



ParaSwap AugustusV6
Security Audit Report

March 18, 2024

Contents

1	Introduction	3
1.1	About ParaSwap AugustusV6	3
1.2	Source Code	3
2	Overall Assessment	4
3	Vulnerability Summary	5
3.1	Overview	5
3.2	Security Level Reference	6
3.3	Vulnerability Details	7
4	Appendix	18
4.1	About AstraSec	18
4.2	Disclaimer	18
4.3	Contact	19

1 | Introduction

1.1 About ParaSwap AugustusV6

ParaSwap is a decentralized finance (DeFi) platform that aggregates liquidity from various decentralized exchanges (DEXs) to facilitate swift and efficient token swaps. It optimizes trades for the best available rates, minimizes slippage, and provides users with a user-friendly interface for seamless cryptocurrency trading across multiple blockchain networks. Compared with AugustusV5, the audited AugustusV6 introduces several key features:

- New gas optimised architecture for generic swap execution
- Gas efficient fee model
- Support of permit2
- ERC-2535 compatible Diamond proxy
- New FeeVault compatible with multiple versions of Augustus

1.2 Source Code

The following source code was reviewed during the audit:

- <https://github.com/paraswap/paraswap-contracts-v6/tree/fix/fixed-fees2>
- CommitID: d405c95

And this is the final version representing all fixes implemented for the issues identified in the audit:

- <https://github.com/paraswap/paraswap-augustus>
- CommitID: eea7610

Note the `src/executors/generic/Executor*.sol` contracts are out of the audit scope.

2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the ParaSwap AugustusV6 project. Throughout this audit, we identified a total of 8 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	2	-	-	2
Medium	4	-	-	4
Low	2	1	-	1
Informational	-	-	-	-
Undetermined	-	-	-	-

3 | Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

H-1 [Incorrect Offset for Next Pool Address in UniswapV2Utils](#)

H-2 [Revised Multi-Pool Swap in UniswapV3Utils](#)

M-1 [Improper Calldata Overwritten in `_executeSwapOnCurveV2\(\)`](#)

M-2 [Pulling Fee Failure under Permit2 Usage in AugustusFees](#)

M-3 [Improper Parameters Order in `swapExactAmountOutOnUniswapV3\(\)`](#)

M-4 [Improper Fee Charge Logic in AugustusFees](#)

L-1 [Improved swapType Decode for Curve Swap](#)

L-2 [Potential Risks Associated with Centralization](#)

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

[H-1] Incorrect Offset for Next Pool Address in UniswapV2Utils

Target	Category	IMPACT	LIKELIHOOD	STATUS
UniswapV2Utils.sol	Business Logic	High	High	Addressed

In the `UniswapV2Utils` contract, the `_callUniswapV2PoolsSwapExactOut()` function implements the exchange in `UniswapV2` with an exact output amount. When it loads the next pool address from the memory, we notice it doesn't use the correct offset `mul(add(i, 1), 32)`. Instead, it uses the offset `mul(i, 32)` (line 188) which is used to store the current pool address. As a result, the exchanges across multiple pools will fail.

```
UniswapV2Utils::_callUniswapV2PoolsSwapExactOut()

177 // Loop for each pool
178 for { let i := 0 } lt(i, poolCount) { i := add(i, 1) } {
179     // Check if it is the first pool
180     if iszero(poolAddress) {...}

182     // Adjust toAddress depending on if it is the last pool in the array
183     let toAddress := address()

185     // Check if it is not the last pool
186     if lt(add(i, 1), poolCount) {
187         // Load next pool address
188         nextPoolAddress := mload(add(poolAddresses, mul(i, 32)))

190         // Adjust toAddress to next pool address
191         toAddress := nextPoolAddress
192     }
193     ...
194 }
```

Remediation Use the correct offset `mul(add(i, 1), 32)` to load the next pool address.

[H-2] Revised Multi-Pool Swap in UniswapV3Utils

Target	Category	IMPACT	LIKELIHOOD	STATUS
UniswapV3Utils.sol	Business Logic	High	High	Addressed

In the `UniswapV3Utils` contract, the `UniswapV3Utils::_callUniswapV3PoolsSwap()` routine implements the exchange in `UniswapV3` with an exact input amount. For the exchange across multiple

pools, this contract, i.e., `address(this)`, shall serve as both the input payer and the output recipient. However, we notice that the current implementation uses the next pool as the recipient for the current exchange (line 338). This will lead to the exchange failure in the next pool, because this contract, i.e., the payer, has no fund to pay the exchange.

