Djed Protocol

Security Audit Report

July 25, 2024

Contents

1 Introduction **3**

1.1 About Djed 3

1.2 Source Code 3

2 Overall Assessment **4**

3 Vulnerability Summary **5**

3.1 Overview 5

3.2 Security Level Reference 6

3.3 Vulnerability Details 7

4 Conclusion **11**

5 Appendix **12**

5.1 About AstraSec 12

5.2 Disclaimer 12

5.3 Contact 12

1 Introduction

1.1 About Djed

`Djed` is an algorithmic stablecoin protocol that behaves like an autonomous bank that buys and sells stablecoins for a price in a range that is pegged to a target price. It is crypto-backed in the sense that the bank keeps a volatile cryptocurrency in its reserve. The reserve is used to buy stablecoins from users that want to sell them. And revenue from sales of stablecoins to users are stored in the reserve. Besides stablecoins, the bank also trades reservecoins in order to capitalize itself and maintain a reserve ratio significantly greater than one.

1.2 Source Code

The following repository and commit hash are used in this audit:

- https://github.com/DjedAlliance/Djed-Solidity
- CommitID: ebc2b0a

And this is the final repository and commit hash after all fixes for the issues found in the audit have been checked in:

- https://github.com/DjedAlliance/Djed-Solidity
- CommitID: f1d00c3

Note that in this audit, we only cover the following smart contracts: `HebeSwapInvertingOracle.sol` and the code change to `Djed.sol` in code-diff. What is more, we assume the price oracles involved in `Djed` protocol are reliable and they are not in the audit scope.

2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `Djed` protocol. Throughout this audit, we notice the contracts are well designed and engineered. We identified a total of 3 issues of low severity. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	-	-	-	-
Medium	-	-	-	-
Low	3	1	-	2
Informational	-	-	-	-
Total	3	1	-	2

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~L-1~~ Suggested Usage of immutable in HebeSwapInvertingOracle

~~L-2~~ Improved Price Feed in HebeSwapInvertingOracle

L-3 Revisited Validation of The Last Update Time for Price

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

[L-1] Suggested Usage of immutable in HebeSwapInvertingOracle

Target	Category	IMPACT	LIKELIHOOD	STATUS
HebeSwapInvertingOracle.sol	Coding Practices	Low	N/A	Addressed

In the `Djed` protocol, the `HebeSwapInvertingOracle` contract is used to read the `USD/ETC` price from `HebeSwap`. The price is retrieved by interacting with the configured `ref`, which is the `StdReference` of `HebeSwap`.

While reviewing the usage of the `ref` state variable, we notice it is assigned only once in the `constructor()`. Therefore, it is suggested to declare the `ref` state variable with the `immutable` keyword.

With the `immutable` keyword, the state variables provide a way to define values that are fixed upon deployment and remain constant throughout the contract's lifecycle. Furthermore, accessing `immutable` variables is much cheaper than accessing regular state variables because they are stored in the contract's bytecode rather than in the more expensive storage slots.

Based on this, we suggest declaring the `ref` state variable with the `immutable` keyword, similar to the `scalingFactor` state variable (line 9).

HebeSwapInvertingOracle.sol

```
2  pragma solidity ^0.8.0;

4  import "@hebeswap/IStdReference.sol";
5  import "./IOracle.sol";

7  contract HebeSwapInvertingOracle is IOracle {
8      IStdReference public ref;
9      uint256 public immutable scalingFactor;

11     constructor(IStdReference _ref, uint8 _decimals, uint8 _hebeSwapDecimals) {
12         ref = _ref;
13         scalingFactor = 10**(uint256(_decimals + _hebeSwapDecimals) );
14     }
15     ...
16 }
```

Remediation Declare the `ref` state variable with the `immutable` keyword for improved gas efficiency.

[L-2] Improved Price Feed in HebeSwapInvertingOracle

Target	Category	IMPACT	LIKELIHOOD	STATUS
HebeSwapInvertingOracle.sol	Business	Low	Low	Addressed

In the `HebeSwapInvertingOracle` contract, the `readData()` function is designed to return the `USD/ETC` price for the `Djed` contract. The price is used in the operations of buying and selling stablecoins with `ETC`. Specifically, it calls `ref.getReferenceData("ETC", "USDT")` to read the `ETC/USDT` price from `HebeSwap` (line 19) and reverts the price to get the `USDT/ETC`, which will be used as the reference price of `USD/ETC`.

