SOLIDIFIED

Audit Report for Paraswap - July 22, 2021

Summary

Audit Report prepared by Solidified covering a subset of the Paraswap smart contracts for the system's v5 release.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on 22 July 2021, and the results are presented here.

Audited Files

The source code has been supplied in the form of a private GitHub repository:

https://github.com/paraswap/paraswap-contracts/tree/feature/v5

Commit number: `f615d82144b9390e1ffaf9ce1a0728bb5ea2424`

The scope of the audit was limited to the following files:

```
original_contracts/AdapterStorageV5.sol
original_contracts/AugustusRegistry.sol
original_contracts/AugustusSwapperV5.sol
original_contracts/TokenTransferProxy.sol
original_contracts/lib/Utils.sol
original_contracts/fee/FeeModel.sol
original_contracts/routers/IRouter.sol
original_contracts/routers/ProtectedMultiPath.sol
original_contracts/routers/SimpleSwap.sol
original_contracts/routers/ZeroXV2.sol
original_contracts/routers/MultiPath.sol
original_contracts/routers/ProtectedSimpleSwap.sol
original_contracts/routers/Uniswap.sol
original_contracts/routers/ZeroXV4.sol
original_contracts/routers/helpers/SimpleSwapHelper.sol
```

Intended Behavior

The ParaSwap smart contracts implement the on-chain component of a DEX aggregator. The audited components include single-path and multi-path token swap contracts and DEX router adapters.

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium	-
Level of Documentation	Medium-low	-
Test Coverage	Medium	-

SOLIDIFIED

Audit Report for Paraswap - July 22, 2021

Issues Found

Solidified found that the Paraswap contracts contain no critical issues, no major issues, 3 minor issues, 1 warning, in addition to 5 informational notes,.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	AugustusSwapper.sol and Utils.sol: Avoid using hard-coded gas limits	Minor	Pending
2	AugustusSwapper.sol: Function initializeAdapter() fails to set the adapterInitialized and adapterVsData mappings	Minor	Pending
3	Protected*.sol: Does not return token and ether	Minor	Pending
4	AugustuSwapper.sol: AugustusSwapper might misbehave with some ERC-20 tokens	Warning	-
5	AugustusRegistry.sol: Contract does not support unbanning a previously banned augustus	Note	-
6	Code Repetition	Note	-
7	Missing validation for nested params	Note	-
8	Swap method assumes from and to tokens are always different	Note	-
9	Misc Notes	Note	-

SOLIDIFIED

Audit Report for Paraswap - July 22, 2021

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. AugustusSwapper.sol and Utils.sol: Avoid using hard-coded gas limits

The function `transferTokens()` limits gas forwarded to 10.000, in the case of ETH transfers. Presumably, this is done to limit the possibility of a receiving smart contract misbehaving, for example through a reentrancy attack.
However, it is generally bad practice to rely on gas costs for opcodes for this purpose, since these may change and have changed in the past. Instead, reentrancy can be avoided using a reentrancy guard (mutex). However, in this case, it does not seem necessary.

Recommendation
Do not limit gas forwarded in ETH transfers.

2. AugustusSwapper.sol: Function initializeAdapter() fails to set the adapterInitialized and adapterVsData mappings

Function `initializeAdapter()` does not update the `adapterInitialized` mapping with the newly initialized adapter, nor does it set `adapterVsData`.

Note
The same issue exists in function `initializeRouter()` with `routerInitialized` and `routerData`.

3. Protected*.sol: Does not return token and ether

The function `retrieve*()` in contracts `ProtectedMultiPath` and `ProtectedSimpleSwap` do not return both token and ether, whereas the contract supports both to be swapped.

Recommendation
Consider allowing the user to retrieve both ether and tokens for every swap using the protected swap contracts.

Warnings

4. AugustuSwapper.sol: AugustusSwapper might misbehave with some ERC-20 tokens

There are some ERC-20 implementations out there and some of them might cause unexpected consequences, such as tokens that charge fees on transfer, malicious implementations, or tokens that return false instead of reverting.

Recommendation
There's not a particular way to deal with this. One option is to add a list of allowed tokens and block execution to others. Another option is to keep this list in the user interface and warn users if they are interacting with tokens that might misbehave.

Informative Notes

5. AugustusRegistry.sol: Contract does not support unbanning a previously banned augustus

Consider adding a function that allows the unbanning of previously a banned `augustus`.

6. Code Repetition

The codebase replicates a number of functions and code segments, in particular, in the routers implementations.

Examples:

Functions `performSimpleSwap()` and `performSimpleBuy()` in `SimpleSwap.sol` and `ProtectedSimpleSwap.sol`.

This code organization makes the codebase larger than necessary and harder to maintain.

Recommendation
Consider refactoring the code to avoid replicated code segments.

7. Missing validation for nested params

The swap method in all routers uses complex data structures as a parameter and is missing validations for most of the values passed through that parameter. This makes the contract completely dependent on the input data for any failures and invalid transfers.

Recommendation
Consider adding more validations to such params.

8. Swap method assumes from and to tokens are always different

The `_swapOn0xV2` method in the `ZeroXV2` contract assumes the `from` and `to` token addresses are different. Specifically if the from and to token address are the same as `Utils.ethAddress()`, only one of it will be updated to `weth` and the method will try to swap the token.

Recommendation
Consider checking if `from` and `to` tokens are the same.

9. Misc Notes

- FeeModel.sol: Constructor does not validate that `partnerSharePercent` and `maxFeePercent` are less than 100%.
- FeeModel.sol: Function `takeFeeAndTransferTokens()` declares `feeStructure` as a `memory` variable. Consider declaring it as `storage` instead to save on gas fees.
- AugustusStorage.sol: `tokenTransferProxy` should be declared as immutable since it's only assigned once in the descendant's constructor.
- MultiPath.sol: Consider declaring functions `multiSwap()` and `megaSwap()` as `external` instead of `public` to save on gas fees. The same is also applicable to `ProtectedMultiPath.protectedMultiSwap()` and `ProtectedMultiPath.protectedMegaSwap()`.
- `ZeroXV2.sol`: Unused variable `_fromToken` in the method `swapOnZeroXv2()`
- `ZeroXV2.sol`: Performs unwanted casting to address type on the swap method. Consider removing duplicate casting.
- `MultiPath.sol`: Consider checking if the path length is less than the `uint8` max value before converting the type in the method `megaSwap`.
- `ZeroXV2.sol` & `ZeroXV4.sol`: The method `swapOnZeroXv2` and `swapOnZeroXv4` are not marked as payable methods.

Audit Report for Paraswap - July 22, 2021

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Paraswap or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.