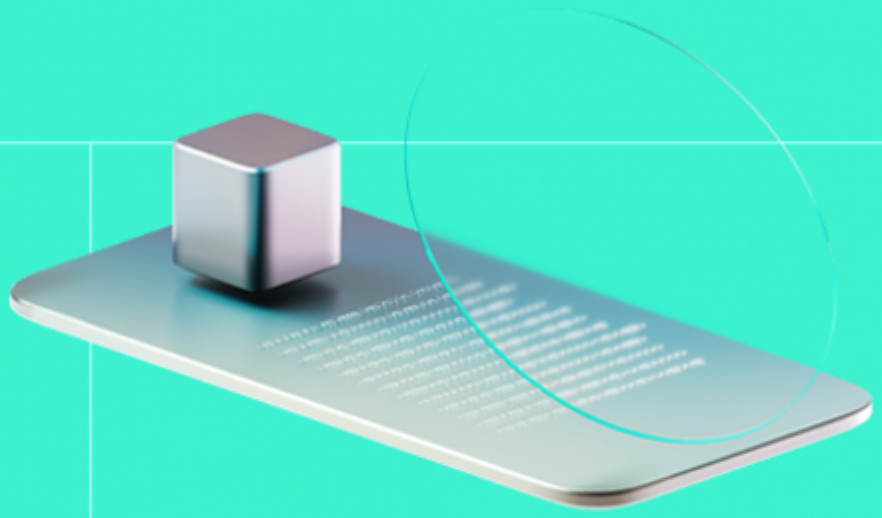
HACKEN

Smart Contract Code Review And Security Analysis Report

Customer: Paraswap

Date: 01/03/2024



We express our gratitude to the Paraswap team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Paraswap is a DeFi aggregator that unites the liquidity of decentralized exchanges and lending protocols into one comprehensive and secure interface and APIs.

Platform: EVM

Language: Solidity

Tags: DeFi aggregator; DEX

Timeline: 08/02/2024 - 01/03/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/paraswap/paraswap-contracts-v6/
Commit	3724093dd6f412ace020799be11b830fda8b8b0a

Audit Summary

9/10	8/10	64.38%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 7.6/10

The system users should acknowledge all the risks summed up in the risks section of the report

9	1	4	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	2
Low	1

Vulnerability	Status
F-2024-1039 - Missing validation for quotedAmount parameter	Accepted
F-2024-1040 - Missing checks for zero address	Accepted
F-2024-1041 - Violation of the Checks-Effects-Interactions (CEI) pattern in AugustusFees.sol	Accepted
F-2024-1050 - Use of deprecated selfdestruct function in SelfdestructFacet.sol	Accepted
F-2024-1042 - Use of transfer instead of call to send native assets	Fixed
F-2024-0924 - Missing return value check in ERC20Utils's getBalance function	Pending Fix
F-2024-0954 - Incorrect swap amount calculations due to overflows and underflows in MakerPSMRouterFacet	Pending Fix
F-2024-1038 - Floating pragma	Pending Fix
F-2024-1052 - Missing events for important state changes	Pending Fix

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Paraswap
Audited By	Carlo Parisi, Eren Gonen, Viktor Raboshchuk
Approved By	Przemyslaw Swiatowiec
Website	https://www.paraswap.io/
Changelog	01/03/2024 - Preliminary Report

Table of Contents

System Overview

ParaSwap aggregates decentralized exchanges and DeFi services in one comprehensive interface to streamline and facilitate users' interactions with decentralized finance on Ethereum and EVM-compatible chains: Polygon, Avalanche, BSC, Optimism, Arbitrum, Base, Polygon zkEVM, Fantom & more.

- AugustusV6.sol - The V6 implementation of the ParaSwap onchain aggregation protocol
- AugustusV6Init.sol - Initialize state varaibles for AugustusV6 Diamond
- AugustusV6Types.sol - Structures variables declaration
- facets/BackwardCompatibilityFacet.sol - A facet that returns the address of the TokenTransferProxy contract
- facets/FeeAdminFacet.sol - A facet for the control the fee related storage variables on AugustusV6
- facets/MakerPSMRouterFacet.sol - A facet for executing direct MakerPSM swaps
- facets/SelfdestructFacet.sol - A facet which allows the contract owner to selfdestruct the contract
- fees/AugustusFees.sol - Contract for handling fees
- fees/AugustusFeeVault.sol - Contract allows partners to collect fees stored in the vault, and allows augustus contracts to register fees
- interfaces/IAugustusFeeVault.sol - Interface for the AugustusFeeVault contract
- interfaces/IAugustusRFQ.sol - Interface for the AugustusRFQ contract
- interfaces/IAugustusRFQRouter.sol - Interface for direct swaps on AugustusRFQ
- interfaces/IBackwardCompatibility.sol - Interface for the BackwardCompatibility contract
- interfaces/IBalancerV2SwapExactAmountIn.sol - Interface for executing swapExactAmountIn directly on Balancer V2 pools
- interfaces/IBalancerV2SwapExactAmountOut.sol - Interface for executing swapExactAmountOut directly on Balancer V2 pools
- interfaces/ICurveV1SwapExactAmountIn.sol - Interface for direct swaps on Curve V1
- interfaces/ICurveV2SwapExactAmountIn.sol - Interface for direct swaps on Curve V2
- interfaces/IErrors.sol - Common interface for errors
- interfaces/IFeeAdmin.sol - Interface for interacting with the FeeAdminFacet contract, which controls fee related storage variables all functions are callable only by the contract owner set by the ownership facet
- interfaces/IGenericSwapExactAmountIn.sol - Interface for executing a generic swapExactAmountIn through an Augustus executor
- interfaces/IGenericSwapExactAmountOut.sol - Interface for executing a generic swapExactAmountOut through an Augustus executor
- interfaces/IMakerPSMRouter.sol - Interface for direct swaps on MakerPSM
- interfaces/ISelfdestruct.sol - Interface for interacting with the SelfdestructFacet contract, which allows the contract owner to selfdestruct the contract
- interfaces/IUniswapV2SwapExactAmountIn.sol - Interface for direct swaps on Uniswap V2
- interfaces/IUniswapV2SwapExactAmountOut.sol - Interface for direct swapExactAmountOut on Uniswap V2
- interfaces/IUniswapV3SwapCallback.sol - Any contract that calls IUniswapV3PoolActions#swap must implement this interface
- interfaces/IUniswapV3SwapExactAmountIn.sol - Interface for executing direct swapExactAmountIn on Uniswap V3
- interfaces/IUniswapV3SwapExactAmountOut.sol - Interface for executing direct swapExactAmountOut on Uniswap V3
- interfaces/IWETH.sol - An interface for WETH IERC20
- libraries/ERC20Utils.sol - A contract with optimized functions for ERC20 tokens
- routers/Routers.sol - A wrapper for all router contracts
- routers/general/AugustusRFQRouter.sol - A contract for executing direct AugustusRFQ swaps
- routers/swapExactAmountIn/GenericSwapExactAmountIn.sol
- routers/swapExactAmountIn/direct/BalancerV2SwapExactAmountIn.sol - A contract for executing direct swapExactAmountIn on Balancer V2
- routers/swapExactAmountIn/direct/CurveV1SwapExactAmountIn.sol - A contract for executing direct CurveV1 swaps
- routers/swapExactAmountIn/direct/CurveV2SwapExactAmountIn.sol - A contract for executing direct CurveV2 swaps

- routers/swapExactAmountIn/direct/UniswapV2SwapExactAmountIn.sol - A contract for executing direct swapExactAmountIn on UniswapV2 pools
- routers/swapExactAmountIn/direct/UniswapV3SwapExactAmountIn.sol - A contract for executing direct swapExactAmountIn on Uniswap V3
- routers/swapExactAmountOut/GenericSwapExactAmountOut.sol - Router for executing generic swaps with exact amount in through an executor
- routers/swapExactAmountOut/direct/BalancerV2SwapExactAmountOut.sol - A contract for executing direct swapExactAmountOut on BalancerV2 pools
- routers/swapExactAmountOut/direct/UniswapV2SwapExactAmountOut.sol - A contract for executing direct swapExactAmountOut on UniswapV2 pools
- routers/swapExactAmountOut/direct/UniswapV3SwapExactAmountOut.sol - A contract for executing direct swapExactAmountOut on UniswapV3 pools
- storage/AugustusStorage.sol - Inherited storage layout for AugustusV6, contracts should inherit this contract to access the storage layout
- util/AugustusRFQUtils.sol - A contract containing common utilities for AugustusRFQ swaps
- util/BalancerV2Utils.sol - A contract containing common utilities for BalancerV2 swaps
- util/GenericUtils.sol - A contract containing common utilities for Generic swaps
- util/MakerPSMUtils.sol - A contract containing swap functions for Maker PSM
- util/Permit2Utils.sol - A contract containing common utilities for Permit2
- util/UniswapV2Utils.sol - A contract containing common utilities for UniswapV2 swaps
- util/UniswapV3Utils.sol - A contract containing common utilities for UniswapV3 swaps
- util/WETHUtils.sol - A contract containing common utilities for WETH
- vendor/Diamond.sol - Implementation of a diamond
- vendor/facet/DiamondCutFacet.sol - A facet that can add/replace/remove any number of functions and optionally execute a function with delegatecall
- vendor/facet/DiamondLoupeFacet.sol - A facet is in charge of documenting other facets. Allowing users to see what the facet addresses and functions are.
- vendor/facet/OwnershipFacet.sol - A facet designed to manage ownership
- vendor/interfaces/IAllowanceTransfer.sol - Handles ERC20 token permissions through signature based allowance setting and ERC20 token transfers by checking allowed amounts
- vendor/interfaces/IDiamondCut.sol - Interface for the DiamondCut contract
- vendor/interfaces/IDiamondLoupe.sol - Interface for the DiamondLoupe contract
- vendor/interfaces/IEIP712.sol - Interface defines function for domain separator value, that unique to each domain
- vendor/interfaces/IERC165.sol - Interface defines function for support of contract interfaces
- vendor/interfaces/IERC173.sol - Interface defines standard functions for owning or controlling a contract.
- vendor/libraries/LibDiamond.sol - Library that provides helper functions to write an application with the Diamond Standard.

Privileged roles

- The owner of the FeeAdminFacet contract can set the fee wallet address, set the second fee wallet address, set the fee blacklisted status of a token, batch set the fee blacklisted status of tokens.
- The owner of the SelfdestructFacet contract can selfdestruct the contract.
- The owner of the AugustusFeeVault contract can set the augustus contract approval status.
- The owner of the Augustv6 coontract can either add/remove/replace a facet on the diamond proxy or transfer the ownership.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.
- NatSpec is present.

Code quality

The total Code Quality score is **8** out of **10**.

- 0 address checks are missing.
- Important events are not emitted.
- Floating pragma is used.

Test coverage

Code coverage of the project is **64.38%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **2** medium, and **1** low severity issues, leading to a security score of **9** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **7.6**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- Numerous contracts are out of scope, including executors, AugustRFQ, Curve, Balance, Uniswap, among others. This poses a risk because the code cannot be reviewed, and the proper functioning of interactions cannot be checked, as these contracts (along with other smaller ones) are excluded from the scope.
- In AugustRFQRouter.sol, there is a check on line 131 that verifies if `afterBalance - beforeBalance < toAmount`. This check might fail if the destToken is a reflexive token or a token with a transfer fee, especially if the `toAmount` is not calculated while considering these characteristics of the token.
- The contracts are upgradeable, which poses a risk when adding new functionalities. Additionally, vulnerabilities could be introduced during the upgrade process.
- Solidity Version Compatibility and Cross-Chain Deployment: The project utilizes Solidity version 0.8.20 or higher, which includes the introduction of the PUSH0 (0×5f) opcode. This opcode is currently supported on the Ethereum mainnet but may not be universally supported across other blockchain networks. Consequently, deploying the contract on chains other than the Ethereum mainnet, such as certain Layer 2 (L2) chains or alternative networks, might lead to compatibility issues or execution errors due to the lack of support for the PUSH0 opcode. In scenarios where deployment on various chains is anticipated, selecting an appropriate Ethereum Virtual Machine (EVM) version that is widely supported across these networks is crucial to avoid potential operational disruptions or deployment failures.

Findings

Vulnerability Details

F-2024-0954 - Incorrect swap amount calculations due to overflows and underflows in MakerPSMRouterFacet - Medium

Description:

The MakerPSMUtils contract, utilized by the MakerPSMRouterFacet for swap operations, contains critical arithmetic vulnerabilities in its `_executeSwapExactAmountInOnMakerPSM` and `_executeSwapExactAmountOutOnMakerPSM` functions. These vulnerabilities stem from potential overflow and underflow scenarios during the calculation of swap amounts, leading to incorrect token swap values.

In the `_executeSwapExactAmountInOnMakerPSM` function, the `gemAmount` variable is calculated using a formula that can result in overflow under certain conditions. Specifically, if the product of `_fromAmount` and `WAD` (10^18) exceeds the maximum value of a `uint256`, an overflow occurs. This can happen when `_fromAmount` is extremely large (e.g., greater than `2^196`). Additionally, if the sum of `WAD` and `toll` exceeds max `uint256`, or if the final product of `((_fromAmount * WAD) / (WAD + toll)) * to18ConversionFactor` surpasses the max `uint256` limit, an overflow is also possible.

_executeSwapExactAmountInOnMakerPSM Overflow Scenarios In Case 0:

- If `(_fromAmount * WAD)` exceeds `2^256`, an overflow occurs. For example, if `_fromAmount` equals **115792089237316195423570985008687907853269984665640564039458** `(2^196 < _fromAmount ≤ 2^256)`, it results in an overflow.
- If `(WAD + toll)` exceeds `2^256`, an overflow occurs.
- If `(_fromAmount * WAD) / (WAD + toll) * to18ConversionFactor` exceeds `2^256`, an overflow occurs.

In the `_executeSwapExactAmountOutOnMakerPSM` function, the calculation of `gemAmount` involves multiple steps where overflow and underflow can occur. For instance, if `toAmount` is exceedingly large (greater than 2^196), the multiplication with `WAD` can overflow. Similarly, the calculation of `b` (the product of `(WAD - toll)` and `to18ConversionFactor`) can overflow if `to18ConversionFactor` is extremely large or underflow if `toll` is greater than `10^18.` The final calculation of `gemAmount` can also result in overflow or underflow depending on the values of **a** and **b**.

_executeSwapExactAmountOutOnMakerPSM Overflow and Underflow Scenarios In The Default Case:

- `a` will overflow if `toAmount` is equal to `(2^196 < toAmount ≤ 2^256)`.
- `b` will overflow if `to18ConversionFactor` is `(2^196 < to18ConversionFactor ≤ 2^256)`, and `(sub(WAD, toll) = 1E18)`.
- `b` will underflow if `toll` is greater than 1E18.
- `gemAmount` will overflow if `a + b` exceeds `2^256`.
- `gemAmount` will underflow if `(a + b = 0)` since `div(sub(add(a, b), 1), b)`.

These arithmetic vulnerabilities can lead to incorrect swap amounts being calculated, resulting in users either receiving fewer tokens than they should or more tokens than intended.

Assets:

- util/MakerPSMUtils.sol [https://github.com/hknio/paraswap-contracts-v6-6d60d0c74979dfeaa/tree/feat/gettokentransferproxy]

Status: Pending Fix

Classification

Severity: Medium

Impact: Likelihood [1-5]: 3

Impact [1-5]: 4

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 2.724296895429098 [Medium]

Recommendations

Recommendation: Either add checks before performing calculations, or, considering that Solidity version ≥0.8 includes overflow/underflow checks, perform this calculation outside of the assembly.

F-2024-1050 - Use of deprecated selfdestruct function in SelfdestructFacet.sol - Medium

Description: The `selfdestruct` function is used in the `SelfdestructFacet.sol` file. This function is deprecated and considered dangerous to use. It can introduce security vulnerabilities. Furthermore, there are plans in the Ethereum roadmap to either completely remove `selfdestruct` or change its behavior in unpredictable ways.

Assets:

- facets/SelfdestructFacet.sol [https://github.com/hknio/paraswap-contracts-v6-6d60d0c74979dfeaa/tree/feat/gettokentransferproxy]

Status: Accepted

Classification

Severity: Medium

Impact: Likelihood [1-5]: 2

Impact [1-5]: 5

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 3.5 [Medium]

Recommendations

Recommendation: It is recommended to remove the use of `selfdestruct` from the smart contract. If the goal is to disable the contract, consider implementing a "circuit breaker" or "pause" functionality that allows certain functions to be disabled, while still allowing others (like a withdrawal function) to be used. This provides more control and safety compared to `selfdestruct`.

Remediation: The team responded that they will deploy on several EVM chains that do not follow the same roadmap as Ethereum. This is a facet that can be removed easily or not deployed at all. The issue was mitigated.

External References:

- EIP-6046
- EIP-4758

F-2024-1042 - Use of transfer instead of call to send native assets - Low

Description:

In lines 119, 146, and 162 of `AugustRFQRouter.sol`, the contract uses built-in `transfer()` function for transferring of ETH to a beneficiary. However, if the beneficiary is a smart contract, they might not be able to receive these ETH due to the gas limit associated with the transfer function in Solidity.

The `transfer()` function was commonly used in earlier versions of Solidity for its simplicity and automatic reentrancy protection. However, it was identified as potentially problematic due to its fixed gas limit of `2300`.

The usage of `transfer()` function can lead to unintended function call revert when the receiving contract's `receive()` or `fallback()` functions require more than `2300` Gas for processing.

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 2

Impact [1-5]: 3

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 2.5 [Low]

Recommendations

Recommendation:

It is recommended to use built-in `call()` function instead of `transfer()` to transfer native assets. This method does not impose a gas limit, it provides greater flexibility and compatibility with contracts having more complex business logic upon receiving the native tokens. When working with then `call()` function ensure that its execution is successful by checking the returned boolean value. It is also recommended to fallow the Check-Effects-Interactions (CEI) pattern in every case to prevent reentrancy issues.

Remediation: `transfer()` has been changed with `safeTransfer()` from `ERC20Utils.sol`.

F-2024-0924 - Missing return value check in ERC20Utils's getBalance function - Info

Description:

There is a missing return value check inside the `ERC20Utils` contract. If the provided address is an Externally Owned Account (EOA) or a contract that either does not return data or returns less data than expected, the `getBalance` function will return incorrect data.

The `getBalance` function uses `staticcall` to query the `balanceOf` function of an ERC20 token.

```
function getBalance(IERC20 token, address account) internal view returns (uint256
balanceOf) {
// solhint-disable-next-line no-inline-assembly
assembly {
switch eq(token, 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE)
// ETH
case 1 { balanceOf := balance(account) }
// ERC20
default {
let x := mload(64) // get the free memory pointer
mstore(x, 0x70a08231000000000000000000000000000000000000000000000000000000000000000) //
store the selector
mstore(add(x, 4), account) // store the account
let success := staticcall(gas(), token, x, 36, x, 32) // call balanceOf
if success { balanceOf := mload(x) } // load the balance
}
}
}
```

If the `token` address is an EOA or a contract that does not return data (or returns less data than expected), the `staticcall` will succeed but return no data. In such cases, the function erroneously returns the function selector for `balanceOf()` (**0×70a082310**), instead of the actual balance or an error. This issue arises because `staticcall` does not validate the length of the returned data, leading to incorrect or misleading results. The function may report incorrect balances, leading to confusion and potential errors in dependent operations.

Assets:

- libraries/ERC20Utils.sol [https://github.com/hknio/paraswap-contracts-v6-6d60d0c74979dfeaa/tree/feat/gettokentransferproxy]

Status: Pending Fix

Classification

Severity: Info

Impact:

Likelihood [1-5]: 3

Impact [1-5]: 1

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 1.38 [Informational]

Recommendations

Recommendation:

Before executing the `getBalance` function, insert a check to determine if the target address is a smart contract.

External References:

- Consensys

F-2024-1038 - Floating pragma - Info

Description: The project uses floating pragmas that is not locked.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Status: Pending Fix

Classification

Severity: Info

Recommendations

Recommendation: It is recommended to specify a fixed compiler version to ensure that the contracts always behave as expected, regardless of any changes in future compiler versions.