However, our study shows that the price of `USDT/USD` also fluctuates, making the `USDT` price differ from $1 `USD`. As a result, the price of `ETC/USD` may not equal to the price of `ETC/USDT`. Based on this, it is suggested to check and apply a more precise price feed for the `ETC/USD` price.

HebeSwapInvertingOracle::readData()

```
18 function readData() external view returns (uint256) {
19     IStdReference.ReferenceData memory data = ref.getReferenceData("ETC", "USDT"
           );
20     return (scalingFactor) / uint256(int256(data.rate));
21 }
```

Remediation Properly check and apply a more precise price feed for the `ETC/USD` price.

Response By Team The oracle of `HebeSwap` is actually providing the price of `ETC` in `USD`, even though they are calling it `USDT`. We have addressed the `USDT/USD` issue by making the base token and quote token variables set by the constructor, instead of hard-coded constants.

[L-3] Revisited Validation of The Last Update Time for Price

Target	Category	IMPACT	LIKELIHOOD	STATUS
HebeSwapInvertingOracle.sol	Coding Practices	Low	Low	Acknowledged

In the `HebeSwapInvertingOracle` contract, the `USDT/ETC` price is read from `HebeSwap` to serve as the reference price for `SC/ETC`. Upon examining the integration of HebeSwap price data, we identify a potential issue where the retrieved price may be outdated.

To elaborate, we show below the related code snippet of the `HebeSwapInvertingOracle::readData()` function. Specifically, it reads the `ETC/USDT` price from `HebeSwap` (line 19) and inverts the price to get the `USDT/ETC` price with the specified decimals.

HebeSwapInvertingOracle::readData()

```
18 function readData() external view returns (uint256) {
19   IStdReference.ReferenceData memory data = ref.getReferenceData("ETC", "USDT");
20   return (scalingFactor) / uint256(int256(data.rate));
21 }
```

By referring to the definition of the `IStdReference::getReferenceData()` interface from the HebeSwap document, it returns a `ReferenceData` struct as shown in the code snippet below. In the `ReferenceData` struct, the `lastUpdatedBase` parameter represents the last time when the base price was updated (line 9), and the `lastUpdatedQuote` parameter represents the last time when the quoted price was updated (line 10). Normally, the price for each token can be updated promptly, ensuring that consumers use the latest price. However, in some edge cases, if the price update is delayed, the price read from `HebeSwap` may be outdated. Applying an outdated price in the `Djed` protocol could lead to unpredictable consequences, potentially resulting in user fund losses.

Our analysis shows that we can check the last update time for both the base token and the quote token in the `HebeSwapInvertingOracle::readData()` function. If either of these times exceeds a predetermined threshold, the outdated price should be discarded to protect the protocol.

IStdReference::ReferenceData

```
7  /// A structure returned whenever someone requests for standard reference data.
8  struct ReferenceData {
9    uint256 rate; // base/quote exchange rate, multiplied by 1e18.
10   uint256 lastUpdatedBase; // UNIX epoch of the last time when base price gets
         updated.
11   uint256 lastUpdatedQuote; // UNIX epoch of the last time when quote price gets
          updated.
12 }
```

Remediation Properly validate the last update time for both the base token and the quote token, and revert the transaction if the retrieved price is outdated.

Response By Team We are aware of the risks associated with outdated prices, but there are several tradeoffs that need to be considered when addressing such risks. For example, the suggested remediation would have the drawback that users would be unable to redeem stablecoins until the oracle submits a new price with a recent timestamp. Furthermore, an old timestamp does not necessarily mean that the corresponding price is outdated and wrong, since oracles may follow the policy of only submitting a new price on-chain when it deviates from the previously submitted price by more than a threshold. Thus, the suggested remediation could prevent users from redeeming stablecoins when the price is up-to-date and correct albeit having an old timestamp. All pros and cons

considered, in the Osiris version of Djed, a conscious decision was made not to take the timestamp into account.

4 Conclusion

In this audit, we have analyzed the code changes to the smart contracts of the `Djed` protocol. `Djed` is an algorithmic stablecoin protocol that behaves like an autonomous bank that buys and sells stablecoins for a price in a range that is pegged to a target price. It is crypto-backed in the sense that the bank keeps a volatile cryptocurrency in its reserve. The reserve is used to buy stablecoins from users that want to sell them. And revenue from sales of stablecoins to users are stored in the reserve. Besides stablecoins, the bank also trades reservecoins in order to capitalize itself and maintain a reserve ratio significantly greater than one.

The current code base is well-structured and neatly organized. Those identified issues have been promptly acknowledged and fixed.

5 Appendix

5.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

5.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

5.3 Contact

Phone	+86 176 2267 4194
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI