BEOSIN
Blockchain Security

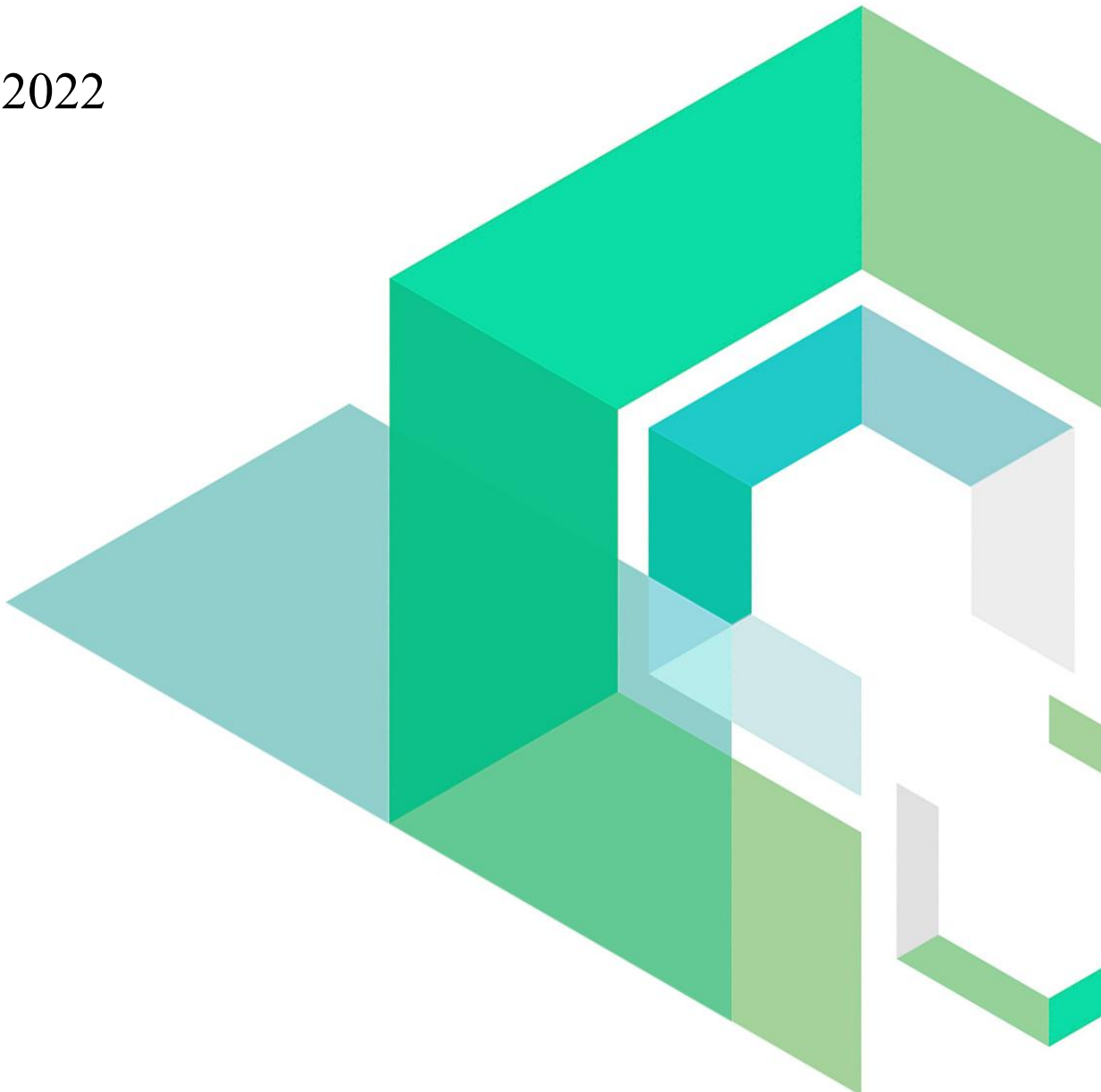
Whale Loans

Smart Contract Security Audit

V1.0

No. 202206281847

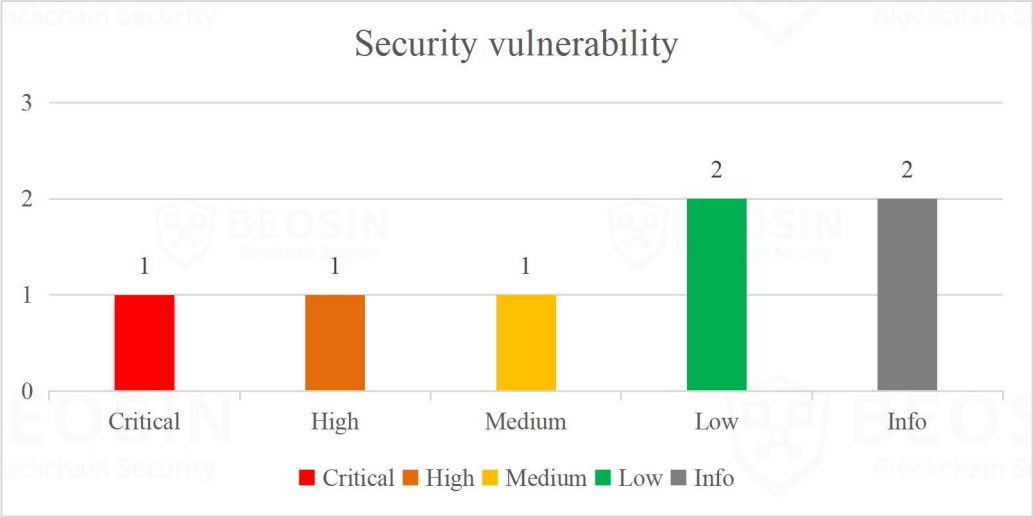
Jun 28th, 2022



Contents

Summary of audit results

After auditing, 1 Critical-risk, 1 High-risk, 1 Medium-risk, 2 Low-risk and 2 Info items were identified in the Whale Loans project. Specific audit details will be presented in the **Findings section**. Users should pay attention to the following aspects when interacting with this project：



*Notes：

- **Project Description:**

1. Business overview

The Whale Loans project provides decentralized transaction functions and flash loan functions. In the WhaleswapFactory contract, anyone can create a trading pair with two different tokens, and then users can swap tokens through the WhaleswapRouter contract, add liquidity to earn rewards and remove liquidity operations. It should be noted that the Whale Loans project provides another set of k-value algorithm, which will have a smaller slippage than uniswap when swapping. If users want to use this algorithm, they only need to set stable to true when adding trading pairs. Currently, when stable is true, the fee is 0.04%, the