SOLIDIFIED

Audit Report for Dexible - August 16, 2021

Summary

Audit Report prepared by Solidified covering the Dexible smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code. The debrief was on 16 August 2021.

Audited Files

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

https://github.com/BUIDLHub/dexible-contracts

Commit hash: `55888177254b9d74c659a915a197f8d357a3fe27`

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

```
├── BaseAccess.sol
├── BaseConfig.sol
├── IDexRouter.sol
├── IERC20.sol
├── Settlement.sol
├── Types.sol
├── deployment
│   ├── SettlementAdmin.sol
│   └── ZrxAdmin.sol
├── interfaces
│   └── uniswap
│       ├── IPair.sol
│       ├── IPairFactory.sol
│       └── IV2Router.sol
├── libs
│   ├── LibAccess.sol
│   ├── LibConfig.sol
│   └── LibStorage.sol
└── routers
    └── zrx
        └── ZrxRouter.sol
```

Intended Behavior

The smart contracts implements token swaps using multiple decentralized exchanges.

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	High	-
Test Coverage	Medium	-

Issues Found

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

1 Warning aimed at end users has been noted.

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	Settlement.sol: Unsafe casting between uint256 and uint112	Minor	Pending
2	Settlement.sol: Relayers can overcharge the users	Warning	Pending
3	Settlement.sol: Consider supporting relay smart contracts	Note	-
4	Floating Solidity compiler version	Note	-
5	Miscellaneous notes	Note	-

SOLIDIFIED

Audit Report for Dexible - August 16, 2021

Critical Issues

No critical Issues found.

Major Issues

No critical Issues found.

Minor Issues

1. Settlement.sol: Unsafe casting between uint256 and uint112

The function `_preAdjustOrder` creates the `newInput` variable as a `uint256`, then within the return casts it to a `uint112`. This may silently overflow.

Recommendation
Consider creating the `newInput` as a `uint112`, or add a check that the cast `uint112` has not underflowed.

Warnings

2. Settlement.sol: Relayers can overcharge the users

The trusted relayers can set the `ethUSDPrice` and the `feeTokenETHPrice` price using the arguments in the method `fill`. This allows the relayers to choose how much they get paid per gas by manipulating that value.

Furthermore, for tokens that may have pre or post transfer hook functions, the relayer could choose to use a function that wastes tons of gas and use MEV inclusion to split the proceeds with the miner, by submitting the transaction when the `basefee` is low and priority fee/miner tip is high. A small amount of the tx gas would be burnt but the rest could be split between the relayer and miner.

Recommendation
This can be considered as a side effect of the design and is given as a warning for the reader. Alternatively an on-chain price oracle can be used to ensure price accuracy.

Notes

3. `Settlement.sol`: Consider supporting relay smart contracts

The current implementation of the `_postActions` method expects the relayer to be an EOA and any smart contract with a fallback or receive method will have issues interacting with it. This is because the refund function which sends ETH back to the relayer is using the `transfer` method and forwards only 2300 gas.

Recommendation
Consider adding support for smart contract relayers by directly using call to transfer value and specifying a gas allowance, or by paying relayers via WETH.

4. Floating Solidity compiler version

The smart contract does not lock the compiler version to a specific version. It is considered best practice to stick to a single compiler version throughout the codebase.

Recommendation
Consider locking the compiler version.

5. Miscellaneous notes

Miscellaneous notes for improving the code quality and readability.

1. `Settlement.sol` & `ZRXRouter.sol`: The type `uint` implicitly refers to `uint256`. It is recommended to use one approach and use libraries accordingly.
2. Unused variables and event related to penalty - `penaltyFee` and `TraderPenalized`

3. The naming convention set out in the contracts is inconsistently used in the following places:
    - `BaseAcess.sol` function initAccess has no underscore prefix despite being internal.
    - `Settlement.sol` function `_trySwap` has a prefix underscore despite being external.
    - `Settlement.sol` function `performFill` does not have a prefix underscore despite being internal.
    - Deployment contract `ZrxAdmin` has a different file name to contract name. Contract `SettlementAdmin` is identical to `ZrxAdmin`.
4. `Settlement.sol` in `_preCheck` has a commented out safety check. Either remove the unused check or reinstate it.
5. The `README` refers to the contract `ConfigStorage` but no such contract exists. Ensure that documentation is up to date with current contracts.
6. The link https://github.com/BUIDLHub/dexible-contracts/tree/master/contracts in the main `README` gives a 404 error.

Recommendation
Consider updating the code based on the notes.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of BUIDLHub Inc 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.