UniswapV3Utils::_callUniswapV3PoolsSwap()

```
307 if lt(add(i, 1), poolCount) {
308     //-----//
309     // Calculate Next Pool Address
310     //-----//
311     // Store 0xff + factory address (right padded)
312     mstore(ptr, uniswapV3FactoryAndFF)
313
314     // Store pools offset + 21 bytes (UNISWAP_V3_FACTORY_AND_FF SIZE)
315     let token0ptr := add(ptr, 21)
316
317     // Copy next pool data to free memory pointer + 21 bytes (
318     // UNISWAP_V3_FACTORY_AND_FF SIZE)
319     calldatacopy(add(token0ptr, 1), add(add(pools.offset, 1), mul(add(i, 1), 96)
320     ), 95)
321
322     // Calculate keccak256(abi.encode(address(token0), address(token1), fee))
323     mstore(token0ptr, keccak256(token0ptr, 96))
324
325     // Store POOL_INIT_CODE_HASH
326     mstore(add(token0ptr, 32), uniswapV3PoolInitCodeHash)
327
328     // Calculate keccak256(abi.encodePacked(hex'ff', address(factory_address),
329     // keccak256(abi.encode(token0,
330     // token1, fee)), POOL_INIT_CODE_HASH));
331     mstore(ptr, keccak256(ptr, 85)) // 21 + 32 + 32
332
333     // Load pool
334     p := mload(ptr)
335
336     // Get the first 20 bytes of the computed address
337     nextPoolAddress := and(p, 0xffffffffffffffffffffffffffffffff)
338
339     // Adjust toAddress to next pool address
340     toAddress := nextPoolAddress
341 }
342 }
```

Moreover, the `UniswapV3Utils::_callUniswapV3PoolsSwapExactAmountOut()` routine implements the exchange in `UniswapV3` with an exact output amount. Our analysis shows that it doesn't properly implement the exchange across multiple pools.

To support the exchanges across multiple pools, it is necessary to call the `pool.swap()` function recursively, with the output of each step serving as the input for the previous step, thus completing

the entire exchange process. We suggest to refer to the [SwapRouter](#) contract from `Uniswap` for implementing exchanges across multiple pools.

Remediation Refer to the [SwapRouter](#) contract from `Uniswap` for implementing exchanges across multiple pools.

[M-1] Improper Calldata Overwritten in `_executeSwapOnCurveV2()`

Target	Category	IMPACT	LIKELIHOOD	STATUS
<code>CurveV2SwapExactAmountIn.sol</code>	Business Logic	Medium	Medium	Addressed

The `_executeSwapOnCurveV2()` function in the `CurveV2SwapExactAmountIn` contract implements the exchanges in `CurveV2`. Before invoking the `exchange()` function of `CurveV2`, it assembles the calldata according to the input data, e.g., `poolAddress`, `i`, `j`, `fromAmount`.

However, we notice the assembled calldata is erroneously overwritten, potentially leading to a failed call to the exchange (line 212).

```
CurveV2SwapExactAmountIn::_executeSwapOnCurveV2()

204 case 0x02 {
205     // Store function selector for function exchange(address,uint256,uint256,
        uint256,uint256)
206     mstore(ptr, 0
        x64a1455800000000000000000000000000000000000000000000000000000000)
207     mstore(add(ptr, 4), poolAddress) // store poolAddress
208     mstore(add(ptr, 36), i) // store index i
209     mstore(add(ptr, 68), j) // store index j
210     mstore(add(ptr, 100), fromAmount) // store fromAmount
211     mstore(add(ptr, 132), 1) // store 1
212     calldatacopy(add(ptr, 4), 164, 160)
213     // Perform the external call with the prepared calldata
214     // Check the outcome of the call and handle failure
215     if iszero(call(gas(), exchange, 0, ptr, 164, 0, 0)) {
216         // The call failed; we retrieve the exact error message and revert with
            it
217         returndatacopy(0, 0, returndatasize()) // Copy the error message to the
            start of memory
218         revert(0, returndatasize()) // Revert with the error message
219     }
220 }
```

Remediation Properly remove the overwrite of the assembled calldata.

[M-2] Pulling Fee Failure under Permit2 Usage in AugustusFees

Target	Category	IMPACT	LIKELIHOOD	STATUS
UniswapV3SwapExactAmountOut.sol AugustusFees.sol	Business Logic	Medium	Medium	Addressed

The `swapExactAmountOutOnUniswapV3()` function is employed to exchange a specified quantity of a given token via `UniswapV3`. It allows users to grant approval to the `UniswapV3SwapExactAmountOut` contract via `permit2` (line 53). Inside it, the `processSwapExactAmountOutFeesAndTransferUniV3()` function is invoked (line 65) after the swap in order to charge fee and transfer the destination token to the beneficiary. Ultimately, the `_distributeFeesUniV3()` function is called to handle fee distribution. Upon thorough examination, it comes to our attention that this function does not consider scenarios where users grant approval through `permit2`. Consequently, this oversight may result in fee distribution failures in such cases.

UniswapV3SwapExactAmountOut::swapExactAmountOutOnUniswapV3()

```
40 function swapExactAmountOutOnUniswapV3(
41     UniswapV3Data calldata uniData, uint256 partnerAndFee, bytes calldata permit
42 ) external payable returns (uint256 receivedAmount, uint256 spentAmount) {
43     ...
44     // Check if we need to wrap or permit
45     if (isFromETH) {
46         ...
47     } else {
48         ...
49         if (permit.length < 257) {
50             ...
51         } else {
52             // Otherwise Permit2.permit
53             permit2Approve(permit);
54             permit2 = 1;
55         }
56         // Swap will be paid from msg.sender
57         fromAddress = msg.sender;
58     }
59     ...
60     if (isFromETH) {
61         ...
62     } else {
63         // Process fees and transfer destToken and srcToken to feeVault or
64         // partner and
65         // feeWallet if needed
66         receivedAmount = processSwapExactAmountOutFeesAndTransferUniV3(
67             beneficiary, srcToken, destToken, partnerAndFee, maxAmountIn,
68             receivedAmount, spentAmount, quotedAmountIn
```

```

68     );
69 }
70 }

```

AugustusFees::processSwapExactAmountOutFeesAndTransferUniV3(&_distributeFeesUniV3())

```

39 function processSwapExactAmountOutFeesAndTransferUniV3(
40     address beneficiary, IERC20 srcToken, IERC20 destToken,
41     uint256 partnerAndFee, uint256 fromAmount, uint256 receivedAmount,
42     uint256 spentAmount, uint256 quotedAmount
43 ) internal returns (uint256 returnAmount) {
44     ...
45     if (partner != address(0x0)) {
46         ...
47         if (feeData & IS_REFERRAL_MASK != 0) {
48             if (surplus > 0) {
49                 // the split is 50% for paraswap, 25% for the referrer and 25%
                    // for the user
50                 uint256 paraswapShare = (surplus * PARASWAP_REFERRAL_SHARE) / 10
                        _000;
51                 uint256 referrerShare = (paraswapShare * 5000) / 10_000;
52                 // distribute fees from srcToken
53                 _distributeFeesUniV3(
54                     remainingAmount, msg.sender, srcToken, partner,
55                     referrerShare, paraswapShare,
56                     skipWhitelist // Check if skipWhitelist flag is true
57                 );
58                 return (receivedAmount);
59             }
60         }
61         ...
62     }
63     ...
64 }

66 function _distributeFeesUniV3(
67     uint256 currentBalance, address payer, IERC20 token,
68     address payable partner, uint256 partnerShare, uint256 paraswapShare,
69     bool skipWhitelist
70 ) private {
71     uint256 totalFees = partnerShare + paraswapShare;
72     if (totalFees != 0) {
73         if (totalFees > currentBalance) {
74             revert InsufficientBalanceToPayFees();
75         }
76         if (skipWhitelist) {
77             // transfer the fees directly to the partner and feeWallet
78             if (paraswapShare > 0) {
79                 token.safeTransferFrom(payer, feeWallet, paraswapShare);

```

```

80         }
81         if (partnerShare > 0) {
82             token.safeTransferFrom(payer, partner, partnerShare);
83         }
84     } else {
85         // transfer the fees to the fee vault
86         token.safeTransferFrom(payer, address(FEE_VAULT), totalFees);
87         if (paraswapShare > 0) {
88             FEE_VAULT.registerFee(feeWalletDelegate, token, paraswapShare);
89         }
90         if (partnerShare > 0) {
91             FEE_VAULT.registerFee(partner, token, partnerShare);
92         }
93     }
94     // othwerwise do not transfer the fees
95 }
96 }

```

Remediation Consider situations where users grant approval through `permit2`.

[M-3] Improper Parameters Order in `swapExactAmountOutOnUniswapV3()`

Target	Category	IMPACT	LIKELIHOOD	STATUS
UniswapV3SwapExactAmountOut.sol	Business Logic	Medium	Medium	Addressed

As previously mentioned, the `swapExactAmountOutOnUniswapV3()` function is employed to exchange a specified quantity of a given token via `UniswapV3`. Within it, if the source token is `ETH`, the `processSwapExactAmountOutFeesAndTransfer()` function is invoked (line 54) after the swap in order to handle fee distribution and transfer both the destination token and any remaining source token to the beneficiary. Upon further analysis, we noticed that the sixth parameter (i.e., `receivedAmount`, line 60) and the seventh parameter (i.e., `remainingAmount`, line 61) are set in the wrong order, which may lead to unexpected or undesired outcomes.

UniswapV3SwapExactAmountOut::swapExactAmountOutOnUniswapV3()

```

40 function swapExactAmountOutOnUniswapV3(
41     UniswapV3Data calldata uniData, uint256 partnerAndFee, bytes calldata permit
42 ) external payable returns (uint256 receivedAmount, uint256 spentAmount) {
43     ...
44     // Check if srcToken is ETH and unwrap
45     if (isFromETH) {
46         uint256 remainingAmount = maxAmountIn - spentAmount;
47
48         // Withdraw remaining WETH if any

```

```

49     if (remainingAmount > 0) {
50         WETH.withdraw(remainingAmount);
51     }

53     // Process fees using processSwapExactAmountOutFeesAndTransfer
54     (spentAmount, receivedAmount) = processSwapExactAmountOutFeesAndTransfer
55     (
56         beneficiary,
57         ERC20Utils.ETH,
58         destToken,
59         partnerAndFee,
60         maxAmountIn,
61         receivedAmount,
62         remainingAmount,
63         quotedAmountIn
64     );
65 } else {
66     ...
67 }

```

AugustusFees::processSwapExactAmountOutFeesAndTransfer()

```

203 function processSwapExactAmountOutFeesAndTransfer (
204     address beneficiary,
205     IERC20 srcToken,
206     IERC20 destToken,
207     uint256 partnerAndFee,
208     uint256 fromAmount,
209     uint256 remainingAmount,
210     uint256 receivedAmount,
211     uint256 quotedAmount
212 ) internal returns (uint256 spentAmount, uint256 returnAmount)
213 { ... }

```

Remediation Rectify the parameter order to ensure the function behaves as intended.

[M-4] Improper Fee Charge Logic in AugustusFees

Target	Category	IMPACT	LIKELIHOOD	STATUS
AugustusFees.sol	Business Logic	Medium	Low	Addressed

The AugustusFees contract implements the logic for charging fees after the completion of swaps, and sends the received/remaining tokens to the beneficiary. While reviewing its logic, we notice the current fee charging logic needs to be revisited.

To elaborate, we take the `processSwapExactAmountInFeesAndTransfer()` routine for example and show below the code snippet from the `AugustusFees`. The fee logic that the contract aims to implement is as follows: if the `IS_REFERRAL_MASK` flag is set to true and `surplus > 0`, then the fee will be charged from the `surplus`. Otherwise, if the `IS_TAKE_SURPLUS_MASK` flag is set to true and `surplus > 0`, then the fee will be charged from the `surplus`. If neither the `IS_REFERRAL_MASK` nor `IS_TAKE_SURPLUS_MASK` flags are set, then fees will be charged based on the `feePercent` configured in the function input parameters. However, the current code implementation allows the fee logic to be bypassed if `surplus == 0` and either the `IS_REFERRAL_MASK` or `IS_TAKE_SURPLUS_MASK` flag is set to true.

AugustusFees::processSwapExactAmountInFeesAndTransfer()

```
74 function processSwapExactAmountInFeesAndTransfer(  
75     address beneficiary,  
76     IERC20 destToken,  
77     uint256 partnerAndFee,  
78     uint256 receivedAmount,  
79     uint256 quotedAmount  
80 )  
81     internal  
82     returns (uint256 returnAmount)  
83 {  
84     ...  
85     // if partner address is not 0x0  
86     if (partner != address(0x0)) {  
87         ...  
88         // if slippage is positive and referral flag is true  
89         if (feeData & IS_REFERRAL_MASK != 0) {  
90             if (surplus > 0) {  
91                 ...  
92             }  
93         }  
94         // if slippage is positive and takeSurplus flag is true  
95         else if (feeData & IS_TAKE_SURPLUS_MASK != 0) {  
96             if (surplus > 0) {  
97                 ...  
98             }  
99         }  
100        // partner takes fixed fees if isTakeSurplus and isReferral flags are  
101        // false,  
102        // and feePercent is greater than 0  
103        else if (feeData & IS_TAKE_SURPLUS_MASK == 0 && feeData &  
104        IS_REFERRAL_MASK == 0) {  
105            uint256 feePercent = _getAdjustedFeePercent(feeData);  
106            if (feePercent > 0) {  
107                ...  
108            }  
109        }  
110    }  
111 }
```

```
110     ...
111 }
```

Note the `processSwapExactAmountOutFeesAndTransfer()` and `processSwapExactAmountOutFeesAndTransferUniV3()` routines share the same issue.

Remediation Implement the correct fee charging logic when `surplus == 0`.

[L-1] Improved swapType Decode for Curve Swap

Target	Category	IMPACT	LIKELIHOOD	STATUS
CurveV1SwapExactAmountIn.sol CurveV2SwapExactAmountIn.sol	Coding Practices	Low	Low	Addressed

In the `CurveV1SwapExactAmountIn` contract, the `swapExactAmountInOnCurveV1()` function is implemented to execute a `swapExactAmountIn` on Curve V1 pools. While examining its logic, we notice that the use of one input parameter is inconsistent with the descriptions provided in the comments for this parameter.

To elaborate, we show below the related code snippet of the `CurveV1SwapExactAmountIn` contract. Based on the comments, the function input parameter `curveV1Data` is encapsulated as follows: the first 160 bits is the target exchange address, bit 161 is the approve flag, bits 162 - 163 are used for the wrap flag, bits 164 - 165 are used for the swapType flag, and the last 91 bits are unused. However, in the current code implementation, bits 165-166 are used as the swapType flag, instead of bits 164 - 165. This may bring unnecessary hurdles to understand and/or maintain the smart contract.

CurveV1SwapExactAmountIn::swapExactAmountInOnCurveV1()

```
33 function swapExactAmountInOnCurveV1(
34     CurveV1Data calldata curveV1Data,
35     uint256 partnerAndFee,
36     bytes calldata permit
37 )
38     external
39     payable
40     returns (uint256 receivedAmount)
41 {
42     ...
44     // solhint-disable-next-line no-inline-assembly
45     assembly ("memory-safe") {
```

```

46     exchange := and(curveData, 0xffffffffffffffffffffffffffffffff)
47     approveFlag := and(shr(160, curveData), 1)
48     wrapFlag := and(shr(161, curveData), 3)
49     swapType := and(shr(164, curveData), 3)
50 }

52     ...
53 }

```

Note that the `CurveV2SwapExactAmountIn` contract shares the same issue.

Remediation Ensure the consistency between documents and implementation.

[L-2] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Low	Low	Acknowledged

In the `ParaSwap AugustusV6` protocol, the existence of a privileged `owner` account introduces centralization risks, as it holds significant control and authority over critical operations governing the protocol. In the following, we show the representative function potentially affected by the privileges associated with the privileged account.

Example Privileged Operations in `ParaSwap AugustusV6`

```

27 function executeSelfdestruct() external onlyOwner {
28     // Selfdestruct the contract
29     selfdestruct(payable(msg.sender));
30 }

32 function setfeeWalletDelegate(address payable _feeWalletDelegate) external
    onlyOwner {
33     // Make sure the fee wallet is not the zero address
34     if (_feeWalletDelegate == address(0)) revert InvalidWalletAddress();
35     // Set the fee wallet
36     feeWalletDelegate = _feeWalletDelegate;
37 }

39 function setTokenBlacklisting(IERC20 token, bool isBlacklisted) public onlyOwner
    {
40     // Set the blacklisting status
41     blacklistedTokens[token] = isBlacklisted;
42     // Emit an event
43     emit TokenBlacklistUpdated(token, isBlacklisted);
44 }

```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

4 | Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Name	AstraSec Team
Phone	+86 176 2267 4194
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI