BLOCKAPEX

SMART CONTRACT SECURITY

V 1.0

DATE: 22nd MAR 2024

PREPARED FOR: ELEKTRIK

About BlockApex

Founded in early 2021, is a security-first blockchain consulting firm. We offer services in a wide range of areas including Audits for Smart Contracts, Blockchain Protocols, Tokenomics along with Invariant development (i.e., test-suite) and Decentralized Application Penetration Testing. With a dedicated team of over 40+ experts dispersed globally, BlockApex has contributed to enhancing the security of essential software components utilized by many users worldwide, including vital systems and technologies.

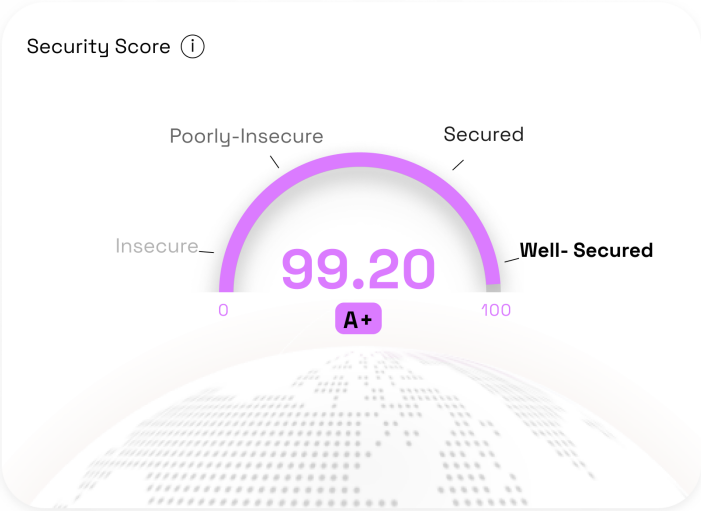
BlockApex has a focus on blockchain security, maintaining an expertise hub to navigate this dynamic field. We actively contribute to security research and openly share our findings with the community. Our work is available for review at our public repository, showcasing audit reports and insights into our innovative practices.

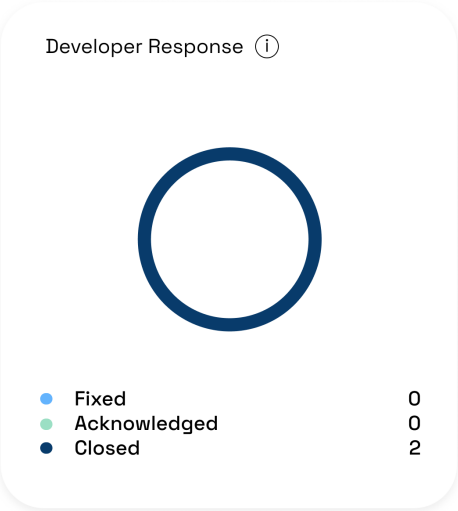
To stay informed about BlockApex's latest developments, breakthroughs, and services, we invite you to follow us on Twitter and explore our GitHub. For direct inquiries, partnership opportunities, or to learn more about how BlockApex can assist your organization in achieving its security objectives, please visit our Contact page at our website , or reach out to us via email at hello@blockapex.io.

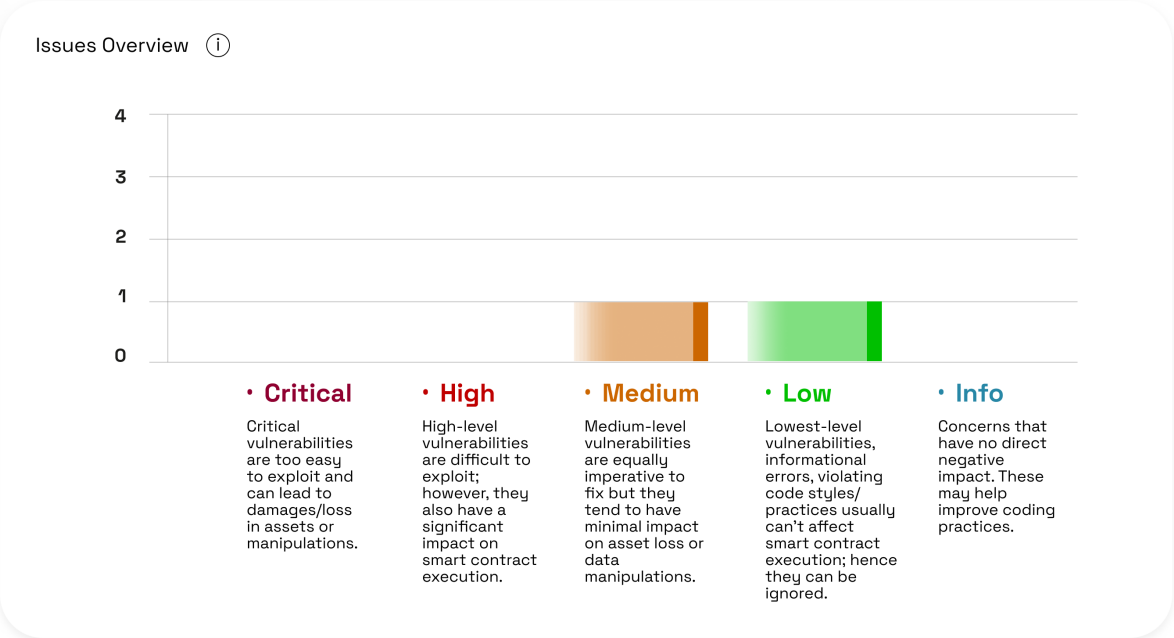
Contents

1 Executive Summary

The Elektrik Protocol features two primary components designed to enhance decentralized finance operations: the Advanced Order Engine and Liquidity Migrations. The Advanced Order Engine allows for complex trading through multiple order types with robust security features, while Liquidity Migrations facilitate secure and efficient cross-chain asset transfers. These tools provide comprehensive support for a variety of trading strategies and liquidity management needs in the DeFi space. Elektrik also features staking as a service where a Curve-style staking platform offers rewards for $ELTK and $veELTK stakers.







1.1 Scope

1.1.1 In Scope

The smart contract audit focused on three modules offered by Elektrik Protocol including Staking, Liquidity Migration and the Order Book.

- Liquidity providers earn ELTK tokens by staking their liquidity in pools, with emissions distributed based on weekly votes by token holders. ELTK holders can lock their tokens to participate in voting, influencing the allocation of emissions among the pools.
- Liquidity Migrations enable seamless liquidity transfers across blockchain networks to maintain robust liquidity levels in DeFi ecosystems. Features include adjustable migration parameters and stringent whitelisting of pools and position managers to ensure secure and flexible cross-chain transactions.
- The Order Book efficiently handles diverse and complex order types with robust security features like EIP-712 integration, advanced order processing, and customizable settings for fee management and token whitelisting, making it ideal for varied DeFi applications.

Contracts in Scope:

1. **Staking**

- Repo: BlockApex/elektrik_staking_contracts
- Commit Hash: b79a829a0248980612c6682c819eb1c4908aa66b

2. **Liquidity Migration**

- Repo: BlockApex/elektrik_liquidity_migration
- Commit Hash: 169c53cf3ae110e546de0ba72cbc83d190b39326

3. **Elektrik Order Matching Engine - Elektrik V2**

- Repo: BlockApex/Elektrik-V2-Contracts
- Commit Hash: ab6c30f744168c04e5413604eb8006d0aed706b8

1.1.2 Out of Scope

All features or functionalities not delineated within the “In Scope” section of this document shall be deemed outside the review of this audit. This exclusion particularly applies to **the backend operations of the Elektrik V2’s Advanced Order Matching Engine** contracts associated with the platform & any other external libraries.

1.2 Methodology

The audit of the Elektrik Protocol was conducted over a course of 3 weeks, utilizing a mix of approaches involving automated analysis and manual code review by a team of two security researchers. The methodology began with a reconnaissance phase to gain an initial understanding of the protocol's various contracts structure and business intended functionalities. Following this, the security researchers engaged in a detailed, line-by-line examination of the code. This manual review process focused on identifying logical flaws, security vulnerabilities, and opportunities for optimization, while ensuring adherence to Solidity best practices, security design patterns, and coding standards for clarity and efficiency.

1.3 Project Goals

The audit aimed to conduct a comprehensive security assessment of the Elektrik Protocol, focusing on its key components—the Advanced Order Engine, the staking module and the Liquidity Migrations feature. Specifically, the audit sought to address the following questions:

1. Are there any vulnerabilities in the order processing mechanisms of the Advanced Order Engine, such as susceptibility to reentrancy or price manipulation attacks?
2. Does the Liquidity Migrations feature contain any vulnerabilities, particularly in its handling of cross-chain transactions or liquidity pool whitelisting?
3. Is there any risk associated with the fee management system, including potential for incorrect fee calculations or mishandling of fee transfers?
4. Can any unauthorized changes be made to critical contract settings such as operator privileges, token whitelisting, or other permission-based actions?
5. How does the protocol ensure consistency of state across its multiple functionalities, especially after complex operations such as partial order fills or migration completions?
6. Is the smart contract complex vulnerable to attacks that could exploit its dependency on block.timestamp for functionality such as order expiration or liquidity migration timing?

1.4 Status Descriptions

Acknowledged: The issue has been recognized and is under review. It indicates that the relevant team is aware of the problem and is actively considering the next steps or solutions.

Fixed: The issue has been addressed and resolved. Necessary actions or corrections have been implemented to eliminate the vulnerability or problem.

Closed: This status signifies that the issue has been thoroughly evaluated and acknowledged by the development team. While no immediate action is being taken.

1.5 Summary of Findings Identified

S.No	Severity	Findings	Status
#1	MEDIUM	Design Flaw in Order Book for Fetching Price	CLOSED
#2	LOW	Insufficient Input Sanitization	CLOSED

2 Findings and Risk Analysis

2.1 Design Flaw in Order Book for Fetching Price

Severity: Medium

Status: Closed

Location :

AdvancedOrderEngine.sol#L311-L368

Description The existing logic within the `fillOrders` function of the Order Matching Engine directly utilizes the initially provided executedSellAmounts and `executedBuyAmounts` for final order settlements. This approach does not account for adjustments due to partial fills or actual liquidity conditions encountered during order processing. As a result, the final amounts transferred may not accurately reflect the transactions executed, leading to discrepancies between the anticipated and actual exchange of tokens.

Impact This design flaw can result in financial discrepancies where either too many or too few tokens are transferred compared to what is intended by the executed orders. It can lead to market inefficiencies, potential losses for users due to unanticipated slippage, and a decrease in platform trustworthiness.

Recommendation Revise the `fillOrders` function to include dynamic tracking of the executed sell and buy amounts throughout the order processing phase. This modification should account for any partial fills and the actual liquidity utilized. The adjusted amounts should then be used when invoking `_processFinalizeOrder` to ensure that the amounts transferred correspond to the actual settlements. Implementing these changes will align the final token transfers with the executed order details, enhancing the accuracy and reliability of trade executions within the platform.

Developer Response

The development team has indicated that their current implementation is designed to prioritize system invariants, suggesting that they believe their method provides the best balance of performance and integrity.

Auditor Response

Acknowedged with no further refute.

2.2 Insufficient Input Sanitization

Severity: Low

Status: Closed

Location :

1. Replicator.sol#L28
2. Migrator.sol#L49-L51

Description During the audit, it was noted that the `Replicator.replicate()` function lacks necessary checks to ensure that `params.amount0Min` and `params.amount1Min` are less than or equal to `params.amount0Desired` and `params.amount1Desired`.

This oversight can lead to a situation where the slippage check fails;

```
(require(amount0 &gt;= params.amount0Min &amp;&amp; amount1 &gt;= params.amount1Min, &#x27;Price slippage check&#x27;);)
```

after significant computational effort has already been expended, leading to inefficient gas usage and potential transaction failures. A similar lack of input validation was observed in the `Migrator.migrate()` function, potentially allowing for the processing of erroneous or malicious input data.

Impact The lack of rigorous input validation can lead to unnecessary computational expenses, inefficient gas usage, and increased potential for transaction failures due to unmet slippage conditions.

Recommendation Implement rigorous input validation checks at the beginning of the `Replicator.replicate()` and `Migrator.migrate()` functions to ensure that minimum desired amounts (`params.amount0Min` and `params.amount1Min`) do not exceed the desired amounts (`params.amount0Desired` and `params.amount1Desired`). This will prevent unnecessary computations and potential transaction failures due to failed slippage checks. Additionally, consider adding safeguards against potentially spammy or malicious inputs to enhance overall contract security and efficiency.

Disclaimer:

The smart contracts provided by the client with the purpose of security review have been thoroughly analyzed in compliance with the industrial best practices till date w.r.t. Smart Contract Weakness Classification (SWC) and Cybersecurity Vulnerabilities in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token (if any), its sale, or any other aspect of the project that contributes to the protocol's public marketing.

Crypto assets/ tokens are the results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the reports in any way, including to make any decisions to buy or sell any token, product, service, or asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the programmable code and only the programmable code, we note, as being within the scope of our review within this report. The smart contract programming language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond the programming language's compiler scope that could present security risks.

This security review cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While BlockApex has done their best in conducting the analysis and producing this report, it is important to note that one should not rely on this report only - we recommend proceeding with several independent code security reviews and a public bug bounty program to ensure the security of smart contracts.