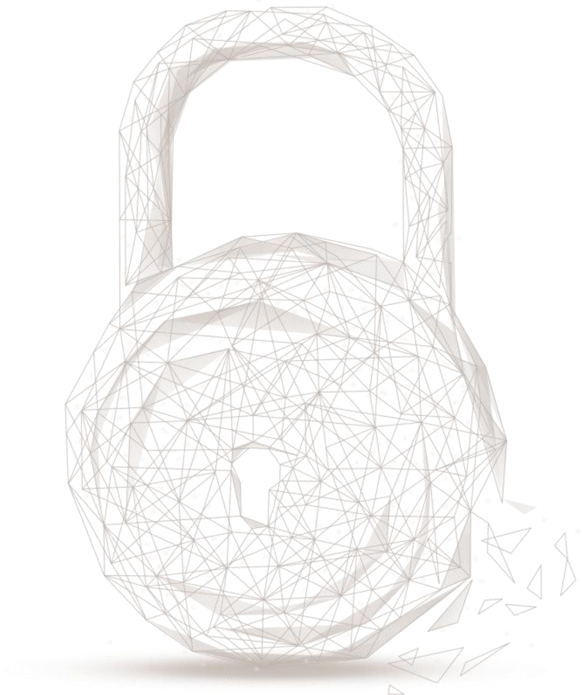
Beosin

BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number：202102051711

Smart Contract Info：

Smart Contract Name	Smart Contract Address	Smart Contract Address Link
LavaSwapERC20	0x1ea26c17d061e8a7cc33b20d8d8dad131d7fb392	https://scan.hecochain.com/address/0x1ea26c17d061e8a7cc33b20d8d8dad131d7fb392#contracts
LavaSwapRouter02	0xe38623b265b5acc9f35e696381769e556ed932f9	https://scan.hecochain.com/address/0xe38623b265b5acc9f35e696381769e556ed932f9#contracts

Start Date：2021.02.02

Completion Date：2021.02.05

Overall Result：Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass

		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Note: Audit results and suggestions in code comments

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contracts LavaSwapERC20&LavaSwapRouter02, including Coding Standards, Security, and Business Logic. **The LavaSwapERC20&LavaSwapRouter02 contracts passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing HT.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract. In this project, the contract
- Result: Pass

2.10 Replay Attack

- Description: Check whether the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

3.1 Business analysis of Contract LavaSwapERC20

(1) Basic Token Information

- Basic Token Information

Token name	LavaSwap
Token symbol	LavaSwap
decimals	18
totalSupply	Mintable without cap, burnable
Token type	HRC20

Table 1 Basic Token Information

(2) HRC20 Token Standard Functions

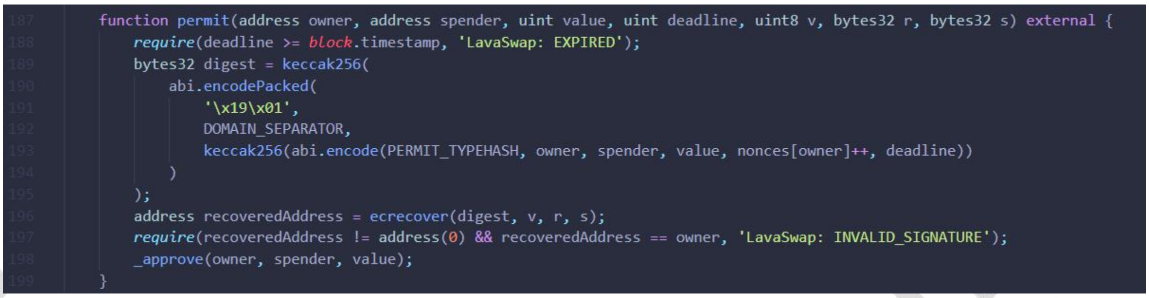
- Description: The Token Contract implements a Token which conforms to the HRC20 Standards. It should be noted that the user can directly call the *approve* function to set the approval value for the specified address, but in order to avoid multiple authorizations, it is recommended to reset the authorization value to 0 before making a new authorization.

- Related functions: *name, symbol, decimals, totalSupply, balanceOf, allowance, transfer, transferFrom, approve*

- Result: Pass

(3) permit function

- Description: The contract implements the *permit* function to obtain the specified authorization. It will first perform a deadline judgment, and then verify the signature to achieve the token authorization between the specified addresses.



```
function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external {
    require(deadline >= block.timestamp, 'LavaSwap: EXPIRED');
    bytes32 digest = keccak256(
        abi.encodePacked(
            '\x19\x01',
            DOMAIN_SEPARATOR,
            keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++, deadline))
        )
    );
    address recoveredAddress = ecrecover(digest, v, r, s);
    require(recoveredAddress != address(0) && recoveredAddress == owner, 'LavaSwap: INVALID_SIGNATURE');
    _approve(owner, spender, value);
}
```

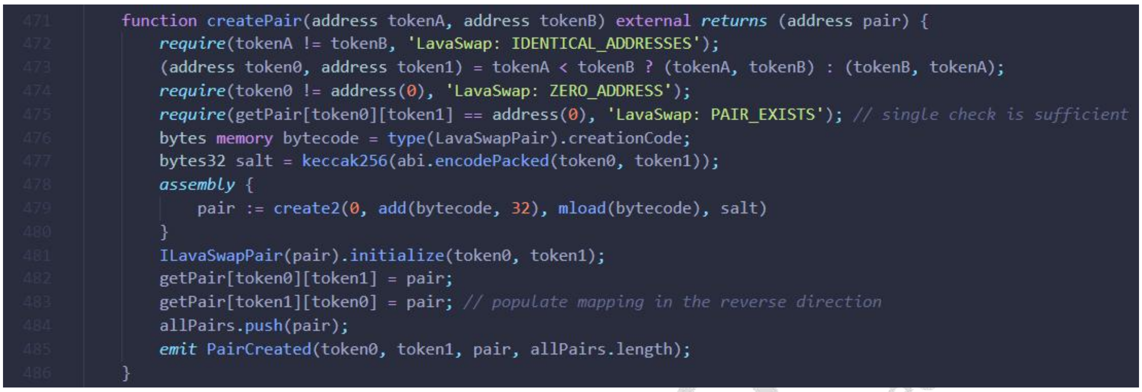
Figure 1 Source code of *permit*

- Related functions: *permit*
- Result: Pass

3.2 Business analysis of Contract LavaSwapFactory

(1) createPair function

- Description: The contract implements that *createPair* is used to create a transaction pair. Users can call this function to create a new transaction pair (requires that the transaction pair of the current two tokens does not exist, and the addresses of the two tokens passed are different and not zero) and create a contract of the transaction pair. Call the initialize of the created pair contract to *initialize* the addresses of the two tokens and update the allPairs information.



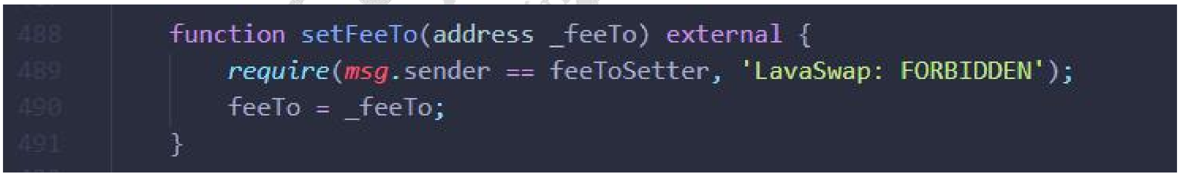
```
    function createPair(address tokenA, address tokenB) external returns (address pair) {
        require(tokenA != tokenB, 'LavaSwap: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'LavaSwap: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'LavaSwap: PAIR_EXISTS'); // single check is sufficient
        bytes memory bytecode = type(LavaSwapPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        ILavaSwapPair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }
```

Figure 2 Source code of *createPair*

- Related functions: *createPair, initialize*
- Result: Pass

(2) setFeeTo function

- Description: The contract implements *setFeeTo* to change the fee collection address, requiring the caller to be *feeToSetter*.



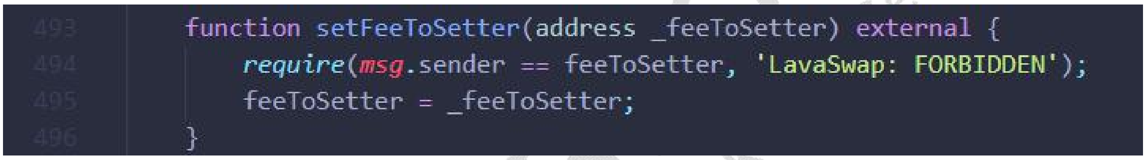
```
    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'LavaSwap: FORBIDDEN');
        feeTo = _feeTo;
    }
```

Figure 3 Source code of *setFeeTo*

- Related functions: *setFeeTo*
- Result: Pass

(3) setFeeToSetter function

- Description: The contract implements *setFeeToSetter* to change the feeToSetter address, requiring the caller to be feeToSetter.



