RISK AUDIT

for

on

August 06, 2024

Executive Summary

Report

9
/100

TOTAL

Low risk

August 06, 2024

Abstract

Fidesium's automated risk assessment service was requested to perform a risk posture audit on DLC.Link's **contracts**

Repository Link: https://github.com/DLC-link/dlc-solidity

Initial Commit Hash:

`6fdde82a6bf8722cda31fdb2b18b3a80232aa5b4`

Issue Summary

Critical	High	Medium	Low	Info
0 Issues	0 Issues	4 Issues	3 Issues	1 Issues

Caveats

Block Asset's codebase is well written, but does incur a handful of high value flaws.

Test Approach

Fidesium performed both Whitebox and Blackbox testing, as per the scope of the engagement, and relied on automated security testing.

Methodology

The assessment methodology covered a range of phases and employed various tools, including but not limited to the following:

- Mapping Content and Functionality of API
- Application Logic Flaws
- Access Handling
- Authentication/Authorization Flaws
- Brute Force Attempt
- Input Handling
- Source Code Review
- Fuzzing of all input parameter
- Dependency Analysis

Severity Definitions

Critical	The issue can cause large economic losses, large-scale data disorder or loss of control of authority management.
High	The issue puts users' sensitive information at risk or is likely to lead to catastrophic financial implications.
Medium	The issue puts a subset of users' sensitive information at risk, reputation damage or moderate financial impact.
Low	The risk is relatively small and could not be exploited on a recurring basis, or is low-impact to the client's business.
Informational	The issue does not pose an immediate risk but is relevant to security best practices or defence in Depth.

Risk Issues

Vunerability	Description	Risk	Probability	Status
Centralization	The priviliged roles owner, admin, and signer. Have significant modification rights over the contract and its state.	Medium	Low	Open
One Step Ownership Transfer	The DLCBTC contract applies the Ownable pattern.	Medium	Low	Open
Race Condition	The `importData` function on the `DLCManager`contract allows for the update of `dlcBTC` state variable.	Medium	Low	Open
Mising Reentrancy Guard	The `DLCManager` does not utilize the `ReentrancyGuard` for functions that interatct with other contracts.	Medium	Unlikely	Open
Race Condition	The `importData` function on the `DLCManager`contract allows for the update of `btcFeeRecipient`, `btcMintFeeRate`, and `btcRedeemFeeRate`state variables.	Low	Low	Open
Race Condition	The `importData` function on the `DLCManager`contract allows for the update of `minimumDeposit` and `maximumDeposit`state variables.	Low	Low	Open
Unbounded Loop Iteration	The `getAllDLCs` function on the `DLCManager` iterates an array of `DLCLink.DLC`.	Low	Unlikely	Open
Non EOA contract injection	The `onlyVaultCreator` modifier `DLCManager`contract relies on `tx.origin` .	Info	Medium	Open

Risk Overview

Team Risk

Low risk: 1

No issues found in founding team

Doxxing Status	Team Experience	Risk Summary
Public	Highly relevant	Low

Liquidity

Risk summary: N/A

As this is a Github assessment, liquidity risks have not been assessed

Whale Concentration

Risk summary: N/A

As this is a Github assessment, whale risks have not been assessed

Smart Contract Risks

Risk summary: 12

The contracts are well written, and have no major flaws. Recommendations are hygienic/preventative in nature, and primarily focus on Front running avoidance, and user error avoidance.

Vulnerabilities Critical

Current scan criticals Clear

During this scan no critical security vulnerabilities were identified. The assessment covered all key components of the project, including smart contract logic, access controls, and potential attack vectors. While no critical issues were found, we recommend ongoing security monitoring and best practices to maintain the integrity and resilience of the system.

Vulnerabilities High

Current scan criticals Clear

During this scan no critical security vulnerabilities were identified. The assessment covered all key components of the project, including smart contract logic, access controls, and potential attack vectors. While no critical issues were found, we recommend ongoing security monitoring and best practices to maintain the integrity and resilience of the system.

Vulnerabilities Medium

Centralization

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

The priviliged roles owner, admin, and signer. Have significant modification rights over the contract and its state. This is amelioratd by the fact these wallets are hardcoded and not updateable

Recommendations:

- Ensure that these roles are tied to well maintained Multisig wallets.

One Step Ownership Transfer

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

The DLCBTC contract applies the Ownable pattern. It relies on a one step `transferOwnership` strategy. This exposes the contract to accidental ownership transfer to malicious or invalid wallets.

Recommendations:

- Implement `Ownable2Step` to drive a two step ownership transfer. This will require applying `Upgradeable` independently.

Vulnerabilities Medium

Race Condition

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

The `importData` function on the `DLCManager`contract allows for the update of `dlcBTC` state variable. If a function accessing this contract is then called, there could be unexpected behavior or even a call to an unexpected contract. Of particular concern are calls to `withdraw` as they have the weakest access controls.

```
function importData(
    DLCBTC _dlcBTC,
    string calldata _btcFeeRecipient,
    uint256 _minimumDeposit,
    uint256 _maximumDeposit,
    uint256 _btcMintFeeRate,
    uint256 _btcRedeemFeeRate,
    bool _whitelistingEnabled
) external onlyAdmin {
    dlcBTC = _dlcBTC;
    ...
```

Recommendations:

- Ensure that any assets held in the old dlcBTC contract are transferred out before updating the state variable
- Add an additional parameter to every function accessing `dlcBTC` for the expected `dlcBTC` contract. Within those functions compare the provided contract address with the expected contract address. If they do not match, revert the transaction.
- Depending on importance of this issue, and to avoid deliberate frontrunning, a commit reveal scheme could be implemented.

Mising Reentrancy Guard

Vulnerability severity: **Medium**

Vulnerability probability: **Unlikely**

The `DLCManager` does not utilize the `ReentrancyGuard` for functions that interatct with other contracts. Although the contract follows the Checks-Effects-Interactions pattern, it remains best practice to implement ReentrancyGuards as an additional security layer. No specific exploit paths have been identified.

Recommendations:

- Apply `nonReentrant` modifiers to any functions interacting with external contracts.

Vulnerabilities Low

Race Condition

Vulnerability severity: **Low**

Vulnerability probability: **Low**

The `importData` function on the `DLCManager`contract allows for the update of `minimumDeposit` and `maximumDeposit`state variables.

```
function importData(
    DLCBTC _dlcBTC,
    string calldata _btcFeeRecipient,
    uint256 _minimumDeposit,
    uint256 _maximumDeposit,
    uint256 _btcMintFeeRate,
    uint256 _btcRedeemFeeRate,
    bool _whitelistingEnabled
) external onlyAdmin {
    ...
    minimumDeposit = _minimumDeposit;
    emit SetMinimumDeposit(_minimumDeposit);
    maximumDeposit = _maximumDeposit;
    ...
}
```

The `setStatusFunded` then relies on this variable alongside user input. This could result in the value changing or being unexpected at runtime due to transaction ordering. This issue is ameliorated due to access control settings on `setStatusFunded`

Recommendations:

- Add an additional parameter to `setStatusFunded` for each of these variables. Within the function, compare the provided values with the expected values. If they do not match, revert the transaction.
- Depending on importance of this issue, and to avoid deliberate frontrunning, a commit reveal scheme could be implemented. Due to access control settings this is unlikely to be necessary.

Vulnerabilities Low

Race Condition

Vulnerability severity: **Low**

Vulnerability probability: **Low**

The `importData` function on the `DLCManager`contract allows for the update of `btcFeeRecipient`, `btcMintFeeRate`, and `btcRedeemFeeRate`state variables.

```
function importData(
    DLCBTC _dlcBTC,
    string calldata _btcFeeRecipient,
    uint256 _minimumDeposit,
    uint256 _maximumDeposit,
    uint256 _btcMintFeeRate,
    uint256 _btcRedeemFeeRate,
    bool _whitelistingEnabled
) external onlyAdmin {
    ...
    btcFeeRecipient = _btcFeeRecipient;
    ...
    btcMintFeeRate = _btcMintFeeRate;
    emit SetBtcMintFeeRate(_btcMintFeeRate);
    btcRedeemFeeRate = _btcRedeemFeeRate;
    ...
}
```

The `setupVault` then relies on this variable alongside user input. This could result in the value changing or being unexpected at runtime due to transaction ordering. This issue is ameliorated due to access control settings on `setupVault`

Recommendations:

- Add an additional parameter to `setupVault` for each of these variables. Within the function, compare the provided values with the expected values. If they do not match, revert the transaction.
- Depending on importance of this issue, and to avoid deliberate frontrunning, a commit reveal scheme could be implemented. Due to access control settings this is unlikely to be necessary.

Vulnerabilities Low

Unbounded Loop Iteration

Vulnerability severity: **Low**

Vulnerability probability: **Low**

The `getAllDLCs` function on the `DLCManager` iterates an array of `DLCLink.DLC`. If the range provided is very large, this could cause transactions to revert due to running out of gas.

```
function getAllDLCs(
    uint256 startIndex,
    uint256 endIndex
) external view returns (DLCLink.DLC[] memory) {
    if (startIndex >= endIndex) revert InvalidRange();
    if (endIndex > _index) endIndex = _index;

    DLCLink.DLC[] memory dlcSubset = new DLCLink.DLC[](
        endIndex - startIndex
    );

    for (uint256 i = startIndex; i < endIndex; i++) {
        dlcSubset[i - startIndex] = dlcs[i];
    }

    return dlcSubset;
}
```

Recommendations:

- Add an additional `require` check to `getAllDLCs` to prevent the range provided from exceeding a reasonably sized hard cap.
- Add frontend pagination

Vulnerabilities Low

Race Condition

Vulnerability severity: **Low**

Vulnerability probability: **Unlikely**

The `importData` function on the `DLCManager`contract allows for the update of `minimumDeposit` and `maximumDeposit`state variables.

```
function importData(
    DLCBTC _dlcBTC,
    string calldata _btcFeeRecipient,
    uint256 _minimumDeposit,
    uint256 _maximumDeposit,
    uint256 _btcMintFeeRate,
    uint256 _btcRedeemFeeRate,
    bool _whitelistingEnabled
) external onlyAdmin {
    ...
    minimumDeposit = _minimumDeposit;
    emit SetMinimumDeposit(_minimumDeposit);
    maximumDeposit = _maximumDeposit;
    ...
}
```

The `setStatusFunded` then relies on this variable alongside user input. This could result in the value changing or being unexpected at runtime due to transaction ordering. This issue is ameliorated due to access control settings on `setStatusFunded`

Recommendations:

- Add an additional parameter to `setStatusFunded` for each of these variables. Within the function, compare the provided values with the expected values. If they do not match, revert the transaction.
- Depending on importance of this issue, and to avoid deliberate frontrunning, a commit reveal scheme could be implemented. Due to access control settings this is unlikely to be necessary.

Vulnerabilities Informational

Non EOA contract injection

Vulnerability severity: **Info**

Vulnerability probability: **Medium**

The `onlyVaultCreator` modifier `DLCManager`contract relies on `tx.origin`. An attacker could create a malicious contract and trick a user into using it to make a call and injecting additional functionality, as the user would remain `tx.origin`.

```
modifier onlyVaultCreator(bytes32 _uuid) {
    if (dlcs[dlcIDsByUUID[_uuid]].creator != tx.origin) revert NotOwner();
    _;
}
```

Recommendations:

- Add an additional check to `onlyVaultCreator` to ensure that `tx.origin` and `msg.sender` represent the same entity to avoid contract injection.

Disclaimer

Disclaimer

This report is governed by the Fidesium terms and conditions.

This report does not constitute an endorsement or disapproval of any project or team, nor does it reflect the economic value or potential of any related product or asset. It is not investment advice and should not be used as the basis for investment decisions. Instead, this report provides an assessment intended to improve code quality and mitigate risks inherent in cryptographic tokens and blockchain technology.

Fidesium does not guarantee the absence of bugs or vulnerabilities in the technology assessed, nor does it comment on the business practices, models, or regulatory compliance of its creators. All services, reports, and materials are provided "as is" and "as available," without warranties of any kind, including but not limited to merchantability, fitness for a particular purpose, or non-infringement.

Cryptographic assets and blockchain technologies are novel and carry inherent technical risks, uncertainties, and the possibility of unpredictable outcomes. Assessment results may contain inaccuracies or depend on third-party systems, and reliance on them is solely at the Customer’s risk.

Fidesium assumes no liability for content inaccuracies, personal injuries, property damages, or losses related to the use of its services, reports, or materials. Third-party components are provided "as is," and any warranties are strictly between the Customer and the third-party provider.

These services and materials are intended solely for the Customer's use and benefit. No third party or their representatives may claim rights to or rely on these services, reports, or materials under any circumstances.