                      |              |                      |
| IUniswapV2Callee   |      1      |       |                      |      No      |                      |
|       Math         |      2      |       |                      |      No      |                      |
|     SafeMath       |      3      |       |                      |      No      |                      |
|    UQ112x112       |      3      |       |                      |      No      |                      |
+--------------------+-------------+-------+----------------------+--------------+----------------------+
INFO:Slither:UniswapV2Factory.sol analyzed (11 contracts)
```

UniswapV2pair

```
INFO:Printers:
Compiled with solc
Number of lines: 469 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 1
Number of contracts: 10 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 18
Number of low issues: 8
Number of medium issues: 4
Number of high issues: 1
ERCs: ERC20

+--------------------+-------------+-------+----------------------+--------------+----------------------+
|        Name        | # functions | ERCS  |      ERC20 info      | Complex code |       Features       |
+--------------------+-------------+-------+----------------------+--------------+----------------------+
|   UniswapV2Pair    |      62     | ERC20 |     ∞ Minting        |      No      |      Ecrecover       |
|                    |             |       | Approve Race Cond.   |              | Tokens interaction   |
|                    |             |       |                      |              |      Assembly        |
|       IERC20       |      9      | ERC20 |     No Minting       |      No      |                      |
|                    |             |       | Approve Race Cond.   |              |                      |
|                    |             |       |                      |              |                      |
|  IUniswapV2Callee  |      1      |       |                      |      No      |                      |
| IUniswapV2Factory  |      8      |       |                      |      No      |                      |
|       Math         |      2      |       |                      |      No      |                      |
|     SafeMath       |      3      |       |                      |      No      |                      |
|    UQ112x112       |      3      |       |                      |      No      |                      |
+--------------------+-------------+-------+----------------------+--------------+----------------------+
INFO:Slither:UniswapV2Pair.sol analyzed (10 contracts)
```

UniswapV2Migrator

```
INFO:Printers:
Compiled with solc
Number of lines: 230 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 7 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 17
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+--------------------+-------------+-------+----------------------+--------------+---------------------+
|        Name        | # functions | ERCS  |     ERC20 info       | Complex code |      Features       |
+--------------------+-------------+-------+----------------------+--------------+---------------------+
| UniswapV2Migrator  |      4      |       |                      |      No      |    Receive ETH      |
|                    |             |       |                      |              |      Send ETH       |
|                    |             |       |                      |              | Tokens interaction  |
|       IERC20       |      9      | ERC20 |     No Minting       |      No      |                     |
|                    |             |       | Approve Race Cond.   |              |                     |
|                    |             |       |                      |              |                     |
| IUniswapV2Router01 |      19     |       |                      |      No      |    Receive ETH      |
| IUniswapV1Exchange |      5      |       |                      |      No      |    Receive ETH      |
| IUniswapV1Factory  |      1      |       |                      |      No      |                     |
|   TransferHelper   |      4      |       |                      |      No      |      Send ETH       |
+--------------------+-------------+-------+----------------------+--------------+---------------------+
INFO:Slither:UniswapV2Migrator.sol analyzed (7 contracts)
```

UniswapV2Router01

```
INFO:Printers:
Compiled with solc
Number of lines: 620 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 9 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 5
Number of informational issues: 30
Number of low issues: 3
Number of medium issues: 4
Number of high issues: 0

ERCs: ERC20

+--------------------+-------------+-------+----------------------+--------------+---------------------+
|        Name        | # functions | ERCS  |     ERC20 info       | Complex code |      Features       |
+--------------------+-------------+-------+----------------------+--------------+---------------------+
| UniswapV2Router01  |      40     |       |                      |      No      |    Receive ETH      |
|                    |             |       |                      |              |      Send ETH       |
|                    |             |       |                      |              | Tokens interaction  |
|       IERC20       |      9      | ERC20 |     No Minting       |      No      |                     |
|                    |             |       | Approve Race Cond.   |              |                     |
|                    |             |       |                      |              |                     |
| IUniswapV2Factory  |      8      |       |                      |      No      |                     |
|   IUniswapV2Pair   |      27     | ERC20 |     ∞ Minting        |      No      |                     |
|                    |             |       | Approve Race Cond.   |              |                     |
|                    |             |       |                      |              |                     |
|       IWETH        |      3      |       |                      |      No      |    Receive ETH      |
|      SafeMath      |      3      |       |                      |      No      |                     |
|   TransferHelper   |      4      |       |                      |      No      |      Send ETH       |
|  UniswapV2Library  |      8      |       |                      |      No      | Tokens interaction  |
+--------------------+-------------+-------+----------------------+--------------+---------------------+
INFO:Slither:UniswapV2Router01.sol analyzed (9 contracts)
```

UniswapV2Router02

```
INFO:Printers:
Compiled with solc
Number of lines: 831 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 10 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 5
Number of informational issues: 31
Number of low issues: 6
Number of medium issues: 5
Number of high issues: 0

ERCs: ERC20

+--------------------+-------------+-------+----------------------+--------------+----------------------+
|        Name        | # functions | ERCS  |     ERC20 info       | Complex code |      Features        |
+--------------------+-------------+-------+----------------------+--------------+----------------------+
| UniswapV2Router02  |      51     |       |                      |      No      |    Receive ETH       |
|                    |             |       |                      |              |      Send ETH        |
|                    |             |       |                      |              | Tokens interaction   |
|       IERC20       |      9      | ERC20 |     No Minting       |      No      |                      |
|                    |             |       | Approve Race Cond.   |              |                      |
|                    |             |       |                      |              |                      |
| IUniswapV2Factory  |      8      |       |                      |      No      |                      |
|   IUniswapV2Pair   |      27     | ERC20 |     ∞ Minting        |      No      |                      |
|                    |             |       | Approve Race Cond.   |              |                      |
|                    |             |       |                      |              |                      |
|       IWETH        |      3      |       |                      |      No      |    Receive ETH       |
|      SafeMath      |      3      |       |                      |      No      |                      |
|   TransferHelper   |      4      |       |                      |      No      |      Send ETH        |
|  UniswapV2Library  |      8      |       |                      |      No      | Tokens interaction   |
+--------------------+-------------+-------+----------------------+--------------+----------------------+
INFO:Slither:UniswapV2Router02.sol analyzed (10 contracts)
```

Slither didn't raise any critical issue with smart contracts. The smart contracts were well tested and all the minor issues that were raised have been documented in the report. Also, all other vulnerabilities of importance have already been covered in the manual audit section of the report.

Mythril

Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum. It uses symbolic execution, SMT solving and taint analysis to detect a variety of security vulnerabilities.

Mythril did not detect any high severity issue. All the considerable issues raised by Mythril are already covered in the **Issues Found** section of this report.

Closing Summary

Overall, the smart contract code is extremely well documented, follows a high-quality software development standard, contains many utilities and automation scripts to support continuous deployment / testing / integration, and does NOT contain any obvious exploitation vectors that QuillAudits was able to leverage within the timeframe of testing allotted. Overall, the smart contracts adhered to ERC20 guidelines. No critical or major vulnerabilities were found in the audit. Several issues of **medium severity and low severity were found and reported during the audit.**

The outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verification of essential properties were performed to achieve objectives and deliverables set in the scope. QuillAudits recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts.

Disclaimer

QuillHash audit is not a security warranty, investment advice, or an endorsement of the **Spherium platform**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Spherium Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

SPHERIUM

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com