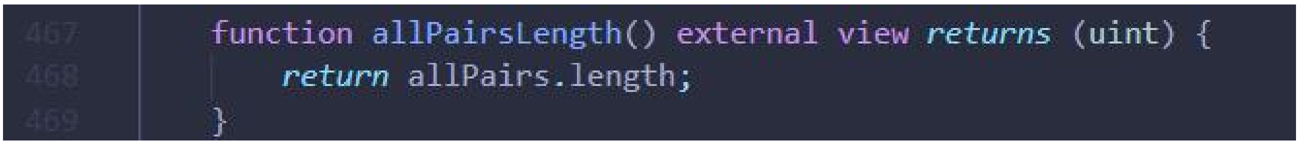
```
function setFeeToSetter(address _feeToSetter) external {
    require(msg.sender == feeToSetter, 'LavaSwap: FORBIDDEN');
    feeToSetter = _feeToSetter;
}
```

Figure 4 Source code of *setFeeToSetter*

- Related functions: *setFeeToSetter*
- Result: Pass

(4) Related query functions

- Description: The contract implements the *allPairsLength* function to query the total number of current trading pairs.



```
function allPairsLength() external view returns (uint) {
    return allPairs.length;
}
```

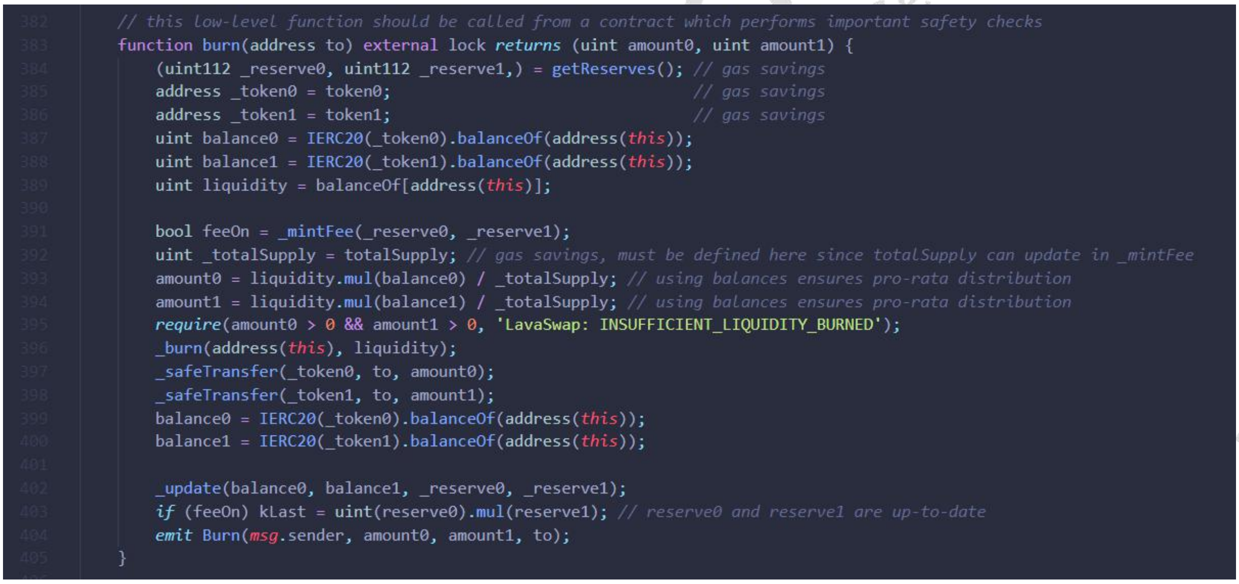
Figure 5 Source code of *allPairsLength*

- Related functions: *allPairsLength*
- Result: Pass

3.3 Business analysis of Contract LavaSwapPair

(1) burn functions

- Description: The contract implements the *burn* function for the user to destroy the corresponding number of lp tokens after removing liquidity from the specified trading pair and send the corresponding number of tokens in the specified trading pair to the user address. If feeOn removes liquidity for true, users, call _ *mintFee* in this function to calculate the handling fee for removing liquidity, and calculate the number of two tokens returned to the user in the transaction pair. If the number of two tokens returned to the user in the transaction pair is not zero, the _ *burn* function is called to destroy the corresponding amount of lp tokens removed by the user and send the two tokens returned to the user to the user address. Update the information of the two tokens in the transaction pair.



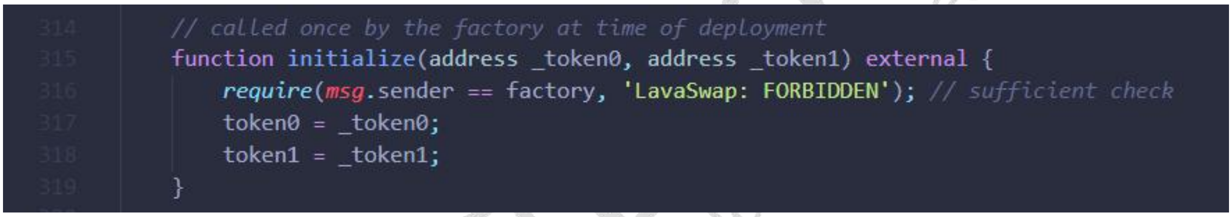
```
// this low-level function should be called from a contract which performs important safety checks
function burn(address to) external lock returns (uint amount0, uint amount1) {
    (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
    address _token0 = token0;                                // gas savings
    address _token1 = token1;                                // gas savings
    uint balance0 = IERC20(_token0).balanceOf(address(this));
    uint balance1 = IERC20(_token1).balanceOf(address(this));
    uint liquidity = balanceOf[address(this)];

    bool feeOn = _mintFee(_reserve0, _reserve1);
    uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in _mintFee
    amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribution
    amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribution
    require(amount0 > 0 && amount1 > 0, 'LavaSwap: INSUFFICIENT_LIQUIDITY_BURNED');
    _burn(address(this), liquidity);
    _safeTransfer(_token0, to, amount0);
    _safeTransfer(_token1, to, amount1);
    balance0 = IERC20(_token0).balanceOf(address(this));
    balance1 = IERC20(_token1).balanceOf(address(this));

    _update(balance0, balance1, _reserve0, _reserve1);
    if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
    emit Burn(msg.sender, amount0, amount1, to);
}
```

Figure 6 Source code of *burn*

- Related functions: *burn, getReserves*
- Result: Pass

(2) initialize functions

- Description: The contract implements the *initialize* function to initialize the pair token information of the contract, LavaSwapFactory contract only called *initialize* of pair contract once.



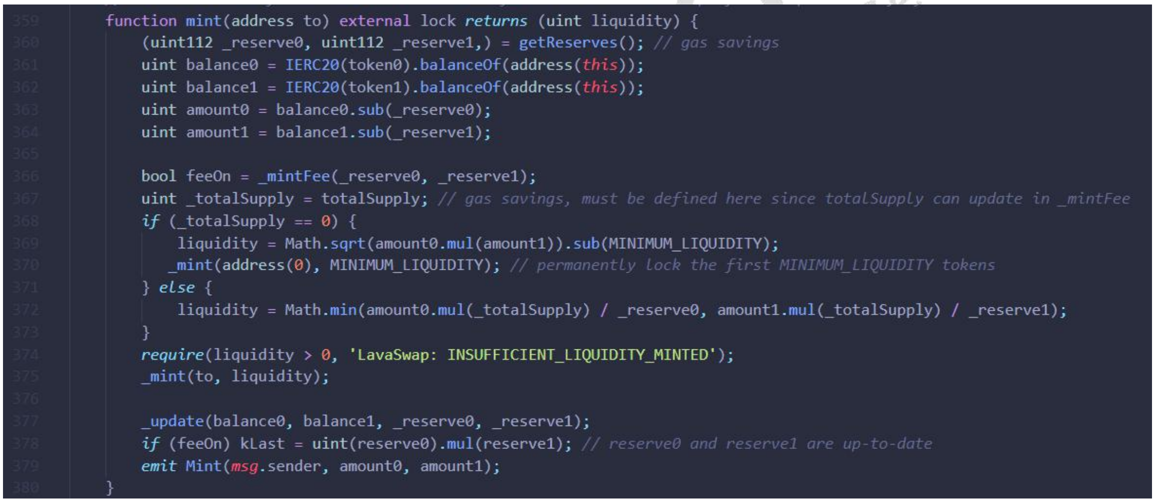
```
// called once by the factory at time of deployment
function initialize(address _token0, address _token1) external {
    require(msg.sender == factory, 'LavaSwap: FORBIDDEN'); // sufficient check
    token0 = _token0;
    token1 = _token1;
}
```

Figure 7 Source code of *initialize*

- Related functions: *initialize*
- Result: Pass

(3) mint functions

- Description: The contract implements the *mint* function for the user to add liquidity to the specified trading pair and cast the corresponding number of lp tokens to the user address. If feeOn adds liquidity for true, users, call _ *mintFee* in this function to calculate the handling fee for adding liquidity, and if the total amount of _ totalSupply of lp token is 0, then the liquidity migration will be carried out. If the total amount of lp tokens is not 0, calculate the liquidity added by the user and call the _ *mint* function to cast the corresponding number of lp tokens to the user address to update the information of the two tokens in the transaction pair.



```
function mint(address to) external lock returns (uint liquidity) {
    (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
    uint balance0 = IERC20(token0).balanceOf(address(this));
    uint balance1 = IERC20(token1).balanceOf(address(this));
    uint amount0 = balance0.sub(_reserve0);
    uint amount1 = balance1.sub(_reserve1);

    bool feeOn = _mintFee(_reserve0, _reserve1);
    uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in _mintFee
    if (_totalSupply == 0) {
        liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
       _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY tokens
    } else {
        liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _reserve1);
    }
    require(liquidity > 0, 'LavaSwap: INSUFFICIENT_LIQUIDITY_MINTED');
    _mint(to, liquidity);

    _update(balance0, balance1, _reserve0, _reserve1);
    if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
    emit Mint(msg.sender, amount0, amount1);
}
```

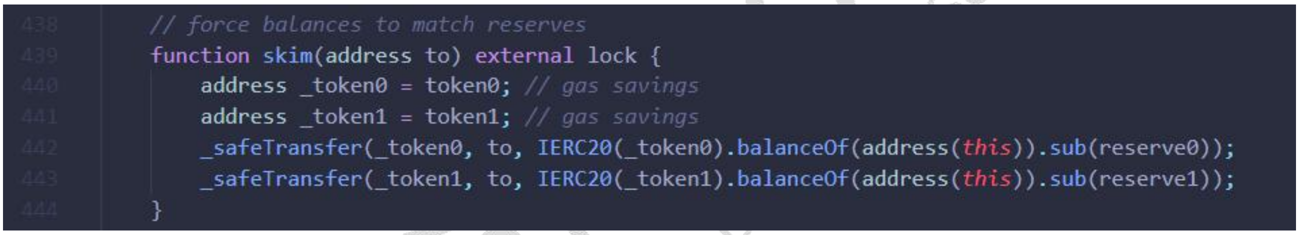
Figure 8 Source code of *mint*

- Related functions: *mint, getReserves*

- Result: Pass

(4) skim functions

- Description: The contract implements the *skim* function to limit the agreement between the actual balance of the two tokens in the contract and the number of assets in the saved constant product (the excess is sent to the caller). Any user can call this function to get additional assets (provided that there are excess assets).



```
// force balances to match reserves
function skim(address to) external lock {
    address _token0 = token0; // gas savings
    address _token1 = token1; // gas savings
    _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).sub(reserve0));
    _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
}
```

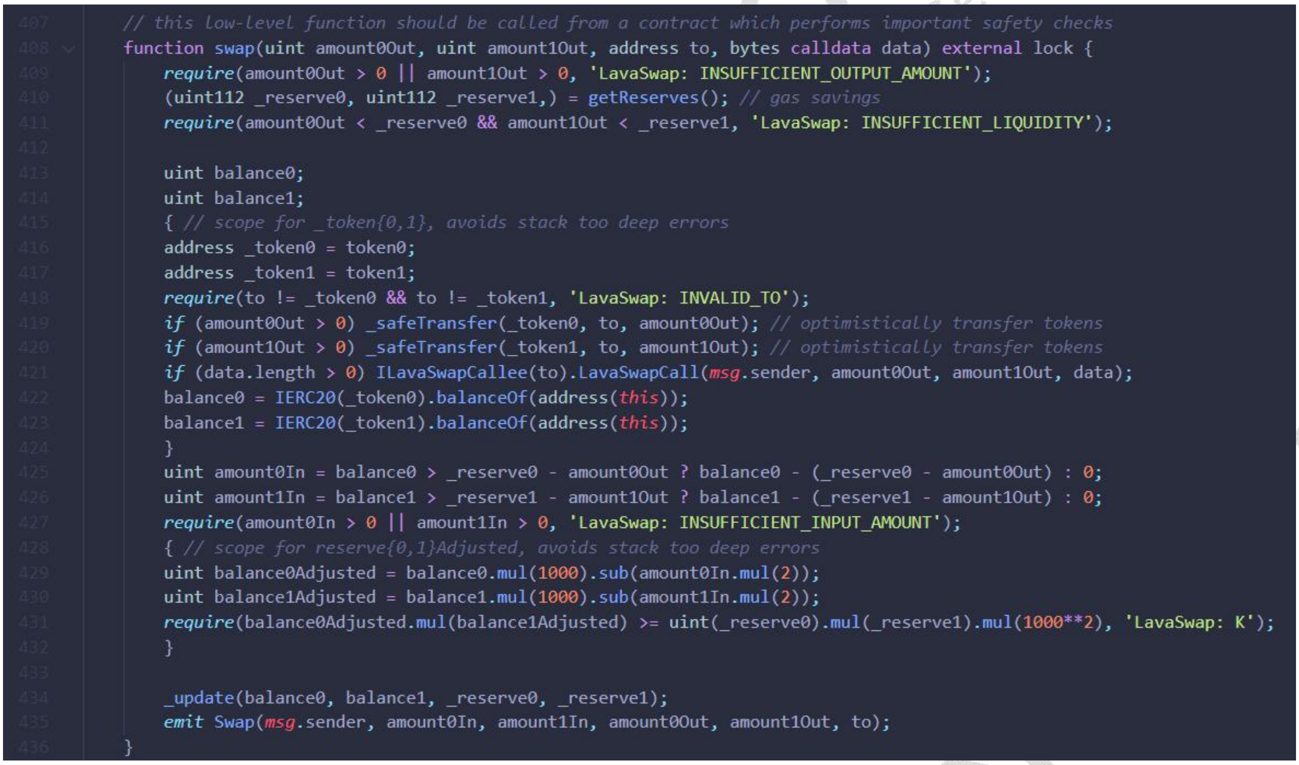
Figure 9 Source code of *skim*

- Related functions: *skim*

- Result: Pass

(5) swap functions

- Description: The contract implements the *swap* function for the user to exchange one token for another from the specified trading pair, calculates the exchange ratio of the two tokens according to the constant K value, and calls the _ *update* function to update the number of the two tokens in the transaction pair.



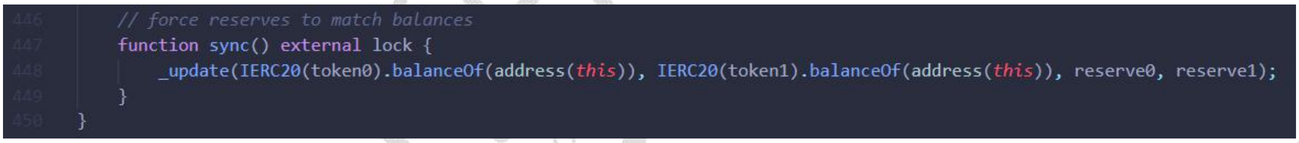
```
    // this low-level function should be called from a contract which performs important safety checks
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
        require(amount0Out > 0 || amount1Out > 0, 'LavaSwap: INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'LavaSwap: INSUFFICIENT_LIQUIDITY');

        uint balance0;
        uint balance1;
        { // scope for _token{0,1}, avoids stack too deep errors
        address _token0 = token0;
        address _token1 = token1;
        require(to != _token0 && to != _token1, 'LavaSwap: INVALID_TO');
        if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
        if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
        if (data.length > 0) ILavaSwapCallee(to).LavaSwapCall(msg.sender, amount0Out, amount1Out, data);
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));
        }
        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'LavaSwap: INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
        uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(2));
        uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(2));
        require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**2), 'LavaSwap: K');
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }
```

Figure 10 Source code of *swap*

- Related functions: *swap, getReserve*
- Result: Pass

(6) sync functions

- Description: The contract implements the *sync* function to update the actual balance and k value of the two tokens in the transaction pair and to deal with some special cases. Any user can call this function to update the actual balance of the two tokens in the transaction pair. Usually, the token balance and the k value in the transaction pair correspond to each other.



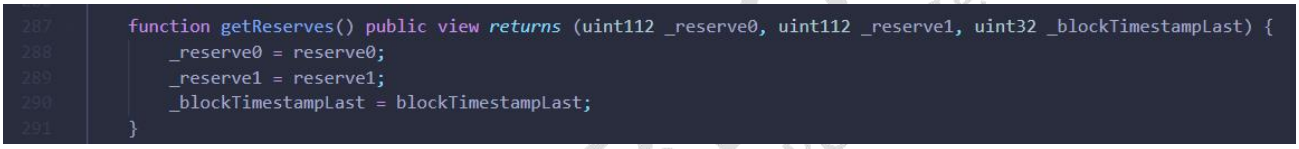
```
    // force reserves to match balances
    function sync() external lock {
        _update(IERC20(token0).balanceOf(address(this)), IERC20(token1).balanceOf(address(this)), reserve0, reserve1);
    }
}
```

Figure 11 Source code of *sync*

- Related functions: *sync*
- Result: Pass

(7) Related query functions

- Description: The contract implements the *getReserve* function to query the reserve and timestamp of the pair.



```
    function getReserves() public view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTimestampLast) {
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }
```