liquidity provider reward is 3/4, the feeto address is 1/4. When stable is false, the fee is 0.25%, the liquidity provider reward is 4/5, and the feeto address is 1/5. In the FlashmintFactory contract, anyone can use any token to create the corresponding fmToken contract. In the fmToken contract, users only need to pay a certain fee to use the flash loan function .

1 Overview

1.1 Project Overview

Project Name	Whale Loans
Platform	BNB Chain
Github	https://github.com/Whale-loans/whaleswap-core
Commit	111c6ddd32935940c1a9e38dfbe167206f28ba71(Initial) caca8b18e566add656977dfb02b0261c9b39493c(Latest)

1.2 Audit Overview

Audit work duration: Jun 23, 2022 – Jun 28, 2022

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Technology Co. Ltd.

2 Findings

Index	Risk description	Severity level	Status
Whale Swap-1	Incorrect k value judgment	Critical	Fixed
Whale Flash-2	Fees will be locked in the contract	High	Fixed
Whale Flash-3	Permission usage error	Medium	Fixed
Whale Swap-4	The *_swapSupportingFeeOnTransferTokens* function design flaw	Low	Fixed
Whale Swap-5	May fail to swap	Low	Acknowledged
Whale-6	Lack event trigger	Info	Fixed
Whale-7	Redundant code	Info	Fixed

Risk Details Description:

1. Whale Swap-5 is not Fixed and will cause users to add inappropriate trading pairs and cannot trade.

[Whale Swap-1] Incorrect k value judgment

Severity Level	**Critical**
Type	Business Security
Lines	WhaleswapPair.sol#L290
Description	In the *swap* function of the WhaleswapPair contract, when making the k-value judgment, the k-value check will pass after borrowing a large amount of funds from the pair contract, as balance0Adjusted and balance1Adjusted have undergone precision expansion before calculating the k-value, while _reserve0 and _reserve1 have not undergone precision expansion before calculating the k-value.

```
277         uint balance1Adjusted;
278
279         // Dynamic fees
280         if(stable){
281             balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(4)); // 4 == 0.04% fee
282             balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(4));
283         }
284         else{
285             balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(25)); // 25 == 0.25% fee
286             balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(25));
287         }
288
289         // The curve, either x3y+y3x for stable pools, or x*y for volatile pools
290         require(_k(balance0Adjusted, balance1Adjusted) >= _k(_reserve0, _reserve1).mul(10000**2), 'Whaleswap: K');
291     }
292
293     _update(balance0, balance1, _reserve0, _reserve1);
294     emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
295 }
```

Figure 1 Source code of *swap* function (Unfixed)

Recommendations	It is recommended that _reserve0 and _reserve1 perform precision expansion before calculating the k value.
Status	Fixed.

```
274     require(amount0In > 0 || amount1In > 0, 'Whaleswap: INSUFFICIENT_INPUT_AMOUNT');
275     { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
276         uint balance0Adjusted;
277         uint balance1Adjusted;
278
279         // Dynamic fees
280         if(stable){
281             balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(4)); // 4 == 0.04% fee
282             balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(4));
283         }
284         else{
285             balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(25)); // 25 == 0.25% fee
286             balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(25));
287         }
288
289         // The curve, either x3y+y3x for stable pools, or x*y for volatile pools
290         require(_k(balance0Adjusted, balance1Adjusted) >= _k(_reserve0 * 10000, _reserve1 * 10000), 'Whaleswap: K');
291     }
292
293     _update(balance0, balance1, _reserve0, _reserve1);
294     emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
295 }
296
```

Figure 2 Source code of *swap* function (Fixed)

[Whale Flash-2] Fees will be locked in the contract

Severity Level	**High**
Type	Business Security
Lines	FlashMain.sol&FlashERC20.sol#L104-106, 85-114
Description	In the FlashMain and FlashERC20 contracts, the handling fee of the flash loan is sent to the contract, because the contract has no withdrawal function, which will cause this part of the handling fee to be locked in the contract.

```
83    // Allows anyone to mint unbacked flash-underlying as long as it gets burned by the end of the transaction.
84    function flashMint(uint256 amount) external nonReentrant {
85        require(amount < (type(uint256).max - totalSupply()));
86
87        // calculate fee
88        uint256 fee = FlashmintFactory(factory).fee();
89        uint256 actualFee = amount.mul(fee).div(oneEth);
90
91        // mint tokens
92        _mint(msg.sender, amount);
93
94        // hand control to borrower
95        IBorrower(msg.sender).executeOnFlashMint(amount, actualFee);
96
97        // burn tokens
98        _burn(msg.sender, amount); // reverts if `msg.sender` does not have enough units of the FMT
99
100       // double-check that all fERC20 is backed by the underlying
101       assert(underlying.balanceOf(address(this)) >= totalSupply());
102
103       // send the fee
104       if (fee != 0) {
105           underlying.safeTransferFrom(msg.sender, address(this), actualFee);
106       }
107
108       emit FlashMint(msg.sender, amount);
109   }
110 }
111
```

Figure 3 Source code of *flashMint* function (Unfixed)

```
85    function flashMint(uint256 amount) external nonReentrant {
86        require(amount < (type(uint256).max - totalSupply()));
87
88        // calculate fee
89        uint256 fee = FlashmintFactory(factory).fee();
90        _borrowerDebt = amount.mul(fee).div(oneEth);
91
92        // mint tokens
93        _mint(msg.sender, amount);
94
95        // hand control to borrower
96        IBorrower(msg.sender).executeOnFlashMint(amount, _borrowerDebt);
97
98        // burn tokens
99        _burn(msg.sender, amount); // reverts if `msg.sender` does not have enough fWBNB
100
101       // double-check that all fWBNB is backed by BNB
102       assert(address(this).balance >= totalSupply());
103
104       // check that fee has been paid
105       require(_borrowerDebt == 0, "Fee not paid");
106
107       emit FlashMint(msg.sender, amount);
108   }
109
110   // @notice Repay all or part of the loan
111   function repayEthDebt() external payable {
112       _borrowerDebt = _borrowerDebt.sub(msg.value); // does not allow overpayment
113   }
114 }
```

Figure 4 Source code of *flashMint* function (Unfixed)

Recommendations	It is recommended to send the handling fee to another address.
Status	Fixed.

```
77      // @audit The `nonReentrant` modifier is critical here.
78      function flashMint(uint256 amount) external nonReentrant {
79          require(amount < (type(uint256).max - totalSupply()));
80
81          // calculate fee
82          uint256 fee = FlashmintFactory(factory).fee();
83          _borrowerDebt = amount.mul(fee).div(oneEth);
84
85          // mint tokens
86          _mint(msg.sender, amount);
87
88          // hand control to borrower
89          IBorrower(msg.sender).executeOnFlashMint(amount, _borrowerDebt);
90
91          // burn tokens
92          _burn(msg.sender, amount); // reverts if `msg.sender` does not have enough fWBNB
93
94          // double-check that all fWBNB is backed by BNB
95          assert(address(this).balance >= totalSupply());
96
97          // send the fee
98          if (fee != 0) {
99              payable(FlashmintFactory(factory).feeTo()).transfer(_borrowerDebt);
100         }
101
102         // check that fee has been paid
103         require(_borrowerDebt == 0, "Fee not paid");
104
105         emit FlashMint(msg.sender, amount);
106     }
107
108     // @notice Repay all or part of the loan
109     function repayEthDebt() external payable {
110         _borrowerDebt = _borrowerDebt.sub(msg.value); // does not allow overpayment
111     }
112 }
```

Figure 5 Source code of *flashMint* function (Fixed)

```
76      // Allows anyone to mint unbacked flash-underlying as long as it gets burned by the end of the transaction.
77      function flashMint(uint256 amount) external nonReentrant {
78          require(amount < (type(uint256).max - totalSupply()));
79
80          // calculate fee
81          uint256 fee = FlashmintFactory(factory).fee();
82          uint256 actualFee = amount.mul(fee).div(oneEth);
83
84          // mint tokens
85          _mint(msg.sender, amount);
86
87          // hand control to borrower
88          IBorrower(msg.sender).executeOnFlashMint(amount, actualFee);
89
90          // burn tokens
91          _burn(msg.sender, amount); // reverts if `msg.sender` does not have enough units of the FMT
92
93          // double-check that all fERC20 is backed by the underlying
94          assert(underlying.balanceOf(address(this)) >= totalSupply());
95
96          // send the fee
97          if (fee != 0) {
98              underlying.safeTransferFrom(msg.sender, FlashmintFactory(factory).feeTo(), actualFee);
99          }
100
101         emit FlashMint(msg.sender, amount);
102     }
103 }
104
```

Figure 6 Source code of *flashMint* function (Fixed)

[Whale Flash-3] Permission usage error

Severity Level	Medium
Type	Business Security
Lines	FlashmintFactory.sol&FlashMain.sol&FlashERC20.sol#L29-32,55,64,58,63
Description	When creating fmtoken through the *createFlashMintableToken* function of the FlashmintFactory contract, the owner of the fmtoken contract will be the FlashmintFactory contract, which will make the *setDepositLimit* and *setWhitelist* functions in the fmtoken contract unavailable.

```
21
22  function createFlashMintableToken(address baseToken) external returns (address fmToken) {
23      require(getFmToken[baseToken] == address(0), 'Flashmint: TOKEN_EXISTS'); // No need to check other way around
24      require(getBaseToken[baseToken] == address(0), 'Flashmint: NESTED_TOKENS'); // Can't create a fmt of another fmt
25
26      // Create the token
27      bytes memory bytecode = getCreationBytecode(baseToken);
28      bytes32 salt = keccak256(abi.encodePacked(baseToken));
29      assembly {
30          fmToken := create2(0, add(bytecode, 32), mload(bytecode), salt)
31      }
32
33      // Store the token details
34      getBaseToken[fmToken] = baseToken;
35      getFmToken[baseToken] = fmToken;
36      allTokens.push(fmToken);
37      emit TokenCreated(baseToken, fmToken, allTokens.length);
38  }
39
```

