SOLIDIFIED

Audit Report for ParaSwap - January 20, 2021

Summary

Audit Report prepared by Solidified covering the ParaSwap smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on January 18, 2021, and the results are presented here.

Audited Files

The contracts audited were supplied in a specific branch of a private source code repository:

https://gitlab.com/paraswap/paraswap-contracts/-/tree/audit/02

The latest commit hash covered by this audit is:

`017e9acf4e86603197c851a8cef44dde35c4d82d`

UPDATE: Fixes were received in commit: `1d107fdec927551ba53fb19d37c82d5d7fc2d5f3`

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

```
original_contracts/AugustusSwapper.sol
original_contracts/lib/Utils.sol
original_contracts/TokenTransferProxy.sol
original_contracts/Partner.sol
original_contracts/PartnerRegistry.sol

Uniswap Router:
original_contracts/lib/UniswapV3Lib.sol
original_contracts/UniswapV3Router.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. The contracts included in the scope of this audit are a single- and multi-path swapper contract the partnership program and Uniswap router.

Findings

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 not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

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

Issues Found

Solidified found that the ParaSwap contracts contain no critical issue, 2 major issue, 4 minor issues, in addition to 1 informational note.

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

Issue #	Description	Severity	Status
1	AugustuSwapper.sol: Use with ERC-777 tokens may lead to reentrancy and cause gas refunds to be exploitable	Major	Acknowledged
2	AugustusSwapper.sol: An attacker can drain any residual ETH available in the contract	Minor	Acknowledged
3	AugustusSwapper might misbehave with some ERC-20 token	Minor	Acknowledged
4	AugustuSwapper.sol: performSimpleSwap() does not verify matching parameter array lengths	Minor	Acknowledged
5	Utils.sol: use call() instead of transfer() for Ether transfer	Minor	Resolved
6	Utils.sol: Gas refund depends on hardcoded values	Note	-
7	AugustusSwapper.sol: Redundant assignment in function takeFeeAndTransferTokens()	Note	-

Critical Issues

No critical issues have been found.

Major Issues

1. AugustuSwapper.sol: Use with ERC-777 tokens may lead to reentrancy and cause gas refunds to be exploitable

Attackers can make use of tokens that trigger execution of injected code, such as ERC-777 hooks, to execute arbitrary code and to get gas refunds at the expense of Paraswap.

Also, If any part of the route triggers execution, the swapper contract becomes vulnerable reenttrancy. There is no clear attack scenario for this, but it is important to make users aware of this issue.

Recommendation
Consider simplifying gas management and making users aware of the incompatibility with ERC-777 tokens. 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.

Minor Issues

2. AugustusSwapper.sol: An attacker can drain any residual ETH available in the contract

An attacker can use function `simplBuy()` to drain any residual ETH available in the contract (e.g. ETH that was sent to the contract by mistake). The same attack can also be performed by calling the `buy()` function.

To perform the attack, they would simply call `simplBuy()` without sending any ETH. Since `simplBuy()` currently assumes that all ETH in the contract was sent by the current caller and attempts to return all `remainingAmount` after it is done buying, the attack would succeed and the caller would have effectively drained all available ETH.

Recommendation
Require that `msg.value` be equal to `fromAmount` (in case `fromToken` is `ETH_ADDRESS`), then calculate `residualEth` by subtracting `msg.value` from contract's initial ETH balance. Only return `balance - residualEth` to the caller when the function is done buying.

3. 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.

4. AugustuSwapper.sol: performSimpleSwap() does not verify matching parameter array lengths

In function `performSimpleSwap()` the arrays `callees` and `values` are passed as parameters and should be of equal length. However, the check for this omitted from the precondition checks.

Recommendation
Add the statement to the precondition checks:

```
require(values.length == callees.length);
```

5. Utils.sol: use call() instead of transfer() for Ether transfer

The function `transferTokens()` uses `transfer()` for Ether transfers. This used to be the recommended method, but is not considered best practise anymore. In particular, since the introduction of new gas cost for some opcodes in the Istanbul fork, the gas stipend forwarded

with `transfer()` is not considered sufficient anymore, leading to smart contract receivers not being able to receive Ether, meaning that the transaction will fail.

Recommendation
Consider using `address.call{value: x}()` instead of `transfer()`.

Update
Resolved

Notes

6. Utils.sol: Gas refund depends on hardcoded values

The function `refundGas()` uses hardcoded gas costs in its calculations. However, gas prices may change in future protocol updates, leading to incorrect calculations.

Recommendation
Consider making the values used configurable.

7. AugustusSwapper.sol: Redundant assignment in function takeFeeAndTransferTokens()

Variable `remainingAmount` is first redundantly assigned to `receivedAmount`, then assigned again to `receivedAmount.sub(fee)`.

Recommendation
Remove redundancy to save on gas costs.

Audit Report for ParaSwap - January 20, 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.