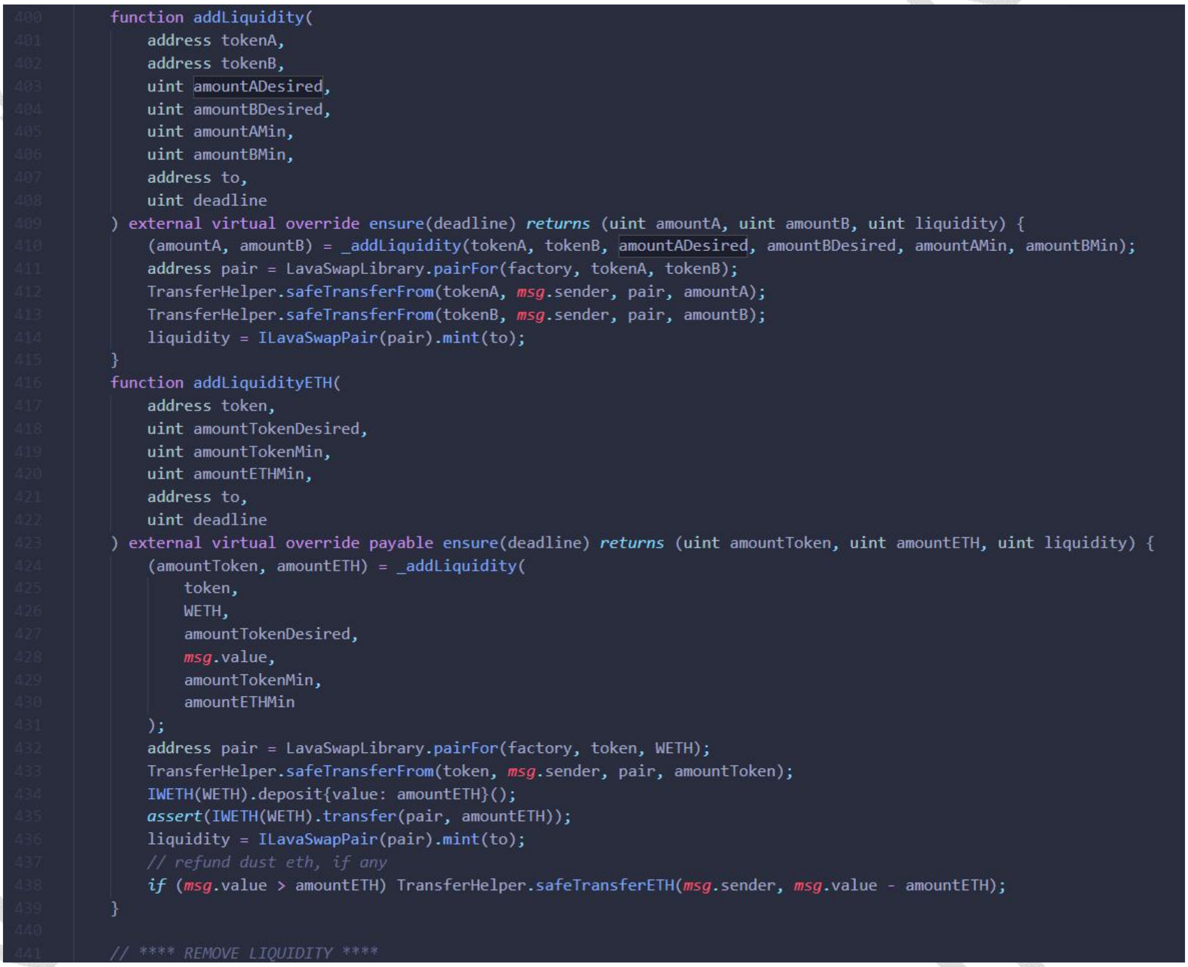
Figure 12 Source code of *getReserve*

- Related functions: *getReserve*
- Result: Pass

3.4 Business analysis of Contract LavaSwapRouter02

(1) add liquidity functions

- Description: The contract implements the *addLiquidity* function and the *addLiquidityETH* function to add liquidity. The implementation and function of the two functions are similar. Both are obtained by calling the internal function *_addLiquidity* to stake pair tokens to the pair contract and obtain LavaSwap tokens. The difference is that one of the tokens of the liquidity added in the *addLiquidityETH* function is the token of the specified WETH address.



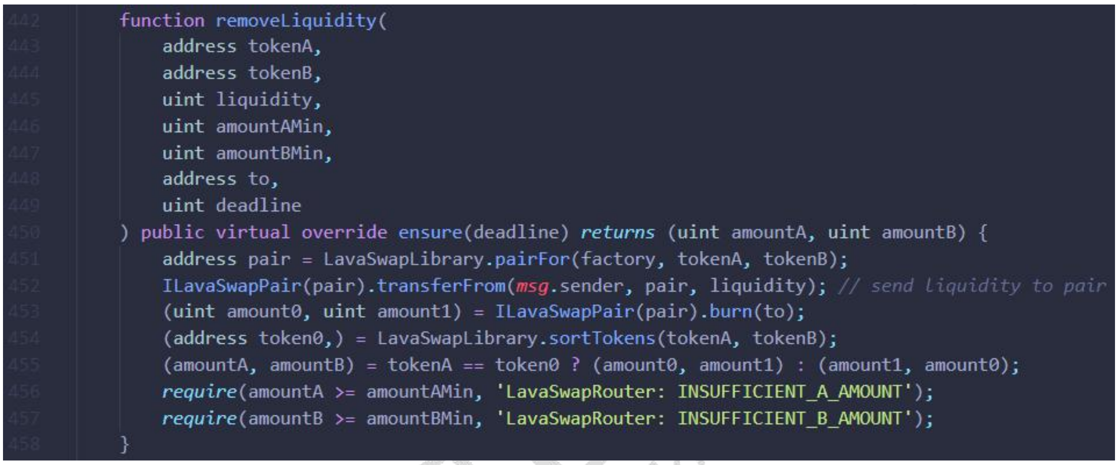
```
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired, amountAMin, amountBMin);
        address pair = LavaSwapLibrary.pairFor(factory, tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = ILavaSwapPair(pair).mint(to);
    }
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline) returns (uint amountToken, uint amountETH, uint liquidity) {
        (amountToken, amountETH) = _addLiquidity(
            token,
            WETH,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountETHMin
        );
        address pair = LavaSwapLibrary.pairFor(factory, token, WETH);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWETH(WETH).deposit{value: amountETH}();
        assert(IWETH(WETH).transfer(pair, amountETH));
        liquidity = ILavaSwapPair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountETH) TransferHelper.safeTransferETH(msg.sender, msg.value - amountETH);
    }

    // **** REMOVE LIQUIDITY ****
```

Figure 13 Source code of *addLiquidity* and *addLiquidityETH*

- Related functions: *addLiquidity, addLiquidityETH*
- Result: Pass

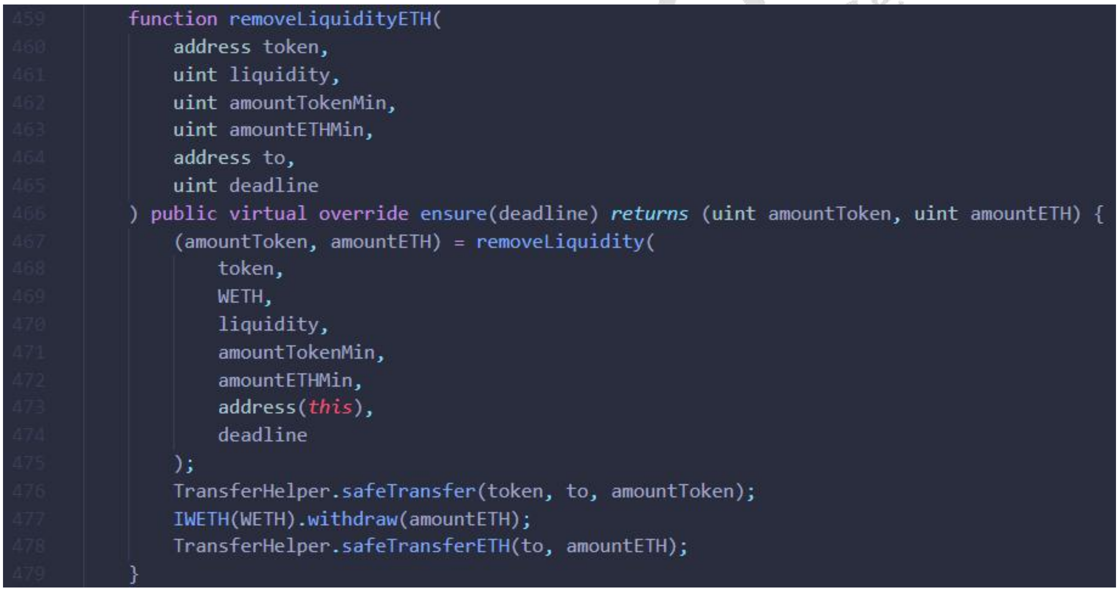
(2) remove liquidity functions

- Description: The contract implements the six functions of *removeLiquidity*, *removeLiquidityETH*, *removeLiquidityWithPermit*, *removeLiquidityETHWithPermit*, *removeLiquidityETHSupportingFeeOnTransferTokens*, *removeLiquidityETHWithPermitSupportingFeeOnTransferTokens* to remove the added liquidity. The last five functions are all implemented to remove liquidity by calling *removeLiquidity*. The difference is that *removeLiquidityETH* is the removed WETH-related liquidity, and *removeLiquidityETHSupportingFeeOnTransferTokens* is the removed WETH-related liquidity while supporting fee-on-transfer. When have a signature authorization, can remove the liquidity through the *removeLiquidityWithPermit*, *removeLiquidityETHWithPermit*, *removeLiquidityETHWithPermitSupportingFeeOnTransferTokens* function proxy.