Figure 7 Source code of *createFlashMintableToken* function

```
54
55  function setWhitelist(address addr, bool status) external onlyOwner {
56      whitelistAddr[addr] = status;
57      emit WhitelistUpdate(addr, status);
58  }
59
60  receive() external payable {
61      deposit();
62  }
63
64  function setDepositLimit(uint256 value) public onlyOwner {
65      _depositLimit = value;
66      emit NewDepositLimit(_depositLimit);
67  }
68
```

Figure 8 Source code of *setWhitelist&setDepositLimit* functions (Unfixed)(FlashMain.sol)

```
57
58  function setWhitelist(address addr, bool status) external onlyOwner {
59      whitelistAddr[addr] = status;
60      emit WhitelistUpdate(addr, status);
61  }
62
63  function setDepositLimit(uint256 value) public onlyOwner {
64      _depositLimit = value;
65      emit NewDepositLimit(_depositLimit);
66  }
67
```

Figure 9 Source code of *setWhitelist&setDepositLimit* functions (Unfixed)(FlashERC20.sol)

Recommendations	It is recommended to use other addresses for permission verification.

Status Fixed.

```
54
55  function setDepositLimit(uint256 value) public {
56      require(msg.sender == factory.feeSetter(), "Only flashmint factory fee setter can update deposit limit");
57      _depositLimit = value;
58      emit NewDepositLimit(_depositLimit);
59  }
60
```

Figure 10 Source code of *setDepositLimit* function (Fixed)(FlashERC20.sol)

```
55
56  function setDepositLimit(uint256 value) public {
57      require(msg.sender == factory.feeSetter(), "Only flashmint factory fee setter can update deposit limit");
58      _depositLimit = value;
59      emit NewDepositLimit(_depositLimit);
60  }
61
```

Figure 11 Source code of *setDepositLimit* function (Fixed)(FlashMain.sol)

[Whale Swap-4] The *_swapSupportingFeeOnTransferTokens* function design flaw

Severity Level	Low
Type	Business Security
Lines	WhaleswapRouter.sol#321-336,440-455
Description	In the *_swapSupportingFeeOnTransferTokens* function, the pair obtained is the pair input by the user, but in the *getAmountOut* function, the optimal pair is selected and the corresponding exchangeable quantity is returned, but the *_swapSupportingFeeOnTransferTokens* function does not reflect the optimal pair for switching. If the pair with the optimal exchange quantity is not equal to the pair input by the user, the user swap will fail as this number belongs to the optimal pair, not the pair entered by the user.

```
321  function _swapSupportingFeeOnTransferTokens(route[] memory routes, address _to) internal virtual {
322      for (uint i; i < routes.length; i++) {
323          (address token0, address token1) = sortTokens(routes[i].from, routes[i].to);
324          WhaleswapPair pair = WhaleswapPair(pairFor(routes[i].from, routes[i].to, routes[i].stable));
325
326          uint amountOutput;
327          { // scope to avoid stack too deep errors
328          (uint reserve0, uint reserve1,) = pair.getReserves();
329          (uint reserveInput,) = routes[i].from == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
330          uint amountInput = IERC20(routes[i].from).balanceOf(address(pair)).sub(reserveInput);
331          (amountOutput,) = getAmountOut(amountInput, token0, token1);
332          }
333          (uint amount0Out, uint amount1Out) = routes[i].from == token0 ? (uint(0), amountOutput) : (amountOutput, uint(0));
334          address to = i < routes.length - 1 ? pairFor(routes[i+1].from, routes[i+1].to, routes[i+1].stable) : _to;
335
336          pair.swap(amount0Out, amount1Out, to, new bytes(0));
337      }
338  }
```

Figure 12 Source code of *_swapSupportingFeeOnTransferTokens* function (Unfixed)

```
440  // given an input amount of an asset and pair reserves, returns the maximum output amount of the other asset
441  function getAmountOut(uint amountIn, address tokenIn, address tokenOut) public view returns (uint amountOut, bool stable) {
442      require(amountIn > 0, 'WhaleswapRouter: INSUFFICIENT_INPUT_AMOUNT');
443      require(tokenIn != address(0) && tokenOut != address(0), 'WhaleswapRouter: INSUFFICIENT_LIQUIDITY');
444
445      address pair = pairFor(tokenIn, tokenOut, true);
446      uint amountStable;
447      uint amountVolatile;
448      if (WhaleswapFactory(factory).isPair(pair)) {
449          amountStable = WhaleswapPair(pair).getAmountOut(amountIn, tokenIn);
450      }
451      pair = pairFor(tokenIn, tokenOut, false);
452      if (WhaleswapFactory(factory).isPair(pair)) {
453          amountVolatile = WhaleswapPair(pair).getAmountOut(amountIn, tokenIn);
454      }
455      return amountStable > amountVolatile ? (amountStable, true) : (amountVolatile, false);
456  }
```

Figure 13 Source code of *getAmountOut* function

Recommendations	It is recommended to judge the amount of the swap as to which pair, and use the corresponding pair to swap.
Status	Fixed.

```
// requires the initial amount to have already been sent to the first pair
function _swapSupportingFeeOnTransferTokens(route[] memory routes, address _to) internal virtual {
    for (uint i; i < routes.length; i++) {
        (address token0, address token1) = sortTokens(routes[i].from, routes[i].to);
        WhaleswapPair pair = WhaleswapPair(pairFor(routes[i].from, routes[i].to, routes[i].stable));

        uint amountOutput;
        { // scope to avoid stack too deep errors
        (uint reserve0, uint reserve1,) = pair.getReserves();
        (uint reserveInput,) = routes[i].from == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
        uint amountInput = IERC20(routes[i].from).balanceOf(address(pair)).sub(reserveInput);
        amountOutput = getAmountOut(amountInput, token0, token1, routes[i].stable);
        }
        (uint amount0Out, uint amount1Out) = routes[i].from == token0 ? (uint(0), amountOutput) : (amountOutput, uint(0));
        address to = i < routes.length - 1 ? pairFor(routes[i+1].from, routes[i+1].to, routes[i+1].stable) : _to;

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

Figure 14 Source code of *_swapSupportingFeeOnTransferTokens* function (Fixed)

```
// given an input amount of an asset, pair reserves and the pool type, returns the maximum output amount of the other asset
function getAmountOut(uint amountIn, address tokenIn, address tokenOut, bool isStable) public view returns (uint amountOut) {
    require(amountIn > 0, 'WhaleswapRouter: INSUFFICIENT_INPUT_AMOUNT');
    require(tokenIn != address(0) && tokenOut != address(0), 'WhaleswapRouter: INSUFFICIENT_LIQUIDITY');

    return WhaleswapPair(pairFor(tokenIn, tokenOut, isStable)).getAmountOut(amountIn, tokenIn);
}
```

Figure 15 Source code of *getAmountOut* function (Fixed)

[Whale Swap-5] May fail to swap

Severity Level	Low
Type	Business Security
Lines	WhaleswapPair.sol#290,84-126
Description	In the WhaleswapPair contract, when _stable is true, since the calculation of the k value is performed with precision expansion, if there is a large amount of tokens, the calculation will overflow and the user cannot swap.

```
255     // this low-level function should be called from a contract which performs important safety checks
256     function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
257         require(amount0Out > 0 || amount1Out > 0, 'Whaleswap: INSUFFICIENT_OUTPUT_AMOUNT');
258         (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
259         require(amount0Out < _reserve0 && amount1Out < _reserve1, 'Whaleswap: INSUFFICIENT_LIQUIDITY');
260
261         uint balance0;
262         uint balance1;
263         { // scope for _token{0,1}, avoids stack too deep errors
264             (address _token0, address _token1) = (token0, token1);
265             require(to != _token0 && to != _token1, 'Whaleswap: INVALID_TO');
266             if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
267             if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
268             if (data.length > 0) IWhaleswapCallee(to).whaleswapCall(msg.sender, amount0Out, amount1Out, data);
269             balance0 = IERC20(_token0).balanceOf(address(this));
270             balance1 = IERC20(_token1).balanceOf(address(this));
271         }
272         uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
273         uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
274         require(amount0In > 0 || amount1In > 0, 'Whaleswap: INSUFFICIENT_INPUT_AMOUNT');
275         { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
276             uint balance0Adjusted;
277             uint balance1Adjusted;
278
279             // Dynamic fees
280             if(stable){
281                 balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(4)); // 4 == 0.04% fee
282                 balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(4));
283             }
284             else{
285                 balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(25)); // 25 == 0.25% fee
286                 balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(25));
287             }
288
289             // The curve, either x3y+y3x for stable pools, or x*y for volatile pools
290             require(_k(balance0Adjusted, balance1Adjusted) >= _k(_reserve0 * 10000, _reserve1 * 10000), 'Whaleswap: K');
291         }
292
293         _update(balance0, balance1, _reserve0, _reserve1);
294         emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
295     }
```

Figure 16 Source code of *swap* function

```
83
84      function _k(uint x, uint y) internal view returns (uint) {
85          if (stable) {
86              uint _x = x * 1e18 / decimals0;
87              uint _y = y * 1e18 / decimals1;
88              uint _a = (_x * _y) / 1e18;
89              uint _b = ((_x * _x) / 1e18 + (_y * _y) / 1e18);
90              return _a * _b / 1e18;  // x3y + y3x >= k
91          } else {
92              return x * y; // xy >= k
93          }
94      }
95
96      function _f(uint x0, uint y) internal pure returns (uint) {
97          return x0*(y*y/1e18*y/1e18)/1e18+(x0*x0/1e18*x0/1e18)*y/1e18;
98      }
99
100     function _d(uint x0, uint y) internal pure returns (uint) {
101         return 3*x0*(y*y/1e18)/1e18+(x0*x0/1e18*x0/1e18);
102     }
103
104     function _get_y(uint x0, uint xy, uint y) internal pure returns (uint) {
105         for (uint i = 0; i < 255; i++) {
106             uint y_prev = y;
107             uint k = _f(x0, y);
108             if (k < xy) {
109                 uint dy = (xy - k)*1e18/_d(x0, y);
110                 y = y + dy;
111             } else {
112                 uint dy = (k - xy)*1e18/_d(x0, y);
113                 y = y - dy;
114             }
115             if (y > y_prev) {
116                 if (y - y_prev <= 1) {
117                     return y;
118                 }
119             } else {
120                 if (y_prev - y <= 1) {
121                     return y;
122                 }
123             }
124         }
125         return y;
126     }
127
```

Figure 17 Source code of related functions

Recommendations	It is recommended that the project party improve the algorithm, or judge whether it will cause overflow when users create a trading pair.
Status	Acknowledged.

[Whale-6] Lack event trigger

Severity Level	Info
Type	Coding Conventions
Lines	WhaleswapFactory.sol&FlashmintFactory.sol#L51-59,44-57
Description	In the *setFeeTo* and *setFeeToSetter* functions of the WhaleswapFactory contract, events should be triggered when important parameters are modified. Similarly, event triggers should be added to related functions in the FlashmintFactory contract.

```
49      }
50
51      function setFeeTo(address _feeTo) external {
52          require(msg.sender == feeToSetter, 'Whaleswap: FORBIDDEN');
53          feeTo = _feeTo;
54      }
55
56      function setFeeToSetter(address _feeToSetter) external {
57          require(msg.sender == feeToSetter, 'Whaleswap: FORBIDDEN');
58          feeToSetter = _feeToSetter;
59      }
60
```

Figure 18 Source code of related functions (Unfixed)(WhaleswapFactory.sol)

```
43
44      function setFee(uint256 _fee) external {
45          require(msg.sender == feeSetter, 'Flashmint: FORBIDDEN');
46          fee = _fee;
47      }
48
49      function setFeeTo(address _feeTo) external {
50          require(msg.sender == feeSetter, 'Flashmint: FORBIDDEN');
51          feeTo = _feeTo;
52      }
53
54      function setFeeSetter(address _feeSetter) external {
55          require(msg.sender == feeSetter, 'Flashmint: FORBIDDEN');
56          feeSetter = _feeSetter;
57      }
58
```

Figure 19 Source code of related functions (Unfixed)(FlashmintFactory.sol)

Recommendations	It is recommended to trigger events for related functions.
Status	Fixed.

```
51
52  function setFeeTo(address _feeTo) external {
53      require(msg.sender == feeToSetter, 'Whaleswap: FORBIDDEN');
54      feeTo = _feeTo;
55      emit FeeToUpdated(_feeTo);
56  }
57
58  function setFeeToSetter(address _feeToSetter) external {
59      require(msg.sender == feeToSetter, 'Whaleswap: FORBIDDEN');
60      feeToSetter = _feeToSetter;
61      emit FeeToSetterUpdated(_feeToSetter);
62  }
```

Figure 20 Source code of related functions (Fixed)

```
46
47  function setFee(uint256 _fee) external {
48      require(msg.sender == feeSetter, 'Flashmint: FORBIDDEN');
49      fee = _fee;
50      emit FeeUpdated(_fee);
51  }
52
53  function setFeeTo(address _feeTo) external {
54      require(msg.sender == feeSetter, 'Flashmint: FORBIDDEN');
55      feeTo = _feeTo;
56      emit FeeToUpdated(_feeTo);
57  }
58
59  function setFeeSetter(address _feeSetter) external {
60      require(msg.sender == feeSetter, 'Flashmint: FORBIDDEN');
61      feeSetter = _feeSetter;
62      emit FeeSetterUpdated(_feeSetter);
63  }
```

Figure 21 Source code of related functions (Fixed)

[Whale-7] Redundant codes

Severity Level	Info
Type	Coding Conventions
Lines	WhaleswapPair.sol&FlashERC20.sol&FlashMain.sol#L159-160,55-58,58-60
Description	In the FlashMain and FlashERC20 contracts, the *setWhitelist* function is not used and is redundant code; in the *_mintFee* function of the WhaleswapPair contract, "numerator = numerator.mul(1);" is redundant code, then "denominator = rootK.mul(3).add(rootKLast.mul(1));" The rootKLast in "doesn't need to be multiplied by 1.

```
144    // if fee is on, mint liquidity equivalent to 5/25th (or 1/4th) of the growth in sqrt(k)
145    function _mintFee(uint112 _reserve0, uint112 _reserve1) private returns (bool feeOn) {
146        address feeTo = WhaleswapFactory(factory).feeTo();
147        feeOn = feeTo != address(0);
148        uint _kLast = kLast; // gas savings
149        if (feeOn) {
150            if (_kLast != 0) {
151                uint rootK = Math.sqrt(uint(_reserve0).mul(_reserve1));
152                uint rootKLast = Math.sqrt(_kLast);
153                if (rootK > rootKLast) {
154                    uint numerator = totalSupply.mul(rootK.sub(rootKLast));
155                    uint denominator;
156
157                    // Dynamic fees for stable pairs
158                    if(stable) {
159                        numerator = numerator.mul(1);
160                        denominator = rootK.mul(3).add(rootKLast.mul(1));
161                    }
162                    else {
163                        numerator = numerator.mul(5);
164                        denominator = rootK.mul(20).add(rootKLast.mul(5));
165                    }
166                    uint liquidity = numerator / denominator;
167                    if (liquidity > 0) _mint(feeTo, liquidity);
168                }
169            }
170        } else if (_kLast != 0) {
171            kLast = 0;
172        }
173    }
```

Figure 22 Source code of *_mintFee* function (Unfixed)

```
54
55    function setWhitelist(address addr, bool status) external onlyOwner {
56        whitelistAddr[addr] = status;
57        emit WhitelistUpdate(addr, status);
58    }
59
```

Figure 23 Source code of *setWhitelist* function (Unfixed)(FlashERC20.sol)

```
57
58    function setWhitelist(address addr, bool status) external onlyOwner {
59        whitelistAddr[addr] = status;
60        emit WhitelistUpdate(addr, status);
61    }
62
```

Figure 24 Source code of *setWhitelist* function (Unfixed)(FlashMain.sol)

Recommendations	It is recommended to delete relevant redundant codes.
Status	Fixed.

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party protocol interface consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable token price
		Centralized asset control
		Asset tradability
		Arbitrage attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in Blockchain.

3.4 About BEOSIN

Affiliated to BEOSIN Technology Pte. Ltd., BEOSIN is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions.BEOSIN has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, BEOSIN has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com