SOLIDIFIED

Audit Report for Diagonal - March 7, 2022

Summary

Audit Report prepared by Solidified covering the Diagonal smart contract.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code. The debrief meeting was held on 23 February 2022. The fixes were verified on 7 March 2022.

Audited Files

The source code has been supplied in the form of a GitHub repository:
https://github.com/oak-security/audit-diagonal-finance

Commit hash: `11daca92fdfbcb655bb4d8d24d6e60c9679280dc`
Commit hash for fixes: `f794985243b9efc78512614a5d61aae165812f91`

```
├── DiagonalDeployer.sol
├── DiagonalRegistryProxy.sol
├── DiagonalServiceBeacon.sol
├── DiagonalServiceProxy.sol
├── Mocks
│   ├── DiagonalRegistryMock.sol
│   ├── DiagonalServiceMock.sol
│   └── WETH9Mock.sol
├── Multicall2.sol
├── V1
│   ├── DiagonalRegistryV1.sol
│   └── DiagonalServiceV1.sol
├── interfaces
│   ├── IDiagonalDeployer.sol
│   ├── IDiagonalServiceProxy.sol
│   └── V1
│       ├── IDiagonalRegistryV1.sol
│       └── IDiagonalServiceV1.sol
├── libraries
│   ├── Mock
│   │   ├── DiagonalServiceManagementMock.sol
│   │   └── DiagonalServiceStreamsMock.sol
│   └── V1
│       ├── DiagonalServiceManagementV1.sol
│       └── DiagonalServiceStreamsV1.sol
└── utils
    └── DataStructures.sol
```

Intended Behavior

The smart contracts implement a SuperFluid Super App which allows on-chain subscriptions.

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 have their 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	High	-

Issues Found

Solidified found that the Diagonal contracts contain 2 critical issues, no major issues, 3 minor issues and 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	DiagonalServiceBeacon.sol: Anyone can upgrade the contract	Critical	Fixed
2	DiagonalServiceV1.sol: After updated callback can be called by anyone	Critical	Fixed
3	Missing input validations	Minor	Fixed
4	DiagonalServiceStreamsV1.sol: _removePackageId method will not work in all cases	Minor	Fixed
5	Revert vs return during iteration	Minor	Acknowledged
6	Custom upgradable contract patterns may be risky	Note	Acknowledged
7	Miscellaneous code cleanup	Note	Fixed
8	Validate the hardcoded addresses before deploying	Note	Acknowledged
9	DiagonalServiceManagementV1.sol: addPackage() function includes costly operations	Note	Acknowledged
10	Dependency on SuperFluid	Note	Acknowledged

Critical Issues

1. DiagonalServiceBeacon.sol: Anyone can upgrade the contract

The `upgradeTo` method in the `DiagonalServiceBeacon` contract is missing the validation to check who is calling the contract. This ideally should be restricted only to the admin and the current implementation allows anyone to call this method to upgrade the contract code.

```
function upgradeTo(address newImplementation) public {
        _setImplementation(newImplementation);
        emit Upgraded(newImplementation);
}
```

Recommendation
It is recommended to add a modifier to restrict the method from being called by anyone to upgrade it.

Update: Resolved

2. DiagonalServiceV1.sol: After updated callback can be called by anyone

The `afterAgreementUpdated` method in the `DiagonalServiceV1` contract is missing the validation to check who is calling the contract. This is meant to be called only by the `SuperFluid` host, whereas the current implementation allows anyone to call it.

```
external override onlyExpected(agreementClass) returns (bytes memory) {
```

Recommendation
It is recommended to add the `onlyHost` modifier to restrict the method from being called by anyone other than the `SuperFluid` host.

Update: Resolved

Major Issues

No major issues have been found.

Minor Issues

3. Missing input validations

The contracts are missing input validations on several methods which is generally recommended to avoid any unexpected fund loss. The following are some recommendations.

- `DiagonalServiceStreamsV1.sol`: In the method `emergencyCloseSubscription`, consider adding zero address validation for the user address.
- `DiagonalServiceStreamsV1.sol`: In the method transferServiceOwnership, it is recommended to validate the to address for `address(0)`.
- `DiagonalRegistryV1.sol`: The `initialize()` function in the contract updates the `treasury`, but doesn't include any zero address validations for the same.
- `DiagonalServiceManagementV1.sol`: The function `_addPackage()` is missing adequate input validations for the package's state. This allows the package to have `STOPPED` state while adding.

Recommendation
Consider removing the unused variable.

Update: Resolved

4. DiagonalServiceStreamsV1.sol: _removePackageId method will not work in all cases

The method `_removePackageId` will remove the first element if no `packageId` match is found during the search. This does not impact any existing workflow, but this method alone is not a valid implementation for removing an element from an array.

Furthermore, while inserting the packageId using `_startSubscription` method, ensure there are no duplicates being inserted into this array.

Recommendation
Consider fixing the method to not remove the first element when no matching item is found.

Update: Resolved

5. Revert vs return during iteration

In some cases the contracts revert if one validation fails in a `for` loop. This can revert the whole transaction and will fail to commit any operation that has happened before.

For example, the `_safeUpsertSubscription` method reverts if the stream is already closed and prevents other streams from closing. In such cases, consider returning to allow the operation to continue if feasible.

Furthermore, the `_upsertSubscriptionWithContext` method is missing this validation completely.

Recommendation
Consider assessing the feasibility between `revert` and `return` especially when there is a loop involved.

Update: Acknowledged

Notes

6. Custom upgradable contract pattern may be risky

It is recommended to use the existing OpenZeppelin libraries for upgrading the contracts rather than implementing custom solutions, even if those are heavily influenced by the said implementation.

Furthermore, the current implementation is missing some validations that are present in the OpenZeppelin implementation - like validating the code length before deployment.

Recommendation
Consider using the OpenZeppelin implementation to avoid potential vulnerabilities.

Update: Acknowledged

7. Miscellaneous code cleanup

The following are some of the code cleanup comments to improve the readability of the overall code.

- `DiagonalRegistryProxy.sol`: Consider adding an interface IDiagonalServiceProxy for the contract.
- `DiagonalServiceBeacon.sol`: Consider making the `upgradeTo` method external.
- Consider fixing the typos in the comments. Some examples - Intilse, optimsiser.

Recommendation
Consider addressing the miscellaneous comments to improve the code readability.

Update: Resolved

8. Validate the hardcoded addresses before deploying

There are several addresses hardcoded in the contracts and it is recommended to verify them before deploying the contracts.

Recommendation
it is recommended to replace the hardcoded values with deployed ones.

Update: Acknowledged

9. DiagonalServiceManagementV1.sol: addPackage() function includes costly operations

The `addPackage()` function in the `DiagonalServiceManagementV1.sol` includes costly ***for loop*** operations.

The loop in the function relies on an arbitrary number of total packages being passed in the argument. Moreover, every iteration of the for loop, in the `_addPackage()` private function, includes an external call as well.

Recommendation
Consider batching the operations to reduce the gas usage.

Update: Acknowledged

10. Dependency on SuperFluid

Consider minimizing the reliance on SuperFluid to reduce the app from being exposed by exploits in SuperFluid. This will include but not limited to:

- adding more validations on the callback methods to check the input
- Reducing the number of public methods (eg: emergencyCloseSubscription)

Update: Acknowledged

Disclaimer