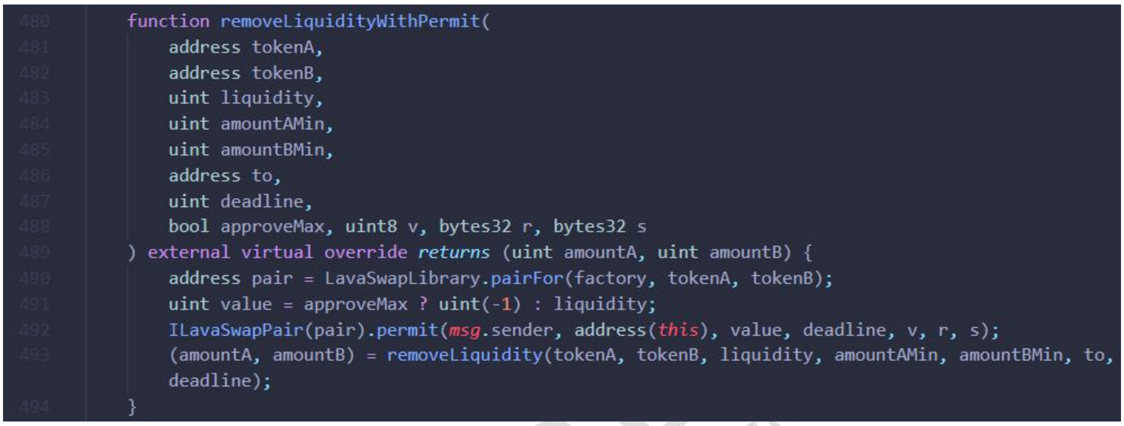
```
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountA, uint amountB) {
        address pair = LavaSwapLibrary.pairFor(factory, tokenA, tokenB);
        ILavaSwapPair(pair).transferFrom(msg.sender, pair, liquidity); // send liquidity to pair
        (uint amount0, uint amount1) = ILavaSwapPair(pair).burn(to);
        (address token0,) = LavaSwapLibrary.sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
        require(amountA >= amountAMin, 'LavaSwapRouter: INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'LavaSwapRouter: INSUFFICIENT_B_AMOUNT');
    }
```

Figure 14 Source code of *removeLiquidity*



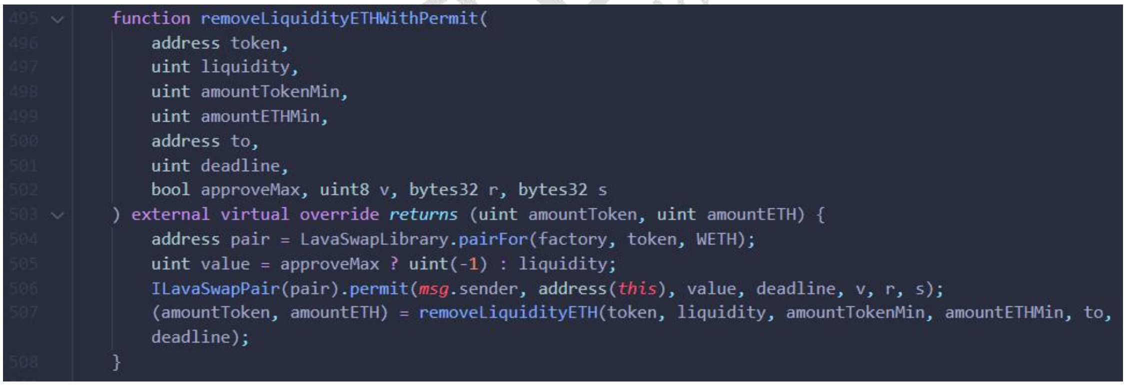
```
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountToken, uint amountETH) {
        (amountToken, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, amountToken);
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
```

Figure 15 Source code of *removeLiquidityETH*



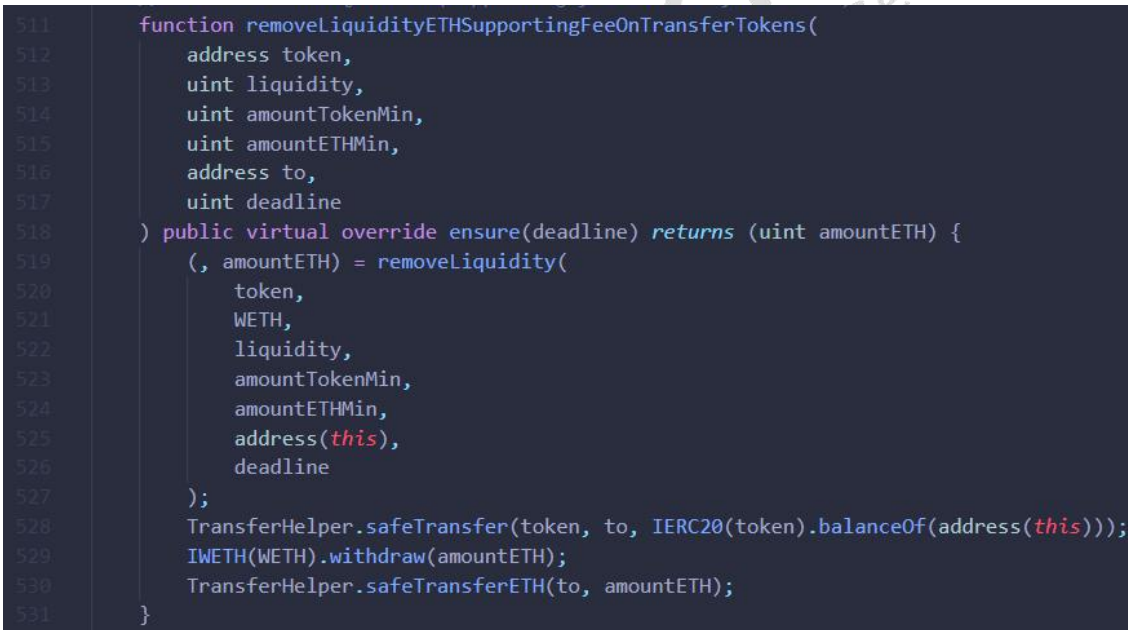
```
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountA, uint amountB) {
        address pair = LavaSwapLibrary.pairFor(factory, tokenA, tokenB);
        uint value = approveMax ? uint(-1) : liquidity;
        ILavaSwapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountA, amountB) = removeLiquidity(tokenA, tokenB, liquidity, amountAMin, amountBMin, to,
        deadline);
    }
```

Figure 16 Source code of *removeLiquidityWithPermit*



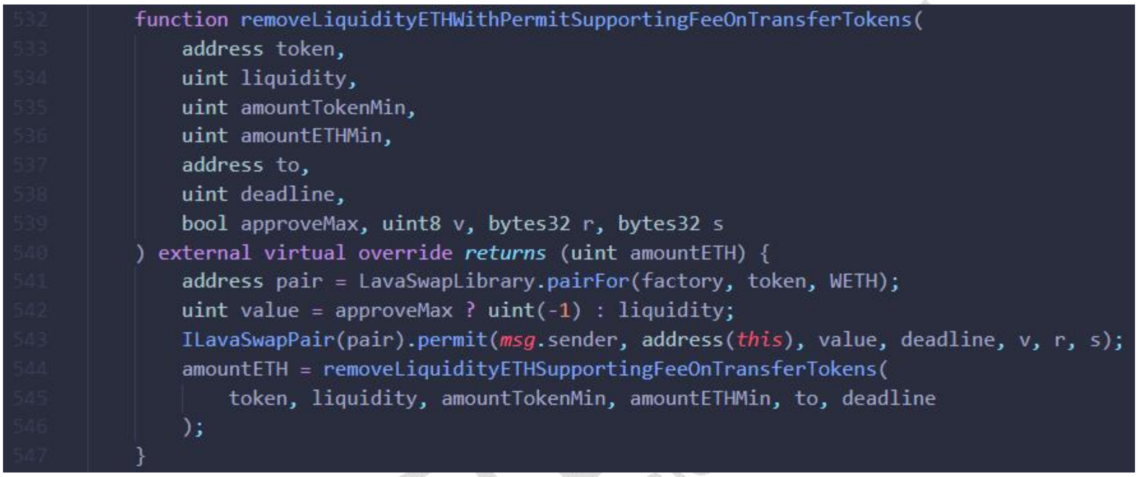
```
    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountToken, uint amountETH) {
        address pair = LavaSwapLibrary.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        ILavaSwapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountToken, amountETH) = removeLiquidityETH(token, liquidity, amountTokenMin, amountETHMin, to,
        deadline);
    }
```

Figure 17 Source code of *removeLiquidityETHWithPermit*



```
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountETH) {
        (, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, IERC20(token).balanceOf(address(this)));
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
```

Figure 18 Source code of *removeLiquidityETHSupportingFeeOnTransferTokens*



```
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountETH) {
        address pair = LavaSwapLibrary.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        ILavaSwapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        amountETH = removeLiquidityETHSupportingFeeOnTransferTokens(
            token, liquidity, amountTokenMin, amountETHMin, to, deadline
        );
    }
```

Figure 19 Source code of *removeLiquidityETHWithPermitSupportingFeeOnTransferTokens*

- Related functions: *removeLiquidity*, *removeLiquidityETH*, *removeLiquidityWithPermit*, *removeLiquidityETHWithPermit*, *removeLiquidityETHSupportingFeeOnTransferTokens*, *removeLiquidityETHWithPermitSupportingFeeOnTransferTokens, permit*

- Result: Pass

(3) swap token functions

- Description: The contract implements the token swap function through the following nine functions: *swapExactTokensForTokens*, exchange token0 with token1, enter the token for exchange and the minimum expected token value, find the path, call the internal function _*swap* to exchange along the path.

swapTokensForExactTokens, exchange token0 with token1, enter the number of tokens to obtain and the maximum value of tokens to pay, find the path, call the internal function *_swap* to exchange along the path.

swapExactETHForTokens, exchange token0 with the token1 of the WETH address, enter the WETH for exchange and the minimum expected token value, find the path, and call the internal function *_swap* to exchange along the path.

swapTokensForExactETH, exchange token0 for the WETH address, enter the expected amount of WETH and the maximum amount of tokens to pay, find the path, call the internal function *_swap* to exchange along the path.

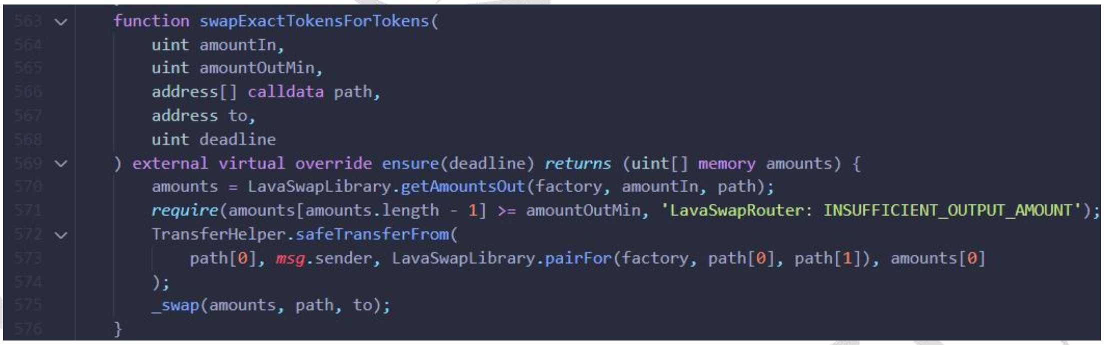
swapExactTokensForETH, exchange token0 for WETH address tokens, enter the desired minimum amount of WETH and the number of tokens paid, find the path, call the internal function *_swap* to exchange along the path.

swapETHForExactTokens, exchange token0 with tokens of WETH address, enter the expected amount of tokens and the maximum amount of WETH to pay, find the path, call the internal function *_swap* to exchange along the path.

swapExactTokensForTokensSupportingFeeOnTransferTokens, exchange token0 with token1, call the *_swapSupportingFeeOnTransferTokens* internal function, and add support for fee-on-transfer based on the *swapExactTokensForTokens* function.

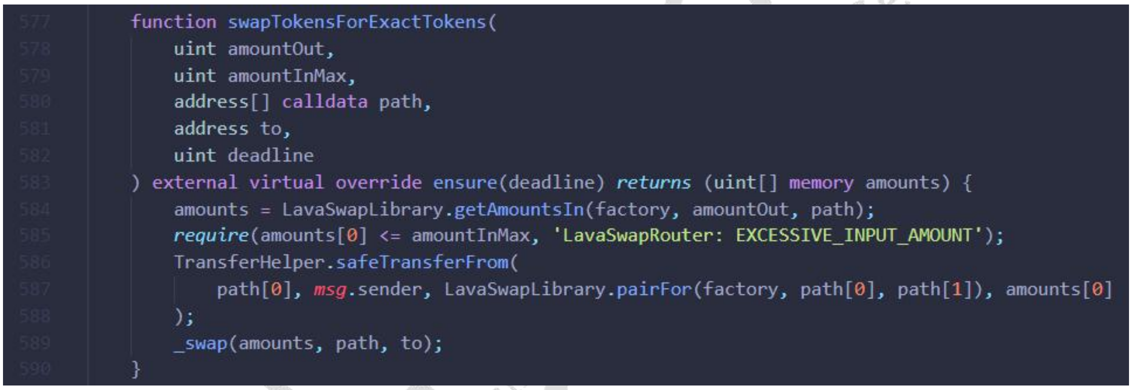
swapExactETHForTokensSupportingFeeOnTransferTokens, exchange token0 with WETH, call the *_swapSupportingFeeOnTransferTokens* internal function, and add support for fee-on-transfer based on the *swapExactETHForTokens* function.

swapExactTokensForETHSupportingFeeOnTransferTokens, exchange token0 for WETH, call the internal function *_swapSupportingFeeOnTransferTokens*, and add support for fee-on-transfer based on the *swapExactTokensForETH* function.



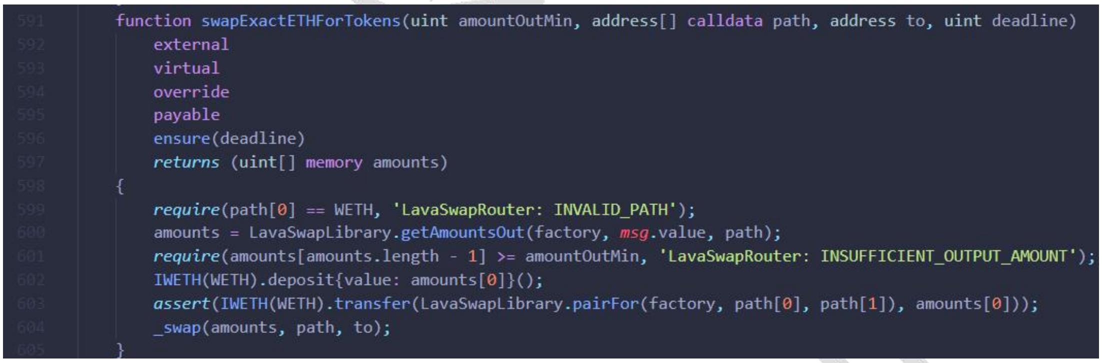
```
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = LavaSwapLibrary.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'LavaSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, LavaSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }
```

Figure 20 Source code of *swapExactTokensForTokens*



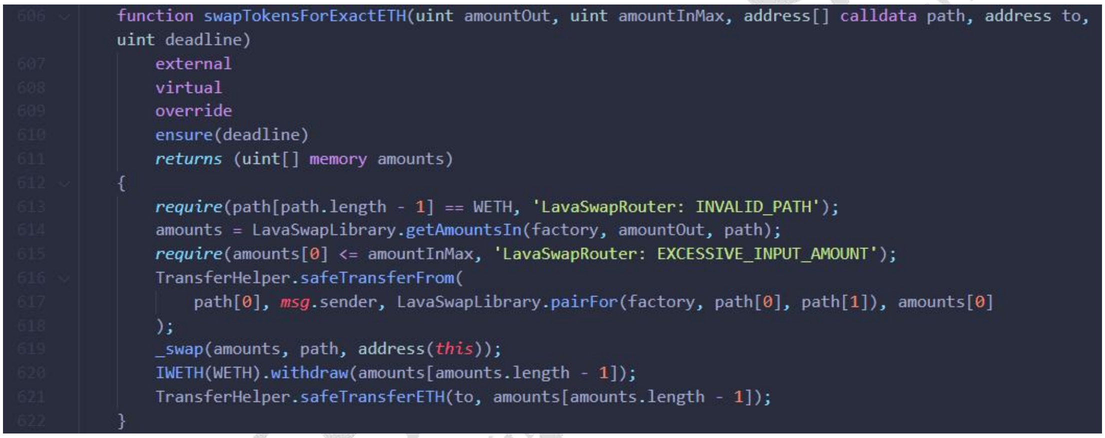
```
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = LavaSwapLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'LavaSwapRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, LavaSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }
```

Figure 21 Source code of *swapTokensForExactTokens*



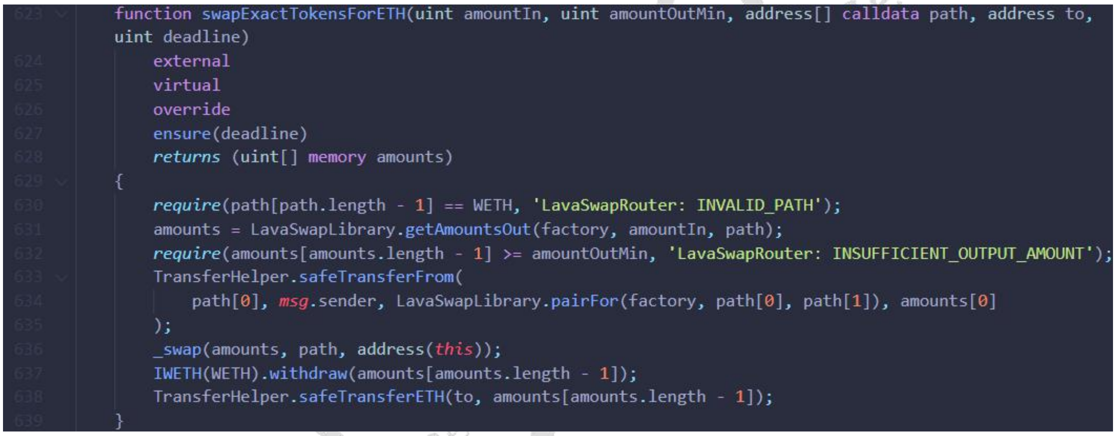
```
    function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
        external
        virtual
        override
        payable
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'LavaSwapRouter: INVALID_PATH');
        amounts = LavaSwapLibrary.getAmountsOut(factory, msg.value, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'LavaSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).deposit{value: amounts[0]}();
        assert(IWETH(WETH).transfer(LavaSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]));
        _swap(amounts, path, to);
    }
```

