AstraSec

ParaSwap AugustusV6

Security Audit Report

November 5, 2025

Contents

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 Audit Scope

AugustusV6 Audit

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.

AugustusV6.1 Audit

The following source code was reviewed during the audit:

- https://github.com/paraswap/paraswap-augustus/commits/deployment/v6.1/
- CommitID: 6093d46

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

- https://github.com/paraswap/paraswap-augustus/commits/deployment/v6.1/
- CommitID: d45477e

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

AugustusV6.2 Audit

The following source code was reviewed during the audit:

- https://github.com/paraswap/paraswap-augustus
- CommitID: d45477e

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

- https://github.com/paraswap/paraswap-augustus
- CommitID: f000850

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

Augustus Partner-Full-Fee

The following source code was reviewed during the audit:

- https://github.com/VeloraDEX/paraswap-augustus/commits/feat/partner-full-fee/
- CommitID: 0e1d30c
- Files: src/facets/ProPartnerFacet.sol; src/fees/AugustusFees.sol

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

- https://github.com/VeloraDEX/paraswap-augustus
- CommitID: 8028125

1.3 Changelog

Version	Date
AugustusV6 Audit	January 9, 2024
AugustusV6.1 Audit	May 22, 2024
AugustusV6.2 Audit	May 27, 2024
Augustus Partner-Full-Fee	November 5, 2025

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `ParaSwap AugustusV6` project. Throughout this audit, we identified several 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	3	-	-	3
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

~~H-3~~ Improper Logic with ETH as srcToken in swapExactAmountOutOnBalancerV2()

~~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., `adderss(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)

314     // Store pools offset + 21 bytes (UNISWAP_V3_FACTORY_AND_FF SIZE)
315     let token0ptr := add(ptr, 21)

317     // Copy next pool data to free memory pointer + 21 bytes (
            UNISWAP_V3_FACTORY_AND_FF SIZE)
318     calldatacopy(add(token0ptr, 1), add(add(pools.offset, 1), mul(add(i, 1), 96)
            ), 95)

320     // Calculate keccak256(abi.encode(address(token0), address(token1), fee))
321     mstore(token0ptr, keccak256(token0ptr, 96))

323     // Store POOL_INIT_CODE_HASH
324     mstore(add(token0ptr, 32), uniswapV3PoolInitCodeHash)

326     // Calculate keccak256(abi.encodePacked(hex'ff', address(factory_address),
327     // keccak256(abi.encode(token0,
328     // token1, fee)), POOL_INIT_CODE_HASH));
329     mstore(ptr, keccak256(ptr, 85)) // 21 + 32 + 32

331     // Load pool
332     p := mload(ptr)

334     // Get the first 20 bytes of the computed address
335     nextPoolAddress := and(p, 0xffffffffffffffffffffffffffffffffffffffff)

337     // Adjust toAddress to next pool address
338     toAddress := nextPoolAddress
339 }
```

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.

[H-3] Improper Logic with ETH as srcToken in swapExactAmountOutOnBalancerV2()

Target	Category	IMPACT	LIKELIHOOD	STATUS
BalancerV2SwapExactAmountOut.sol GenericSwapExactAmountOut.sol	Business Logic	High	Medium	Addressed

The `swapExactAmountOutOnBalancerV2()` function is designed to swap `srcToken` for a specified amount of `destToken` via `BalancerV2`. Within this function, it initially obtain the amount of `srcToken` currently held by the contract before executing the swap (line 45). This is done to calculate the remaining amount of `srcToken` after the swap. Upon thorough examination, we observed that if `srcToken` is `ETH`, the amount obtained before the swap includes the incoming `msg.value`, which is clearly not intended. This leads to an incorrect calculation of the remaining amount (line 56).

BalancerV2SwapExactAmountOut::swapExactAmountOutOnBalancerV2()

```
31 function swapExactAmountOutOnBalancerV2(
32     BalancerV2Data calldata balancerData,
33     uint256 partnerAndFee,
34     bytes calldata permit,
35     bytes calldata data
36 )
37     external
38     payable
39     whenNotPaused
40     returns (uint256 spentAmount, uint256 receivedAmount, uint256 paraswapShare,
          uint256 partnerShare)
41 {
42     ...

44     // Check contract balance
45     uint256 balanceBefore = srcToken.getBalance(address(this));

47     ...

49     // Execute swap
50     _callBalancerV2(data);

52     // Check balance of destToken
```

```
    receivedAmount = destToken.getBalance(address(this));

    // Check balance of srcToken, deducting the balance before the swap if it is
        greater than 1
    uint256 remainingAmount = srcToken.getBalance(address(this)) - (
        balanceBefore > 1 ? balanceBefore : 0);

    // Check if swap succeeded
    if (receivedAmount < amountOut) {
        revert InsufficientReturnAmount();
    }

    // Process fees and transfer destToken and srcToken to beneficiary
    return processSwapExactAmountOutFeesAndTransfer(
        beneficiary,
        srcToken,
        destToken,
        partnerAndFee,
        maxAmountIn,
        remainingAmount,
        receivedAmount,
        quotedAmountIn
    );
}
```

Remediation Properly handle the incoming `msg.value` when `srcToken` is `ETH` in the `swapExactAmount-OutOnBalancerV2()` and `swapExactAmountOut()` functions.

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

Target	Category	IMPACT	LIKELIHOOD	STATUS
CurveV2SwapExactAmountIn.sol	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)
```

```
        mstore(ptr, 0x64a1455800000000000000000000000000000000000000000000000000000000)
        mstore(add(ptr, 4), poolAddress) // store poolAddress
        mstore(add(ptr, 36), i) // store index i
        mstore(add(ptr, 68), j) // store index j
        mstore(add(ptr, 100), fromAmount) // store fromAmount
        mstore(add(ptr, 132), 1) // store 1
        calldatacopy(add(ptr, 4), 164, 160)
        // Perform the external call with the prepared calldata
        // Check the outcome of the call and handle failure
        if iszero(call(gas(), exchange, 0, ptr, 164, 0, 0)) {
            // The call failed; we retrieve the exact error message and revert with it
            returndatacopy(0, 0, returndatasize()) // Copy the error message to the start of memory
            revert(0, returndatasize()) // Revert with the error message
        }
    }
```

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
```

```
    if (isFromETH) {
        ...
    } else {
        ...
        if (permit.length < 257) {
            ...
        } else {
            // Otherwise Permit2.permit
            permit2Approve(permit);
            permit2 = 1;
        }
        // Swap will be paid from msg.sender
        fromAddress = msg.sender;
    }
    ...
    if (isFromETH) {
        ...
    } else {
        // Process fees and transfer destToken and srcToken to feeVault or
            partner and
        // feeWallet if needed
        receivedAmount = processSwapExactAmountOutFeesAndTransferUniV3(
            beneficiary, srcToken, destToken, partnerAndFee, maxAmountIn,
            receivedAmount, spentAmount, quotedAmountIn
        );
    }
}
```

AugustusFees::processSwapExactAmountOutFeesAndTransferUniV3()&_distributeFeesUniV3()

```
function processSwapExactAmountOutFeesAndTransferUniV3(
    address beneficiary, IERC20 srcToken, IERC20 destToken,
    uint256 partnerAndFee, uint256 fromAmount, uint256 receivedAmount,
    uint256 spentAmount, uint256 quotedAmount
) internal returns (uint256 returnAmount) {
    ...
    if (partner != address(0x0)) {
        ...
        if (feeData & IS_REFERRAL_MASK != 0) {
            if (surplus > 0) {
                // the split is 50% for paraswap, 25% for the referrer and 25%
                    for the user
                uint256 paraswapShare = (surplus * PARASWAP_REFERRAL_SHARE) / 10
                    _000;
                uint256 referrerShare = (paraswapShare * 5000) / 10_000;
                // distribute fees from srcToken
                _distributeFeesUniV3(
                    remainingAmount, msg.sender, srcToken, partner,
                    referrerShare, paraswapShare,
                    skipWhitelist // Check if skipWhitelist flag is true
                );
                return (receivedAmount);
            }
        }
        ...
    }
    ...
}

function _distributeFeesUniV3(
    uint256 currentBalance, address payer, IERC20 token,
    address payable partner, uint256 partnerShare, uint256 paraswapShare,
    bool skipWhitelist
) private {
    uint256 totalFees = partnerShare + paraswapShare;
    if (totalFees != 0) {
        if (totalFees > currentBalance) {
            revert InsufficientBalanceToPayFees();
        }
        if (skipWhitelist) {
            // transfer the fees directly to the partner and feeWallet
            if (paraswapShare > 0) {
                token.safeTransferFrom(payer, feeWallet, paraswapShare);
            }
            if (partnerShare > 0) {
                token.safeTransferFrom(payer, partner, partnerShare);
            }
        } else {
            // transfer the fees to the fee vault
            token.safeTransferFrom(payer, address(FEE_VAULT), totalFees);
            if (paraswapShare > 0) {
                FEE_VAULT.registerFee(feeWalletDelegate, token, paraswapShare);
            }
            if (partnerShare > 0) {
                FEE_VAULT.registerFee(partner, token, partnerShare);
            }
        }
        // othwerwise do not transfer the fees
    }
}
```

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;

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              beneficiary,
56              ERC20Utils.ETH,
57              destToken,
58              partnerAndFee,
59              maxAmountIn,
60              receivedAmount,
61              remainingAmount,
62              quotedAmountIn
63          );
64      } else {
65          ...
66      }
67  }
```

AugustusFees::processSwapExactAmountOutFeesAndTransfer()

```
function processSwapExactAmountOutFeesAndTransfer(
    address beneficiary,
    IERC20 srcToken,
    IERC20 destToken,
    uint256 partnerAndFee,
    uint256 fromAmount,
    uint256 remainingAmount,
    uint256 receivedAmount,
    uint256 quotedAmount
) internal returns (uint256 spentAmount, uint256 returnAmount)
{ ... }
```

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 blow 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()

```
function processSwapExactAmountInFeesAndTransfer(
    address beneficiary,
    IERC20 destToken,
    uint256 partnerAndFee,
    uint256 receivedAmount,
    uint256 quotedAmount
)
    internal
    returns (uint256 returnAmount)
{
    ...
    // if partner address is not 0x0
    if (partner != address(0x0)) {
        ...
        // if slippage is postive and referral flag is true
        if (feeData & IS_REFERRAL_MASK != 0) {
            if (surplus > 0) {
                ...
            }
        }
        // if slippage is positive and takeSurplus flag is true
        else if (feeData & IS_TAKE_SURPLUS_MASK != 0) {
            if (surplus > 0) {
                ...
            }
        }
        // partner takes fixed fees if isTakeSurplus and isReferral flags are
            false,
        // and feePercent is greater than 0
        else if (feeData & IS_TAKE_SURPLUS_MASK == 0 && feeData &
            IS_REFERRAL_MASK == 0) {
            uint256 feePercent = _getAdjustedFeePercent(feeData);
            if (feePercent > 0) {
                ...
            }
        }
    }

    ...
}
```

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()

```
function swapExactAmountInOnCurveV1(
    CurveV1Data calldata curveV1Data,
    uint256 partnerAndFee,
    bytes calldata permit
)
    external
    payable
    returns (uint256 receivedAmount)
{
    ...

    // solhint-disable-next-line no-inline-assembly
    assembly ("memory-safe") {
        exchange := and(curveData, 0xffffffffffffffffffffffffffffffffffffffff)
        approveFlag := and(shr(160, curveData), 1)
        wrapFlag := and(shr(161, curveData), 3)
        swapType := and(shr(164, curveData), 3)
    }

    ...
}
```

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`

```
function executeSelfdestruct() external onlyOwner {
    // Selfdestruct the contract
    selfdestruct(payable(msg.sender));
}

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

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

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

4.3 Contact

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