F-2024-1039 - Missing validation for quotedAmount parameter - Info

Description:

In the `AugustusFees.sol` file, the `processSwapExactAmountInFeesAndTransfer` function (and several others) calculate the surplus using the quoted amount. However, there are no checks or restrictions on how large or small the quoted amount can be.

This lack of checks could lead to incorrect surplus calculations if an incorrect quoted amount is provided. This could result in users losing funds.

Status: Accepted

Classification

Severity: Info

Recommendations

Recommendation:

It is recommended to add checks to ensure that the quoted amount is within a reasonable range. The quoted amount should be related to the minimum amount and/or the received end amount. This will help prevent incorrect surplus calculations and protect users' funds.

```
require(quotedAmount >= minAmount, "Quoted amount is too low");
require(quotedAmount <= receivedAmount, "Quoted amount is too high");
```

This issue should be addressed to ensure accurate surplus calculations and protect users' funds.

F-2024-1040 - Missing checks for zero address - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "zero address". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address. The "
Missing zero address control" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.
For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

Missing checks were observed in the following contracts:

- `./AugustFee.sol: constructor()`
- `./Routers.sol: constructor()`
- `./ERC20.sol: constructor()`
- `./Vesting.sol: constructor()`

Assets:

- AugustusV6.sol [https://github.com/hknio/paraswap-contracts-v6-6d60d0c74979dfeaa/tree/feat/gettokentransferproxy]
- routers/Routers.sol [https://github.com/hknio/paraswap-contracts-v6-6d60d0c74979dfeaa/tree/feat/gettokentransferproxy]

Status: Accepted

Classification

Severity: Info

Recommendations

Recommendation:

It is strongly recommended to implement checks to prevent the zero address from being set during the initialization of contracts. This can be achieved by adding require statements that ensure address parameters are not the zero address.

F-2024-1041 - Violation of the Checks-Effects-Interactions (CEI) pattern in AugustusFees.sol - Info

Description:

The Checks-Effects-Interactions (CEI) pattern is a best practice in smart contract development that helps prevent re-entrancy attacks. According to this pattern, checks (such as `require` statements) should be done first, followed by effects (state changes), and finally interactions (calls to external contracts).

In the `AugustusFees.sol` file, on lines 379 and 241, the CEI pattern is violated. The interaction with an external contract (`destToken.safeTransfer(beneficiary, receivedAmount)`) is done before all effects are completed.

Assets:

- fees/AugustusFees.sol [https://github.com/hknio/paraswap-contracts-v6-6d60d0c74979dfeaa/tree/feat/gettokentransferproxy]

Status: Accepted

Classification

Severity: Info

Recommendations

Recommendation:

Despite the low risk in this specific case, it is still a best practice to follow the CEI pattern. This helps prevent potential issues in the future if the code is modified. The recommended change would be to complete all effects before calling `safeTransfer`.

External References:

- Security Considerations
- Use the Checks-Effects-Interactions Pattern

F-2024-1052 - Missing events for important state changes - Info

Description:

Events for important state changes should be emitted for tracking actions off-chain.

It was observed that events are missing events in the following functions:

- `setFeeWallet()`
- `setFeeWalletDelegate()`
- `batchSetTokenBlacklisting()`
- `registerFee()`
- `setAugustusApproval()`

Events are crucial for tracking changes on the blockchain, especially for actions that alter significant contract states or permissions. The absence of events in these functions means that external entities, such as user interfaces or off-chain monitoring systems, cannot effectively track these important changes.

Assets:

- facets/FeeAdminFacet.sol [https://github.com/hknio/paraswap-contracts-v6-6d60d0c74979dfeaa/tree/feat/gettokentransferproxy]
- fees/AugustusFeeVault.sol [https://github.com/hknio/paraswap-contracts-v6-6d60d0c74979dfeaa/tree/feat/gettokentransferproxy]

Status: Pending Fix

Classification

Severity: Info

Recommendations

Recommendation: Consider implementing and emitting events for the necessary functions.

Observation Details

F-2024-1054 - Unsafe Arithmetic Conversions - Info

Description: The AugustusFees contract's `_distributeFeesUniV3()` function contains unsafe arithmetic `uint160` conversions that do not perform bounds checking. For valid fee boundaries used in practice this should never overflow and cause problems. However, it remains an advisable practice to use bounds-checked arithmetic.

Status: Pending Fix

Recommendations

Recommendation: Implement explicit checks before casting from **uint256** to **uint160** to ensure that values do not exceed **uint160** limits.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/paraswap/paraswap-contracts-v6/
Commit	3724093dd6f412ace020799be11b830fda8b8b0a
Whitepaper	https://doc.paraswap.network/
Requirements	https://github.com/paraswap/paraswap-contracts-v6/blob/feat/gettokentransferproxy/README.md
Technical Requirements	https://github.com/paraswap/paraswap-contracts-v6/blob/feat/gettokentransferproxy/README.md

Contracts in Scope

AugustusV6.sol

AugustusV6Init.sol

AugustusV6Types.sol

facets/BackwardCompatibilityFacet.sol

facets/FeeAdminFacet.sol

facets/MakerPSMRouterFacet.sol

facets/SelfdestructFacet.sol

fees/AugustusFees.sol

fees/AugustusFeeVault.sol

interfaces/IAugustusFeeVault.sol

interfaces/IAugustusRFQ.sol

interfaces/IAugustusRFQRouter.sol

interfaces/IBackwardCompatibility.sol

interfaces/IBalancerV2SwapExactAmountIn.sol

interfaces/IBalancerV2SwapExactAmountOut.sol

interfaces/ICurveV1SwapExactAmountIn.sol

interfaces/ICurveV2SwapExactAmountIn.sol

interfaces/IErrors.sol

interfaces/IFeeAdmin.sol

interfaces/IGenericSwapExactAmountIn.sol

interfaces/IGenericSwapExactAmountOut.sol

Contracts in Scope

interfaces/IMakerPSMRouter.sol

interfaces/ISelfdestruct.sol

interfaces/IUniswapV2SwapExactAmountIn.sol

interfaces/IUniswapV2SwapExactAmountOut.sol

interfaces/IUniswapV3SwapCallback.sol

interfaces/IUniswapV3SwapExactAmountIn.sol

interfaces/IUniswapV3SwapExactAmountOut.sol

interfaces/IWETH.sol

libraries/ERC20Utils.sol

routers/Routers.sol

routers/general/AugustusRFQRouter.sol

routers/swapExactAmountIn/GenericSwapExactAmountIn.sol

routers/swapExactAmountIn/direct/BalancerV2SwapExactAmountIn.sol

routers/swapExactAmountIn/direct/CurveV1SwapExactAmountIn.sol

routers/swapExactAmountIn/direct/CurveV2SwapExactAmountIn.sol

routers/swapExactAmountIn/direct/UniswapV2SwapExactAmountIn.sol

routers/swapExactAmountIn/direct/UniswapV3SwapExactAmountIn.sol

routers/swapExactAmountOut/GenericSwapExactAmountOut.sol

routers/swapExactAmountOut/direct/BalancerV2SwapExactAmountOut.sol

routers/swapExactAmountOut/direct/UniswapV2SwapExactAmountOut.sol

routers/swapExactAmountOut/direct/UniswapV3SwapExactAmountOut.sol

storage/AugustusStorage.sol

util/AugustusRFQUtils.sol

util/BalancerV2Utils.sol

util/GenericUtils.sol

util/MakerPSMUtils.sol

util/Permit2Utils.sol

util/UniswapV2Utils.sol

util/UniswapV3Utils.sol

util/WETHUtils.sol

vendor/Diamond.sol

vendor/facet/DiamondCutFacet.sol

vendor/facet/DiamondLoupeFacet.sol

Contracts in Scope

vendor/facet/OwnershipFacet.sol

vendor/interfaces/IAllowanceTransfer.sol

vendor/interfaces/IDiamondCut.sol

vendor/interfaces/IDiamondLoupe.sol

vendor/interfaces/IEIP712.sol

vendor/interfaces/IERC165.sol

vendor/interfaces/IERC173.sol

vendor/libraries/LibDiamond.sol