Least Authority
PRIVACY MATTERS

Decred DEX Zcash Integration
Security Audit Report

ZCG

Final Audit Report: 9 December 2024

Table of Contents

Overview

Background

As the Zcash Ecosystem Security Lead, Least Authority was asked to perform a security audit of the Decred DEX Zcash (ZEC) integration. Decred DEX is a decentralized exchange built by the Decred Project, and the scope of this audit is Zcash's (ZEC) integration with the exchange.

Project Dates

- **September 30, 2024 - October 17, 2024:** Initial Code Review *(Completed)*
- **October 18, 2024:** Delivery of Initial Audit Report *(Completed)*
- **December 9, 2024:** Verification Review *(Completed)*
- **December 9, 2024:** Delivery of Final Audit Report *(Completed)*

Review Team

- Anna Kaplan, Cryptography Researcher and Engineer
- Dominic Tarr, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Decred DEX Zcash Integration followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- decred/dcrdex: https://github.com/decred/dcrdex/tree/master/client/asset/zec

Specifically, we examined the Git revision for our initial review:

- `133e1a05a916f4cd4268f25dcda4d4b8d8be58a1`

For the verification, we examined the Git revision:

- `a4af879cd5828e0315626e2fc35e72df44cbdb59`

For the review, this repository was cloned for use during the audit and for reference in this report:

- decred/dcrdex: https://github.com/LeastAuthority/decred-dcrdex

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- README: https://github.com/decred/dcrdex/blob/master/README.md

In addition, this audit report references the following documents:

- Atomic Swap:
  https://github.com/decred/atomicswap
- Main documentation:
  https://github.com/decred/dcrdex/tree/master/spec

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Whether requests are passed correctly to the network core;
- Key management, including secure private key storage and management of encryption and signing keys;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution;
- Protection against malicious attacks and other ways to exploit;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Decred DEX is a non-custodial cross-chain exchange platform. While the platform is decentralized and peer-to-peer, the orderbook and order matching is performed by a centralized server. Atomic swaps enable the coin exchanges (more on this here) with users exchanging directly via four transaction atomic swaps. Due to the `FundOrder` step, an additional fifth transaction method is possible, which proves to the Decred DEX server that the user controls funds sufficiently for the order. Decred DEX does not collect trading fees, and only network fees apply (i.e., mining transaction fees), although users must pay a registration fee to the server, which may be forfeited as a penalty for incorrect behavior. Since the exchange is non-custodial, users need to be actively running their instances if they have orders pending.

Decred DEX consists of a core implementation and plugins for a range of supported assets, which implement an asset interface. Our team audited only the Zcash asset implementation, which is responsible for the various transactions required in the atomic swap, as well as the calculation of the fees.

We did not find any security issues relating to these areas of investigation; however, our team noted that one limitation to the design is that Zcash users can only do atomic swaps with transparent addresses.

System Design

Our team examined the new addition of Zcash Orchard to Decred DEX and found that it has been well-designed, with security at the forefront of the design considerations. The implementation is open source and uses concurrency safely due to locking mechanisms.

In our review, we investigated the generation of wallets, along with the handling of balances against reserves, and did not find any issues.

We also reviewed the synchronization processes of the transaction history and the adjunct tip of blockchain reporting. We identified two suggestions regarding the elimination of silent errors within the synchronization process (Suggestion 1) and the increasing of the blocks for the query buffer (Suggestion 2).

Additionally, we reviewed the signing processes as well as the orderbook mechanism, and did not identify any issues.

Code Quality

We performed a manual review of the repositories in scope and found the code to be generally well-organized. However, the code is written in a procedural style with very little abstraction used, thereby making the codebase excessively long for the given amount of functionality. Our team noted that this reduces the readability of the code and, thus, makes reasoning about the security of the system more difficult.

Tests

The repositories in scope include sufficient test coverage.

Documentation and Code Comments

The project documentation provided by the Decred DEX team was sufficient in describing the intended functionality of the system. In particular, our team found the exchange specification to be especially accurate and helpful. Additionally, code comments sufficiently describe the intended behavior of security-critical components and functions. While we did identify some typographical errors in the code comments, this did not hinder our understanding nor did it affect the correctness of the code.

Scope

The scope of this review was limited to the Zcash features, which constitute only a small part of the overall system. Our team also noted that there was more code relevant to the Zcash integration in `dex/networks/zec`, which is an internal dependency of the in-scope `client/assets/zec`. While in some instances during the review, it was necessary to refer to it to better understand the in-scope components, we did not review it in depth, as it was out of the scope of this audit. We recommend performing a comprehensive, follow-up security audit of the Decred DEX core, as well as the integrations for other assets.

Dependencies

We examined all the dependencies implemented in the codebase and found only one external dependency, `btcsuite/btcd`, is used, which is a BTC implementation in Golang that has been stable, thoroughly tested, and run in production since 2013. Hence, we did not identify any vulnerabilities in the implementation's use of dependencies.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Suggestion 1: Update Silent Errors for Syncing	Partially Resolved
Suggestion 2: Increase blockQueryBuffer	Resolved

Suggestions

Suggestion 1: Update Silent Errors for Syncing

Location

`asset/zec/zec.go#L3422-L3425`

`asset/zec/zec.go#L3432-L3434`

Synopsis

Due to silent errors when checking whether `txHistoryDB == nil` or the wallet synchronization status during the syncing process, the syncing function will exit without syncing (which is correct behavior). Due to the lack of synchronization and a fast function return without an appropriate error message, the user could be led to incorrectly thinking the syncing process exited as it should.

Mitigation

We recommend adding error messages and logging to the above-mentioned parts of `syncTxHistory`.

Status

The Decred DEX team has updated the error message regarding an empty transaction history database.

Verification

Partially Resolved.

Suggestion 2: Increase blockQueryBuffer

Location

`client/asset/zec/zec.go#L3349`

Synopsis

Due to Zcash's short block times, increasing the buffer would help maintain correctness more efficiently.

Mitigation

We recommend increasing `blockQueryBuffer`.

Status

The Decred DEX team has updated the buffer from 3 to `100`.

Verification

Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.