Figure 22 Source code of *swapExactETHForTokens*



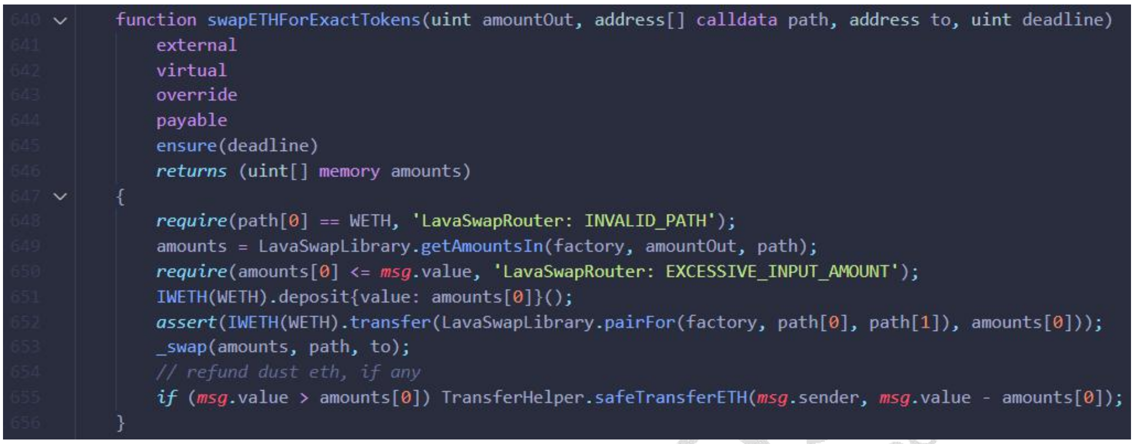
```
    function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to,
    uint deadline)
        external
        virtual
        override
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WETH, 'LavaSwapRouter: INVALID_PATH');
        amounts = LavaSwapLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'LavaSwapRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, LavaSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }
```

Figure 23 Source code of *swapTokensForExactETH*



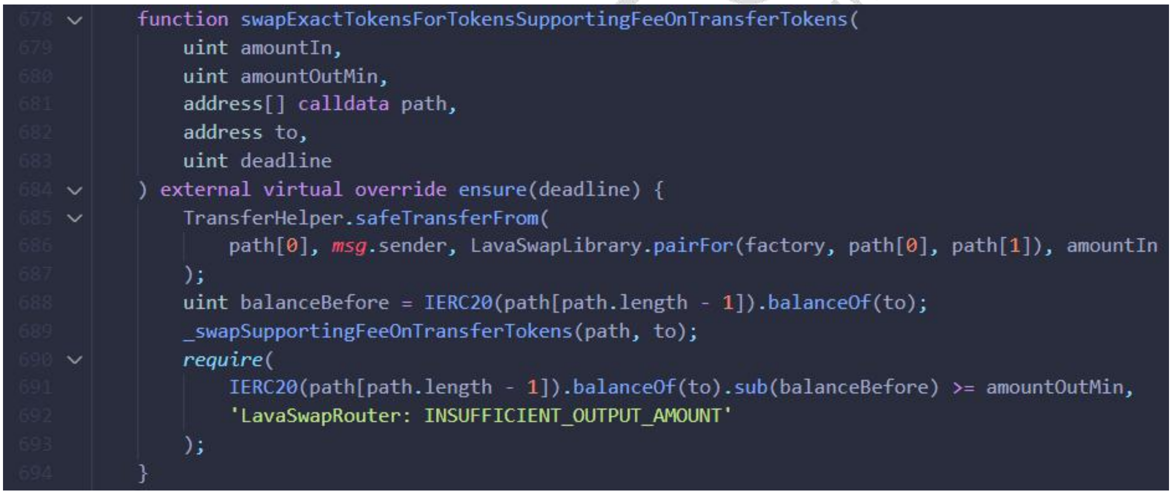
```
function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to,
uint deadline)
    external
    virtual
    override
    ensure(deadline)
    returns (uint[] memory amounts)
{
    require(path[path.length - 1] == WETH, 'LavaSwapRouter: INVALID_PATH');
    amounts = LavaSwapLibrary.getAmountsOut(factory, amountIn, path);
    require(amounts[amounts.length - 1] >= amountOutMin, 'LavaSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT');
    TransferHelper.safeTransferFrom(
        path[0], msg.sender, LavaSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
    );
    _swap(amounts, path, address(this));
    IWETH(WETH).withdraw(amounts[amounts.length - 1]);
    TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
}
```

Figure 24 Source code of *swapExactTokensForETH*



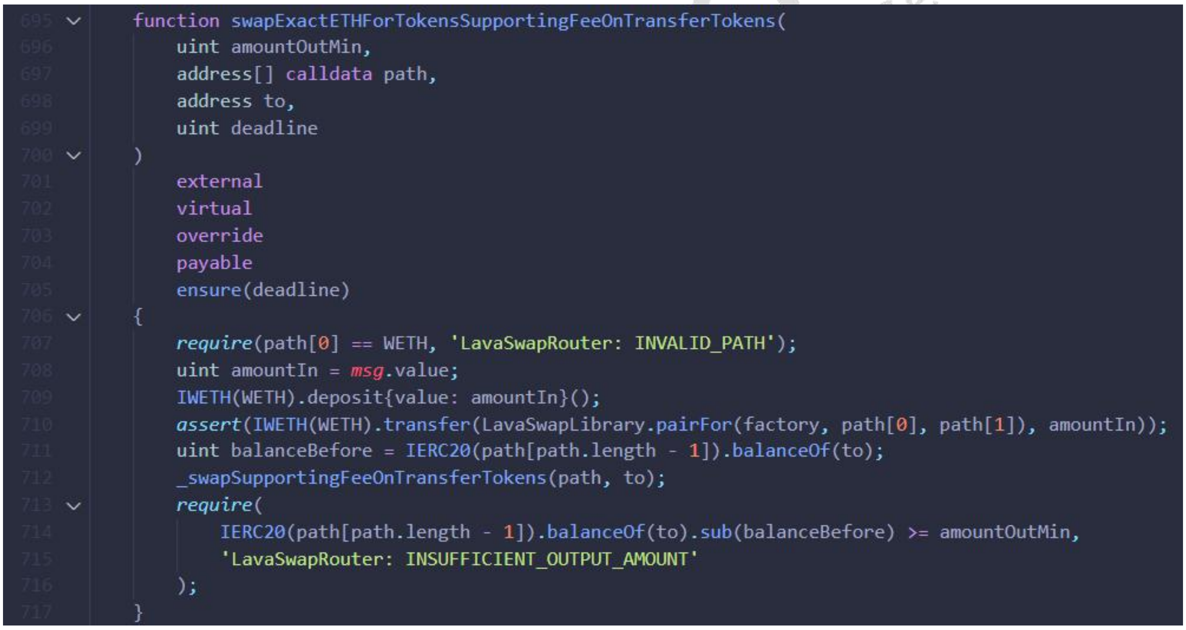
```
function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
    external
    virtual
    override
    payable
    ensure(deadline)
    returns (uint[] memory amounts)
{
    require(path[0] == WETH, 'LavaSwapRouter: INVALID_PATH');
    amounts = LavaSwapLibrary.getAmountsIn(factory, amountOut, path);
    require(amounts[0] <= msg.value, 'LavaSwapRouter: EXCESSIVE_INPUT_AMOUNT');
    IWETH(WETH).deposit{value: amounts[0]}();
    assert(IWETH(WETH).transfer(LavaSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]));
    _swap(amounts, path, to);
    // refund dust eth, if any
    if (msg.value > amounts[0]) TransferHelper.safeTransferETH(msg.sender, msg.value - amounts[0]);
}
```

Figure 25 Source code of *swapETHForExactTokens*



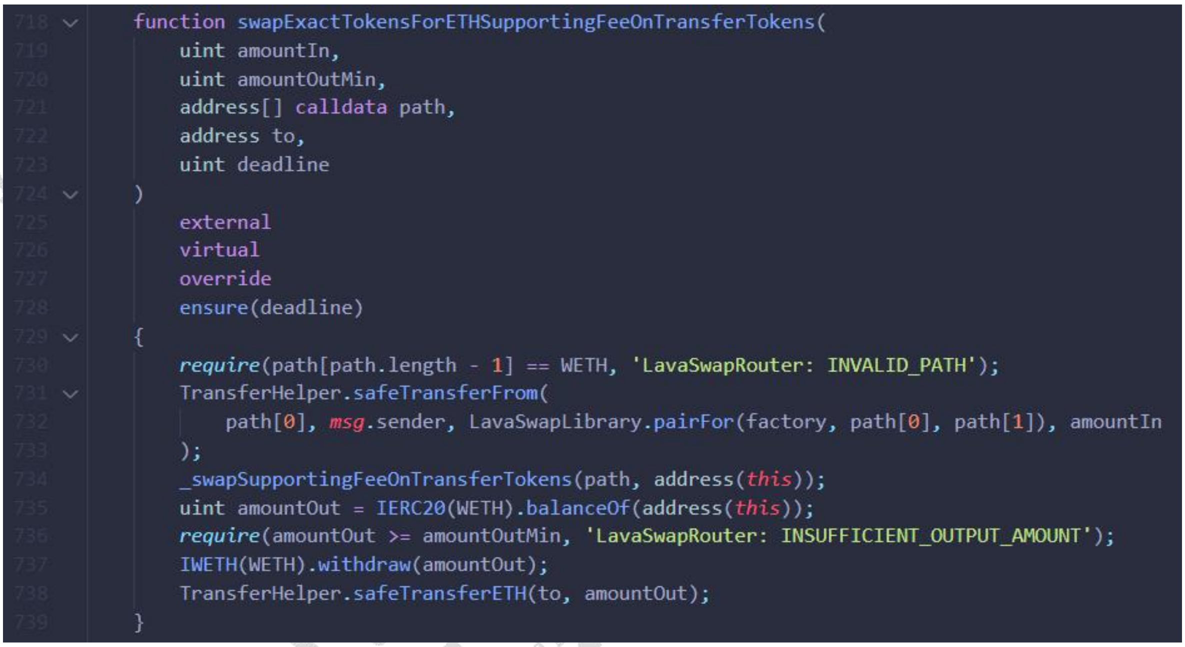
```
function swapExactTokensForTokensSupportingFeeOnTransferTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external virtual override ensure(deadline) {
    TransferHelper.safeTransferFrom(
        path[0], msg.sender, LavaSwapLibrary.pairFor(factory, path[0], path[1]), amountIn
    );
    uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
    _swapSupportingFeeOnTransferTokens(path, to);
    require(
        IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
        'LavaSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT'
    );
}
```

Figure 26 Source code of *swapExactTokensForTokensSupportingFeeOnTransferTokens*



```
function swapExactETHForTokensSupportingFeeOnTransferTokens(
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
)
    external
    virtual
    override
    payable
    ensure(deadline)
{
    require(path[0] == WETH, 'LavaSwapRouter: INVALID_PATH');
    uint amountIn = msg.value;
    IWETH(WETH).deposit{value: amountIn}();
    assert(IWETH(WETH).transfer(LavaSwapLibrary.pairFor(factory, path[0], path[1]), amountIn));
    uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
    _swapSupportingFeeOnTransferTokens(path, to);
    require(
        IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
        'LavaSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT'
    );
}
```

Figure 27 Source code of *swapExactETHForTokensSupportingFeeOnTransferTokens*



```
function swapExactTokensForETHSupportingFeeOnTransferTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
)
    external
    virtual
    override
    ensure(deadline)
{
    require(path[path.length - 1] == WETH, 'LavaSwapRouter: INVALID_PATH');
    TransferHelper.safeTransferFrom(
        path[0], msg.sender, LavaSwapLibrary.pairFor(factory, path[0], path[1]), amountIn
    );
    _swapSupportingFeeOnTransferTokens(path, address(this));
    uint amountOut = IERC20(WETH).balanceOf(address(this));
    require(amountOut >= amountOutMin, 'LavaSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT');
    IWETH(WETH).withdraw(amountOut);
    TransferHelper.safeTransferETH(to, amountOut);
}
```

Figure 28 Source code of *swapExactTokensForETHSupportingFeeOnTransferTokens*

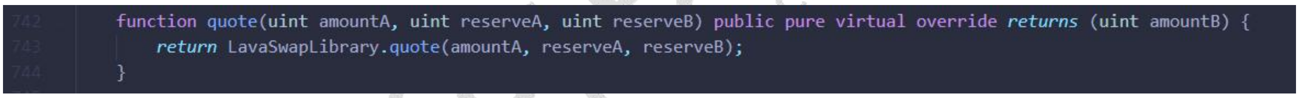
- Related functions: *swapExactTokensForTokens, swapTokensForExactTokens, swapExactETHForTokens, swapTokensForExactETH, swapExactTokensForETH, swapETHForExactTokens, swapExactTokensForTokensSupportingFeeOnTransferTokens, swapExactETHForTokensSupportingFeeOnTransferTokens, swapExactTokensForETHSupportingFeeOnTransferTokens, getReserves, getAmountOut*

- Result: Pass

(4) Related query functions

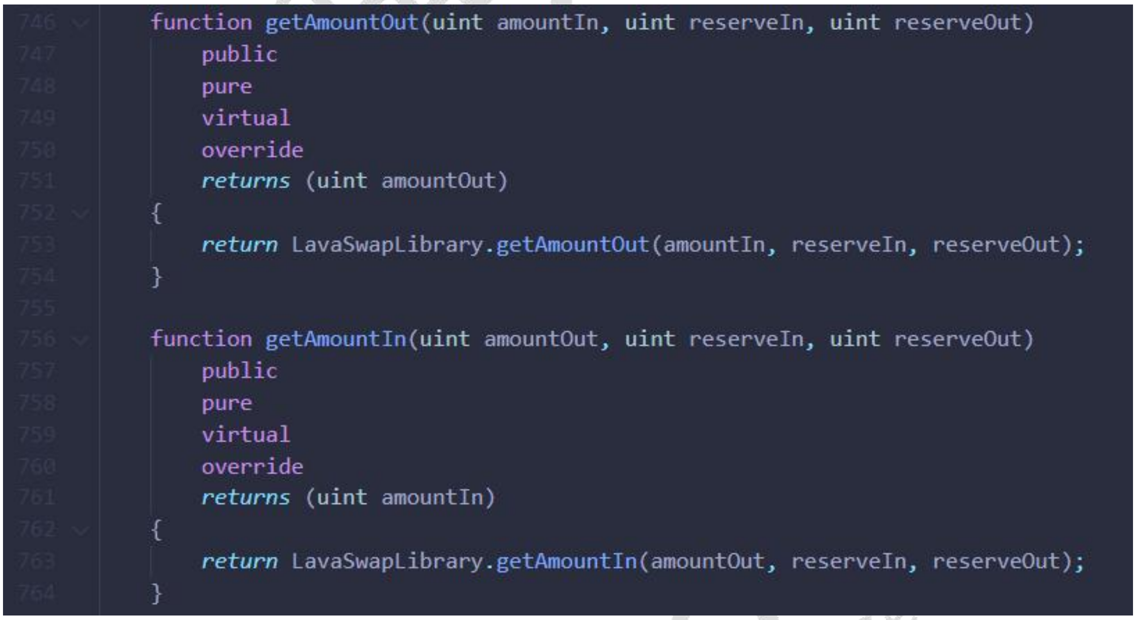
- Description: The contract implements the *quote* function to calculate the value of amountB corresponding to amountA. *getAmountOut* function to calculate the amountOut based on the amountIn.

getAmountIn function to calculate the amountIn based on the amountOut. *getAmountsOut* function to calculate the amountOut of the specified exchange path based on the amountIn. *getAmountsIn* function to calculate the amountIn of the specified exchange path based on the amountOut.



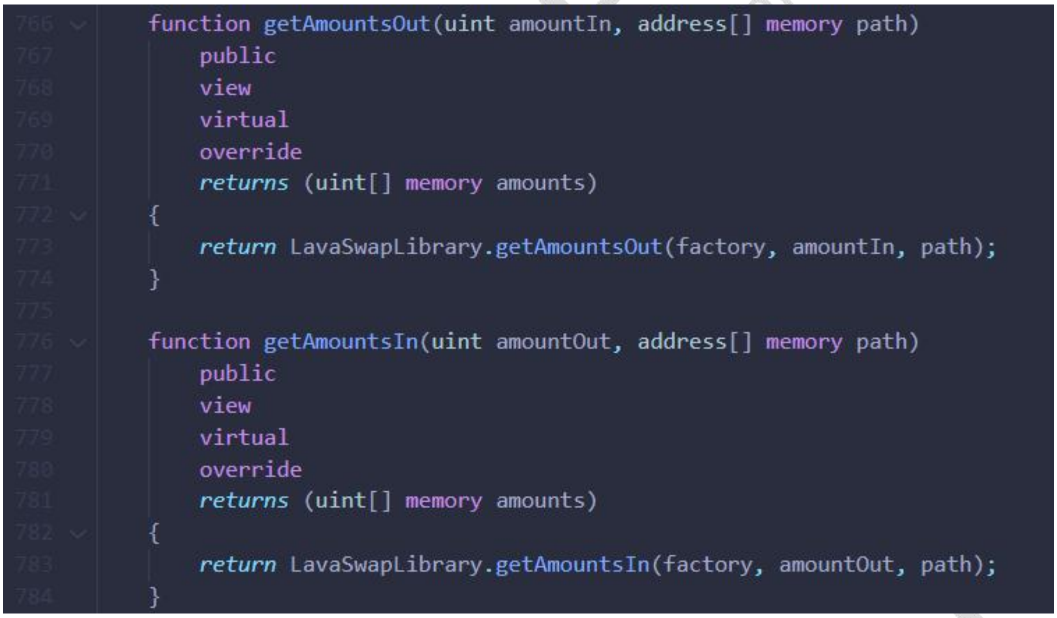
```
    function quote(uint amountA, uint reserveA, uint reserveB) public pure virtual override returns (uint amountB) {
        return LavaSwapLibrary.quote(amountA, reserveA, reserveB);
    }
```

Figure 29 Source code of *quote*



```
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut)
        public
        pure
        virtual
        override
        returns (uint amountOut)
    {
        return LavaSwapLibrary.getAmountOut(amountIn, reserveIn, reserveOut);
    }

    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut)
        public
        pure
        virtual
        override
        returns (uint amountIn)
    {
        return LavaSwapLibrary.getAmountIn(amountOut, reserveIn, reserveOut);
    }
```

Figure 30 Source code of *getAmountOut* and *getAmountIn*



```
    function getAmountsOut(uint amountIn, address[] memory path)
        public
        view
        virtual
        override
        returns (uint[] memory amounts)
    {
        return LavaSwapLibrary.getAmountsOut(factory, amountIn, path);
    }

    function getAmountsIn(uint amountOut, address[] memory path)
        public
        view
        virtual
        override
        returns (uint[] memory amounts)
    {
        return LavaSwapLibrary.getAmountsIn(factory, amountOut, path);
    }
```

Figure 31 Source code of *getAmountsOut* and *getAmountsIn*

- Related functions: *quote, getAmountOut, getAmountIn, getAmountsIn, getAmountsOut*
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts LavaSwapERC20&LavaSwapRouter02. The contracts LavaSwapERC20&LavaSwapRouter02 passed all audit items, The overall audit result is **Pass.**

BEOSIN
Blockchain Security

Official Website

https://liananantech.com

E-mail

vaas@liananantech.com

Twitter

https://twitter.com/Beosin_com