BLOCKSEC

Security Audit

Report for Aggregator Contract

Date: March 13, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	PancakeSwap
Target	Aggregator Contract

Version History

Version	Date	Description
1.0	March 13, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-03-13 02:43:31+0000

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The focus of this audit is on the Aggregator Contract of the PancakeSwap, which splits traders' input tokens into multiple DEX protocols to minimize price slippage.

The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The MD5 values of the provided zip files during the audit are shown in the following. Our audit report is responsible for the only initial version (i.e., `Version 1`), as well as new codes (in the following versions) to fix issues in the audit report.

Project	Version	File MD5
Pancake Aggregator	`Version 1`	`442f7ab8906a840dbe9325bda4df5001`
	`Version 2`	`7d8ad8b63284ee13036614abc487576e`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item
* Verification of the token receiver
* Off-chain metadata security

1.3.4 Additional Recommendation

* Gas optimization

* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology and Common Weakness Enumeration. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **four** potential security issues. Besides, we have **six** recommendations and **three** notes.

- High Risk: 2
- Medium Risk: 2
- Recommendation: 6
- Note: 3

ID	Severity	Description	Category	Status
1	High	Improper value assignment for the variable `payer`	DeFi Security	Fixed
2	High	Lack of handling edge cases in the `_execExactInRoute()` function	DeFi Security	Fixed
3	Medium	Lack of validation checks for `orderId`	DeFi Security	Confirmed
4	Medium	Lack of validation checks for `feeConfig` in the functions `swapWrap()` and `swapExactIn()`	DeFi Security	Confirmed
5	-	Add validation checks for the input `path` of the function `_execExactInHop()`	Recommendation	Confirmed
6	-	Refactor the misleading annotation in the function `_execExactInHop()`	Recommendation	Fixed
7	-	Remove the redundant code	Recommendation	Fixed
8	-	Add input validations for the contract `ImmutableState`'s `constructor()`	Recommendation	Fixed
9	-	Add validation checks for `msg.sender` of the function `algebraSwapCallback()`	Recommendation	Fixed
10	-	Add whitelist checks for `adapter` in the function `_execExactInHop()`	Recommendation	Fixed
11	-	Potential centralization risks	Note	-
12	-	Correct inputs constructed in the front-end	Note	-
13	-	Correct configurations for `adapters`	Note	-

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Improper value assignment for the variable `payer`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `swapExactIn()` of the contract `Aggregator`, when the provided `inputToken` is `WETH` and does not match the `fromToken` specified in the `route[*][0]`, users' `WETH` will be unwrapped and transferred to the `Aggregator` contract. However, in this circumstance,

the variable `payer` is not properly assigned as `address(this)`, leading to a loss of funds and a potential DoS issue.

1. When users have enough `WETH` and approve a large amount of `WETH` to the `Aggregator` contract as well as the `route[*][0]`'s adapters do not require wrapping or unwrapping (i.e., `route[*][0].shouldInputWrapped()/shouldInputUnwrapped() == False`), the `Aggregator` contract will transfer users' `WETH` again.
2. When users lack `WETH` and grant limited approvals and the `route[*][0]`'s adapters do not require wrapping or unwrapping, the transaction will revert.

```
148   // in case more than one route requires wrap/unwrap, we batch it together
149   if (amount > 0) {
150       if (inputToken.isETH()) {
151           _wrap(amount);
152           payer = address(this);
153       } else {
154           // means inputToken is WETH9 and routes[i].fromToken has .isETH()
155           _transferInternal(msg.sender, address(this), inputToken, amount);
156           _unwrap(amount);
157       }
158   }
```

Listing 2.1: src/Aggregator.sol

Impact Potential loss of funds and DoS issues due to the incorrect assignment of the variable `payer`.

Suggestion Revise the value assignment logic of the variable `payer`.

