SOLIDIFIED

Audit Report for Clipper DEX - August 23, 2021

Summary

Audit Report prepared by Solidified covering the Clipper DEX (formerly known as Galleon) smart contracts.

Process and Delivery

Two (2) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on August 23, 2021, and the results are presented here.

Audited Files

The source code has been supplied in a private source code repository:

https://github.com/shipyard-software/galleon-dex/ (branch: `main`)

Commit number: `abbd7f75238cc060f0dd275c5d0f1aa7eb85b5dc`

Intended Behavior

Clipper is a decentralized exchange (DEX) designed to have the lowest per-transaction costs for small-to-medium-sized trades.

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	Medium	-

Issues Found

Solidified found that the Clipper DEX contracts contain no critical issues, no major issues, 2 minor issues, and 1 informational note.

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

Issue #	Description	Severity	Status
1	UniERC20.sol: Function uniTransferFromSender() can potentially fail when transferring ETH to a smart contract	Minor	Pending
2	ClipperRouter.sol: Validation mismatch between the contract's constructor and function modifyContractAddresses()	Minor	Pending
3	Misc Notes	Note	-

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. UniERC20.sol: Function uniTransferFromSender() can potentially fail when transferring ETH to a smart contract

Function `uniTransferFromSender()` calls `transfer()` when sending ETH to `sendTo`, which only forwards 2300 gas. In cases where the `sendTo` address is a smart contract whose fallback function consumes more than 2300 gas, the call will always fail. This will have the side effect of potentially preventing smart contracts (e.g. DAOs) from receiving transfers.

For a more in-depth discussion of issues with `transfer()` and smart contracts, please refer to `https://diligence.consensys.net/blog/2019/09/stop-using-soliditys-transfer-now/`

Recommendation

Replace instances of `transfer()` with `call()`.

2. ClipperRouter.sol: Validation mismatch between the contract's constructor and function modifyContractAddresses()

The contract's constructor does not validate parameters `poolAddress` and `exchangeAddress` as per the `modifyContractAddresses()` function validation.

Recommendation
Fix mismatch by requiring that `(poolAddress!=address(0)) && exchangeAddress!=address(0)`.

Note
Similarly, `CollectionContract`'s constructor does not validate `poolAddress` and `depositAddress`.

Informational Notes

3. Misc Notes

- ClipperDeposit.sol: Consider using constants for clarity instead of magic numbers for `nDays` operations in function `deposit()`.
- ClipperPool.sol: Consider converting `isTradable()` into a modifier and applying it at the entry point of all public/external functions that take tokens as arguments, rather than performing the check inside of `balancesAndMultipliers()`.
- Consider renaming `ClipperExchangeInterface` to just `ClipperExchange`, as that contract is not an interface, and naming it that way is non-standard and potentially confusing.

Disclaimer