Diagonal Finance Charge Security Review

Review Resources:

README

Public Documentation

Review Summary

Charge

Charge is a collection of smart contracts that customers can use to manage deployments and upgradeability using beacon proxies and a registry contract.

The core business logic, and the name for which the repository gets its name is the `Charge` contract, which manages the payment of "charge requests" from source to recipients. When a given signature on a charge is valid and submitted by the Diagonal back-end, funds are transferred from a source to any number of recipients in the signed charge's payout array. For more information on the Charge achitecture, the public documentation can be read.

The master branch of the Charge repo was reviewed over 7 days, 2 of which were used to create an initial overview of the contract. The code review was performed between December 19 and December 26, 2022. The code was reviewed by 1 auditor for a total of 20 man hours. The review was limited to one specific commit.

Scope

Code Repo

Commit

The commit reviewed was b5947e4d936008a185668bb78d6f853dd8f82431. The review covered the entire repository at this specific commit but focused on the src directory.

The review was a time-limited review to provide feedback on potential vulnerabilities. The review was not a full audit. The review did not explore all potential attack vectors or areas of vulnerability and may not have identified all potential issues.

The auditor makes no warranties regarding the security of the code and does not warrant that the code is free from defects. The auditor does not represent nor imply to third parties that the code has been audited nor that the code is free from defects. Diagonal finance and third parties should use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Nearly every external function is protected with a modifier, with the exception of the fallback functions used in the proxies. This could be considered a bad thing depending on the intended users of the contracts. See the `Decentralization` section of the Code Evaluation Matrix below for more information on this.
Mathematics	Low	The only contract involving mathematics is the `Charge` contract, which has very simple subtraction. Solidity 0.8.15 is used throughout the contracts, providing integer over and under flow protection.
Complexity	Medium	There is some complexity around proxy and upgradeability management. However, Foundry scripts have been written to manage this deployment complexity, and tests have been written to ensure these contracts behave as expected. That said, there is always risk when upgrading these contracts that you create a storage collision that opens up a security vulnerability. The contracts that manage funds are relatively simple, and have access control to prevent unauthorized external actors from interacting with them.
Libraries	Good	Charge makes good use of OpenZeppelin libraries, only implementing custom logic when needed and OpenZeppelin is the only external dependency. Many of the proxy related custom implementations are heavily influenced by OpenZeppelin proxy libraries. Solidity 0.8.15 is used giving Diagonal Finance the latest solidity features and overflow/underflow protection, which is assumed in various places in the code.
Decentralization	Good	The Diagonal Admin is a gnosis safe multisig on Ethereum mainnet and the `DiagonalRegistry` and `DiagonalOrgBeacon`, which have the power to upgrade the implementation will be administered by a timelock, which is itself administered by a multi-sig. The two bot addresses are EOAs that anyone could control, and only these actors are able to submit charge requests, however, their only role in charge request submission is to proxy signatures from the users to the smart contract, and therefore their attack surface is limited. However, the merchants must trust that the Diagonal Admin is not malicious and won't divert token payouts to unintended recipients.
Code stability	Good	Changes were reviewed at a specific commit hash and the scope was not expanded after the review was started.

Category	Mark	Description
Documentation	Good	The `docs` directory in the repo contains information on how to deploy the contracts and run the tests. There is also extensive internal documentation on the smart contract architecture permissions, access control, and design decisions, as well as external documentation on how to use The Diagonal Charge Network.
Monitoring	Medium	The `DiagonalDeployer`, `OrganizationManagement`, `DiagonalRegistry`, and `DiagonalOrgBeacon` contracts all emit events when important actions are taken. However, no other contracts emit events.
Testing and verification	High	Charge uses Foundry for testing. Foundry coverage was used to produce a coverage report that indicates 90% line coverage, 90% function coverage, and 70% branch coverage. There are also `attack_simulations` tests, which imply that the team considered adversarial actors when writing their tests and designing their architecture. The coverage could be improved by introducing more tests to the `DiagonalDeployer`, `DiagonalTimelockController`, `Base`, `DiagonalOrgProxy`, and `DiagonalRegistryProxy` contracts. Fuzz tests could also easily be included due to the use of Foundry.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements,
- Gas savings
  - Findings that can improve the gas efficiency of the contracts
- Informational
  - Findings including recommendations and best practices

Low Findings

1. Low - `PAUSE_CONTROLLER_ROLE` is unrecoverable if account access is lost

Details

If the account owning the `PAUSE_CONTROLLER_ROLE` is lost for any reason, `pause` and `unpause` can no longer be called.

Impact

Low

Recommendation

Consider adding functions to manage the `PAUSE_CONTROLLER_ROLE` in `DiagonalTimelockController`.

Developer response

It was decided that the risk and the complexity of adding the additional logic outweighed the risk of losing the address associated with the `PAUSE_CONTROLLER_ROLE` role. Additionally the address associated with the `PAUSE_CONTROLLER_ROLE` role is a Gnosis Safe multisig contract (Diagonal developer multisig with 4/9 signature threshold), so losing it is not a real risk.

Gas Savings

1. Gas - `isBot` contains a tautology

Details

`bot == 0xdAfEF6179E02B73C620Ff9aE416ae8091eF0Cc33 || bot == 0xCef9Db135e6CDbB2E18Ba9ea1344E4b1561001Ab` can be returned directly in the `isBot` function.

Impact

Gas Savings

Developer response

Fixed in pull request #58

Informational Findings

1. Informational - Incorrect comment in `DiagonalRegistryProxy`

Details

The comment above `_IMPLEMENTATION_SLOT` suggests that the hash should be `bytes32(uint256(keccak256("eip1967.proxy.beacon")) - 1)`, when in fact it is `bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1)`

Impact

Informational

Recommendation

Fix the comment

Developer response

Fixed in pull request #58

Final remarks

The code is well architected from a security perspective, as is evident by the lack of findings. All sensitive operations have the appropriate access control where possible. The Charge back-end controls a list of allowed tokens that can be used for payment, and also ensures that malicious input is not passed to the charge smart contract functions from an attacker prior to submitting the charge transaction, further limiting the scope for attack. Finally, there are many tests, to include additional `attack_simulations` tests.

The major security risk with [these upgradeability patterns are typically storage collisions](Proxy Upgrade Pattern - OpenZeppelin Docs. Part of the reason for this risk is it makes auditing difficult as it moves these bugs from compile time bugs to runtime bugs which occur during upgrade when there are presumably fewer code reviewers. This has been discussed with the Diagonal team and they are aware of the risks and pitfalls.

Some of the attacks in the `attack_simulations` directory are somewhat naive, so I would like to see more sophisticated attacks in that directory if possible.

As evident by the code coverage mentioned in the code matrix, there are also lines of code in important contracts that are not being covered by tests. It's advisable to attempt to achieve full code coverage when possible, given the adversarial nature of smart contract development.

About The Author

Jackson is a freelance smart contract auditor at Spearbit, yAcademy, and Oak Security. However, Jackson completed this audit in an independent capacity and is therefore unaffiliated with these entities during the engagement that produced this report.

If you or someone you know needs a smart contract audited, they can contact Jackson at jackson.kelley.labs@gmail.com, or on Twitter at https://twitter.com/sjkelleyjr.