2.1.2 Lack of handling edge cases in the `_execExactInRoute()` function

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `_execExactInRoute()` function, if the `fromToken` and `outputToken` of a hop are both `ETH` or `WETH`, the assignment of `curInputAmount` is incorrect. Taking `WETH` as an example, there are two circumstances:

1. If the output amount of `WETH` is greater than the input amount of `WETH`, the assignment of `curInputAmount` will be the difference between the output amount and the input amount rather than the actual value of the output `WETH`. Consequently, only a limited amount of `WETH` will be used in the subsequent route, potentially leading to a revert or loss of funds.
2. If the output amount of `WETH` is less than the input amount of `WETH`, which can occur when users attempt a cyclic arbitrage with a loss, the following calculation (i.e., `curInputAmount = wethBalanceAfterHop - wethBalanceBeforeHop`) will revert due to an underflow.

```
231   for (uint256 i = 0; i < hopLength; ++i) {
232       if (i > 0) {
233           fromToken = hops[i].fromToken;
234
235           // start from the second hop, we need to check if we need to wrap/unwrap the token
```

```
236         // since the output token of the previous hop can contain both ETH and WETH
237         // imagine we have forks in previous hop i.e. [30% usdc->weth, 70% usdc->eth]
238         if (fromToken.isETH()) {
239             uint256 wethBalanceAfterHop = WETH9.balanceOf(address(this));
240             // if previous hop output: WETH and current hop input: ETH, convert WETH from
                    previous hop output to ETH
241             if (wethBalanceAfterHop > wethBalanceBeforeHop) {
242                 _unwrap(wethBalanceAfterHop - wethBalanceBeforeHop);
243             }
244             uint256 ethBalanceAfterHop = address(this).balance;
245             curInputAmount = ethBalanceAfterHop - ethBalanceBeforeHop;
246         } else if (fromToken == address(WETH9)) {
247             uint256 ethBalanceAfterHop = address(this).balance;
248             // if previous hop output: ETH and current hop input: WETH, convert ETH from
                    previous hop output to WETH
249             if (ethBalanceAfterHop > ethBalanceBeforeHop) _wrap(ethBalanceAfterHop -
                    ethBalanceBeforeHop);
250             uint256 wethBalanceAfterHop = WETH9.balanceOf(address(this));
251             curInputAmount = wethBalanceAfterHop - wethBalanceBeforeHop;
252         } else {
253             // previous hop output is not ETH/WETH, then we don't need to worry we will touch
                    input token
254             // for other routes by mistake, as the input token won't be transferred ahead of
                    time
255
256             curInputAmount = fromToken.balanceOf(address(this));
257         }
258         payer = address(this);
259     }
```

Listing 2.2: src/Aggregator.sol

Impact Potential loss of funds or DoS issues due to the lack of handling edge cases in the `_execExactInRoute()` function.

Suggestion Revise the code logic accordingly.

2.1.3 Lack of validation checks for `orderId`

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description In the contract `Aggregator`, the variable `orderId` is not checked in both functions `swapWrap()` and `swapExactIn()`. Users could provide arbitrary or used `orderId` when invoking these functions. As a result, this could lead to conflicts when tracking the variable `orderId`, potentially impacting off-chain operations.

```
65  /// @notice Wrap the ETH to WETH or unwrap WETH to the ETH, if it's wrap then the amount
        should match the msg value
66  /// @param orderId The unique order id which can be used to track the order through the whole
        process
```

```
67  /// @param amount The amount of ETH/WETH to wrap/unwrap
68  /// @param isWrap True if it's wrap, false if it's unwrap
69  /// @param feeConfig The fee configuration for the transaction, usually charged by the 3rd
        party integrator, 0 means no fee
70  /// @return outputAmtReceived What the swapper get as output amount
71  function swapWrap(uint256 orderId, uint256 amount, bool isWrap, uint256 feeConfig)
72      external
73      payable
74      returns (uint256 outputAmtReceived);
75
76  /// @notice Swap exact input tokens for as much output tokens as possible, along the route
        determined by the path.
77  /// @notice Before calling this function, the caller must approve the contract to spend the
        input token
78  /// @param orderId The unique order id which can be used to track the order through the whole
        process
79  /// @param request The basic request information for the exact-in swap
80  /// @param routesAmount It specifies the exact amount of input token that trader wants to
        trades in for each route
81  /// @param routes The detailed information for all hops, it's two demensional array since each
         route could have multiple hops
82  /// @param feeConfig The fee configuration for the transaction, usually charged by the 3rd
        party integrator, 0 means no fee
83  /// @return outputAmtReceived What the swapper get as output amount
84  function swapExactIn(
85      uint256 orderId,
86      BaseExactInRequest calldata request,
87      uint256[] calldata routesAmount,
88      RouterPath[][] calldata routes,
89      uint256 feeConfig
90  ) external payable returns (uint256 outputAmtReceived);
```

Listing 2.3: src/interfaces/IAggregator.sol

Impact Users could provide arbitrary or used `orderId` when calling the functions `swapWrap()` and `swapExactIn()`, potentially impacting off-chain operations.

Suggestion Revise the code logic accordingly.

Feedback from the project We accept this risk for now. The only solution might be to have some form of validation (e.g. signature checking that its generated by us). We'll monitor to see how this go. If its an issue, we'll think of how to fix it with the next version down the road.

2.1.4 Lack of validation checks for `feeConfig` in the functions `swapWrap()` and `swapExactIn()`

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description In the functions `swapWrap()` and `swapExactIn()`, the input `feeConfig` is used to determine the fee ratio and the fee recipient. However, there is a lack of validations for the

input `feeConfig` and users can set the input `feeConfig` as zero to bypass the fee charging mechanism.

```
66  function swapWrap(uint256 orderId, uint256 amount, bool isWrap, uint256 feeConfig)
67      external
68      payable
69      isNotLocked
70      whenNotPaused
71      returns (uint256 outputAmtReceived)
```

Listing 2.4: src/Aggregator.sol

```
111 /// @inheritdoc IAggregator
112 function swapExactIn(
113     uint256 orderId,
114     BaseExactInRequest calldata request,
115     uint256[] calldata routesAmount,
116     RouterPath[][] calldata routes,
117     uint256 feeConfig
118 ) external payable checkDeadline(request.deadline) isNotLocked whenNotPaused returns (uint256
        outputAmtReceived) {
```

Listing 2.5: src/Aggregator.sol

Impact The fee charging mechanism can be bypassed.

Suggestion Revise the code logic accordingly.

Feedback from the project We accept this risk as well.

1. We accept the risk that a savvy user or malicious partner who integrated us can overwrite this.
2. We'll likely charge partners via API key usage – so this `feeConfig` might not be used.

2.2 Recommendations

2.2.1 Add validation checks for the input `path` of the function `_execExactInHop()`

Status Confirmed

Introduced by `Version 1`

Description The param `path` of the function `_execExactInHop()` is a struct of `RouterPath`. This struct consists of four array-type elements (i.e., `mixAdapters`, `assetTo`, `rawData` and `extraData`). It is recommended to add validation checks for the length of listed array-type elements.

```
297 function _execExactInHop(
298     address payer,
299     address receiver,
300     uint256 curInputAmount,
301     RouterPath calldata path,
302     bool requireTransfer
303 ) internal {
304     address fromToken = path.fromToken;
305     uint256 totalWeight;
```

```
306         // execute multiple Adapters for a transaction pair
307         uint256 pathLength = path.mixAdapters.length;
308         for (uint256 i = 0; i < pathLength; ++i) {
309             uint256 adapter = path.mixAdapters[i];
310
311             (address poolAddress, uint256 weight, bool reverse) = path.rawData[i].decodePathRawData
                    ();
312             totalWeight += weight;
313             if (i == pathLength - 1) {
314                 if (totalWeight != 10_000) revert PathWeightInvalid();
315             }
```

Listing 2.6: src/Aggregator.sol

```
46  struct RouterPath {
47      // each hop could have multiple forks (each fork could be a different pool)
48      // it specifies the adapter address that should be used for interacting with the pool
49      // however, some commonly used protocols like pancake, uniswap will be supported inlined
50      // so the adapter address will be special value, check PathAdapterHelper.sol for more
            details
51      uint256[] mixAdapters;
52      // it specifies where should the input token should be sent to while interacting with the
            pool
53      address[] assetTo;
54      // it specifies the required information for interacting with each pool which includes
55      // - reverse flag: by default it should be zero for one swap, but if it's 1 then it should
            be swapped in reverse order
56      // - weight: it's the percentage of input token that should be used for swap, 10000 means
            100%
57      // - pool address: the pool address where the swap should happen
58      uint256[] rawData;
59      // it specifies the extra data that should be passed to the pool while interacting with it
60      bytes[] extraData;
61      // it specifies the input token for current hop (multiple forks will share the same input
            token)
62      address fromToken;
63  }
```

Listing 2.7: src/interfaces/IAggregator.sol

Suggestion Add validation checks for the length of array-type elements in the param `path`.

Feedback from the project I think it's fine to leave it as the transaction always reverts if the length of those array-type elements in path is not the same.

2.2.2 Refactor the misleading annotation in the function `_execExactInHop()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `_execExactInHop`, the annotation (line354) is misleading, which says `/// @dev path.extraData[i] equal to abi.encode(uint160, address, address)`. How-

ever, the `path.extraData[i]` consists of four elements (i.e., `uint160`, `address`, `address`, and `uint24`). It is recommended to revise the annotation.

```
354    /// @dev path.extraData[i] equal to abi.encode(uint160, address, address)
355    uniV3StyledExactInSwap(\_fromTokenAmount, payer, receiver, poolAddress, path.extraData[i]);
```

Listing 2.8: src/Aggregator.sol

Suggestion Revise the annotation.

2.2.3 Remove the redundant code

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The provided contract contains redundant code that should be removed to improve clarity.

1. In the contract `NativeWrapper`, the `InvalidEthSender` error is not used.

```
20      /// @notice Thrown when an unexpected address sends ETH to this contract
21      error InvalidEthSender();
```

Listing 2.9: src/base/NativeWrapper.sol

2. The inheritances of the contract `ImmutableState` in the `Payment` and `NativeWrapper` contracts are redundant.

```
 8      abstract contract Payment is ImmutableState {
 9          using SafeTransferLib for *;
10          using CommonUtils for address;
```

Listing 2.10: src/base/Payment.sol

```
10      abstract contract NativeWrapper is ImmutableState {
11          /// @notice The address for WETH9
12          IWETH9 public immutable WETH9;
```

Listing 2.11: src/base/NativeWrapper.sol

3. The `buildFeeConfig()` function is an internal function and is not used anywhere.

```
42      /// @notice Build the fee configuration from the fee percentage and fee recipient
            address
43      /// @dev this function is only used in test, it's not expected to be used in
            production
44      /// @param feePercentage The fee percentage to be charged on top of the amount (max
            10000 i.e. 100%)
45      /// @param feeRecipient The address of the fee recipient
46      function buildFeeConfig(uint256 feePercentage, address feeRecipient) internal pure
            returns (uint256 feeConfig) {
47          if (feePercentage > MAX_FEE) {
48              revert InvalidFeePercentage();
49          }
50
51          if (feePercentage == 0 || feeRecipient == address(0)) {
```

```
52              return 0;
53          }
54
55          feeConfig = uint256(uint160(feeRecipient)) | (feePercentage << 240);
56      }
```

Listing 2.12: src/libraries/FeeHelper.sol

Suggestion Remove the redundant code.

2.2.4 Add input validations for the contract `ImmutableState`'s `constructor()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `constructor()` of the contract `ImmutableState` does not validate its input params `_uniV3Factory` and `_pcsV3Deployer`. It is recommended to add corresponding validation checks to prevent mis-operations.

```
20  constructor(
21      address _uniV3Factory,
22      bytes32 _uniV3PoolInitCodeHash,
23      address _pcsV3Deployer,
24      bytes32 _pcsV3PoolInitCodeHash
25  ) {
26      UNISWAP_V3_FACTORY = _uniV3Factory;
27      UNISWAP_V3_POOL_INIT_CODE_HASH = _uniV3PoolInitCodeHash;
28      PANCAKESWAP_V3_DEPLOYER = _pcsV3Deployer;
29      PANCAKESWAP_V3_POOL_INIT_CODE_HASH = _pcsV3PoolInitCodeHash;
30  }
```

Listing 2.13: src/base/ImmutableState.sol

Suggestion Add validation checks for the inputs `_uniV3Factory` and `_pcsV3Deployer`.

2.2.5 Add validation checks for `msg.sender` of the function `algebraSwapCallback()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The caller of the function `algebraSwapCallback()` should be validated to ensure it is the corresponding pool.

```
47  function algebraSwapCallback(int256 amount0Delta, int256 amount1Delta, bytes calldata _data)
        external override {
48      if (amount0Delta <= 0 && amount1Delta <= 0) {
49          // swaps entirely within 0-liquidity regions are not supported
50          revert BothDeltaNonPositive();
51      }
52
53      (address tokenIn, address tokenOut, address refundTo) = abi.decode(_data, (address, address
            , address));
54
```

```
55      (bool isExactInput, uint256 amountToPay) =
56          amount0Delta > 0 ? (tokenIn < tokenOut, uint256(amount0Delta)) : (tokenOut < tokenIn,
                uint256(amount1Delta));
57
58      if (isExactInput) {
59          SafeERC20.safeTransfer(IERC20(tokenIn), msg.sender, amountToPay);
60      } else {
61          tokenIn = tokenOut; // swap in/out because exact output swaps are reversed
62          SafeERC20.safeTransfer(IERC20(tokenIn), msg.sender, amountToPay);
63      }
```

Listing 2.14: src/adapters/ThenaConcentratedAdapter.sol

Suggestion Add proper validation checks for `msg.sender` of the `algebraSwapCallback()` function.

2.2.6 Add whitelist checks for `adapter` in the function `_execExactInHop()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `adapter` used in the `_execExactInHop()` function are extracted from user input (i.e., `path`). These should be validated against a whitelist to prevent potential security or functionality issues.

```
297   function _execExactInHop(
298       address payer,
299       address receiver,
300       uint256 curInputAmount,
301       RouterPath calldata path,
302       bool requireTransfer
303   ) internal {
304       address fromToken = path.fromToken;
305       uint256 totalWeight;
306       // execute multiple Adapters for a transaction pair
307       uint256 pathLength = path.mixAdapters.length;
308       for (uint256 i = 0; i < pathLength; ++i) {
309           uint256 adapter = path.mixAdapters[i];
```

Listing 2.15: src/Aggregator.sol

```
356   } else {
357       // execute the swap through the external adapter
358       if (requireTransfer) {
359           _transferInternal(payer, path.assetTo[i], fromToken, _fromTokenAmount);
360       }
361
362       if (reverse) {
363           IAdapter(adapter.bytes32ToAddress()).swapExactOneForZero(receiver, poolAddress, path.
                  extraData[i]);
364       } else {
365           IAdapter(adapter.bytes32ToAddress()).swapExactZeroForOne(receiver, poolAddress, path.
                  extraData[i]);
```

```
366         }
367     }
```

Listing 2.16: src/Aggregator.sol

Suggestion Add whitelist checks for `adapter` decoded from users' input.

2.3 Notes

2.3.1 Potential centralization risks

Introduced by `Version 1`

Description The owner role could conduct privileged operations (i.e., `pause()` and `unpause()`), which introduces potential centralization risks. If the private key of the privileged account is lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.2 Correct inputs constructed in the front-end

Introduced by `Version 1`

Description The swapping results of the `Aggregator` contract are highly dependent on the provided inputs. The project team should ensure that the correct inputs are constructed in the front-end.

2.3.3 Correct configurations for `adapter`s

Introduced by `Version 1`

Description In the `_execExactInHop()` function, the `_fromTokenAmount` variable represents the token balance of `address(this)` and is used as the `amountIn` in the `uniV3StyledExactInSwap()` function. The swapping flow is only correct when the `payer` is set as the `Aggregator` contract. Additionally, in the future, the project team must ensure that `path.assetTo[i]`, `payer`, and the asset receiver are correctly configured for newly introduced `adapter`s.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS