BLOCKAPEX

SMART CONTRACT SECURITY

v 1.0

Date: 07-07-2025

Prepared For: SONEX MAX DEX - FINAL AUDIT REPORT

About BlockApex

Founded in early 2021 BlockApex 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.

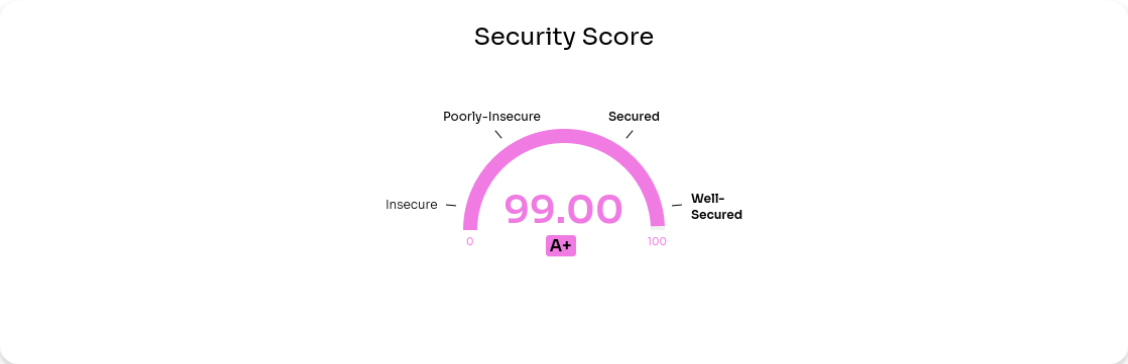
Contents

1 Executive Summary **4**
1.1 Scope . . . 5
1.1.1 In Scope . . . 5
1.1.2 Out of Scope . . . 5
1.2 Methodology . . . 5
1.3 Status Descriptions . . . 7
1.4 Summary of Findings Identified . . . 8

2 Findings and Risk Analysis **9**
2.1 Duplicate Reward-Token Initialization Locks All Reward Slots . . . 9
2.2 Use of disable_forever During Pause May Unintentionally Lock Protocol Permanently . 10
2.3 protocol Fee should be in bounds . . . 11
2.4 Missing Events in administrative functions . . . 12
2.5 Sanity check missing during update_reward_manager . . . 13
2.6 emergency_admin Function Returns Incorrect Address . . . 14

1 Executive Summary

Our team conducted a Filtered Audit, in which two individuals independently reviewed the Sonex Max Dex smart contracts. This process involved thorough and rigorous manual testing, including a line-by-line code review to identify potential bugs. All findings were carefully verified and re-tested to ensure their validity.



1.1 Scope

1.1.1 In Scope

1. **Repository:** https://github.com/max-mov-dex/max-contracts
2. **Initial Commit Hash:** c0287cc20be7f20e48ddfd5618debc4740e45e76
3. **Fixed Git Repo:** https://github.com/max-mov-dex/max-contracts/pull/4/files

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 purview of this audit.

1.2 Methodology

The codebase was audited using a filtered audit technique. A band of two (2) auditors scanned the codebase in an iterative process for a time spanning 3 weeks. Starting with the recon phase, a basic understanding was developed, and the auditors worked on developing presumptions for the developed codebase and the relevant documentation/whitepaper. Furthermore, the audit moved on with the manual code reviews to find logical flaws in the codebase complemented with code optimizations,software, and security design patterns, code styles and best practices

Project Overview

The MaxDex Protocol is a decentralized exchange (DEX) infrastructure built on the Aptos blockchain. The protocol facilitates liquidity provision, token swapping, and incentivized rewards through a modular smart contract architecture.

Key Components:

- **Liquidity Pool Management:** Supports creation and management of token pairs with configurable fees and a concentrated liquidity model based on tick ranges and price bands.
- **Token Swapping:** Provides a robust and efficient swapping mechanism between supported tokens. It dynamically calculates price changes based on current ticks and available liquidity, ensuring precise execution and correct fee accounting for each trade.
- **Reward System:** Enables token-based rewards for liquidity providers through a whitelisted reward index system, allowing multiple reward tokens per pool.
- **Emergency Controls:** Includes an emergency module to pause, resume, or permanently disable protocol operations during critical events, ensuring safety and governance flexibility.
- **Fee Tier Configuration:** Allows dynamic configuration of fee tiers and their corresponding tick spacing, enabling market segmentation and optimization of trading activity.
- **Access Control:** Utilizes Aptos-native security primitives, such as resource accounts, signer capabilities, and object-based storage, to enforce secure, and flexible contract management.

Centralization & Risk Analysis

Administrative Roles and Centralised Controls

- MaxDex operates with multiple admin roles, each holding authority over different protocol functions. The PoolAdmin configures fee tiers and tick spacings that affect how trades are priced. While pool creation itself is permissionless, the available configurations (like fee levels) are governed centrally, allowing admins to influence market dynamics indirectly.
- The RewardAdmin manages the incentive system, including whitelisting reward tokens, setting emissions rates, and funding reward pools. Since rewards can only be distributed from whitelisted tokens, this role effectively controls who gets incentivized and how much a powerful lever that, if abused, can distort liquidity flow or unfairly favor specific assets.
- The EmergencyAdmin has the ability to pause or permanently disable key operations like swaps, reward emissions, or even entire modules. This central authority is crucial for reacting to bugs or exploits but also represents a single point of failure. If compromised or misused, it could bring all trading and reward activity to a halt.

1.3 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.

1.4 Summary of Findings Identified

S.NO	SEVERITY	FINDINGS	STATUS
#1	LOW	Duplicate Reward-Token Initialization Locks All Reward Slots	FIXED
#2	LOW	Use of disable_forever During Pause May Unintentionally Lock Protocol Permanently	ACKNOWLEDGED
#3	INFO	protocol Fee should be in bounds	FIXED
#4	INFO	Missing Events in administrative functions	FIXED
#5	INFO	Sanity check missing during update_reward_manager	FIXED
#6	INFO	emergency_admin Function Returns Incorrect Address	FIXED

2 Findings and Risk Analysis

2.1 Duplicate Reward-Token Initialization Locks All Reward Slots

Severity: Low

Status: Fixed

Location:

1. `sources/reward_manager.move`
2. `sources/liquidity_pool.move`

Description: `initialize_reward()` lets an admin push a new entry into `pool.reward_infos` whenever the array length < 3. It does not check whether the supplied `token_metadata` is already present in an existing slot. As a result, the same token can be initialized up to three times, exhausting the pool's maximum of three reward slots. Because the contract provides no function to delete or replace a reward entry, the pool becomes permanently unable to add any new (distinct) reward tokens.

Impact:

1. **Denial-of-service:** further reward programs cannot be added once the three slots are filled even accidentally with the same token.
2. **Governance/UX disruption:** the pool owner must redeploy a new pool to correct the mistake, confusing LPs and fragmenting liquidity.
3. **Potential accounting anomalies:** three identical reward streams may be displayed or tracked separately.

Recommendation: 1. Add a uniqueness check before push_back 2. Optionally provide an admin-only `remove_reward(index)` or `replace_reward(index, new_token)` to allow recovery from mistakes.

2.2 Use of disable_forever During Pause May Unintentionally Lock Protocol Permanently

Severity: Low

Status: Acknowledged

Location:

1. `sources/emergency.move`

Description: The `disable_forever` function can be invoked even when the protocol is in a paused state. While this function is intended for permanently disabling the emergency module, it currently disables the entire protocol irreversibly when called during a temporary pause. This could lead to unintended consequences, including the permanent disabling of the protocol when only a temporary pause was meant. Additionally, this function is intended to be callable only by the emergency admin, who is considered a trusted party.

Proof of Concept:

```
#[test(max_dex = @max_dex, emer = @emergency_admin)]
fun test_disable_during_pause(max_dex: &amp;signer, emer: &amp;signer) acquires
    EmergencyAccountCap, IsEmergency {
    config::initialize_for_test(max_dex);
    init_module(max_dex);
    pause(emer);               // Pauses the protocol temporarily
    disable_forever(emer);     // Permanently disables protocol even though only a pause
        was intended
    resume(emer);              // This call will no longer work
}
```

Recommendation: Add a condition to prevent disable_forever from being called while paused.

```
public entry fun disable_forever(account: &amp;signer) acquires EmergencyAccountCap {
    assert_no_emergency(); // Prevent permanent disable while paused
    ...
}
```

2.3 protocol Fee should be in bounds

Severity: Info

Status: Fixed

Location:

1. `sources/config.move`

Description: Admin can set the protocol fees by calling `set_protocol_fee` , in the current implementation there is no bounds apply to the `protocol_fee_rate` , an admin can set it upto `FEE_SCALE`. There should be a proper bound like 1-5% in which admin should set the fees.

Recommendation: enforcing a sensible upper bound in `set_protocol_fee` to prevent excessive or accidental fee hikes.

2.4 Missing Events in administrative functions

Severity: Info

Status: Fixed

Location:

1. `sources/config.move`

Description: The config.move module contains multiple state-changing administrative functions that do not emit events. This creates several issues: - Reduced on-chain transparency for critical parameter changes - Difficulty for indexers and UIs to track configuration changes - Missing audit trail for administrative actions - Limited historical querying capabilities

Affected Functions:

- set_pool_admin
- set_reward_admin
- set_emergency_admin
- set_protocol_fee
- set_trader_fee_multipliers

Recommendation: Add Emit Events for the above mentioned affected functions

2.5 Sanity check missing during update_reward_manager

Severity: Info

Status: Fixed

Location:

1. `sources/liquidity_pool.move`

Description: The `update_reward_manager()` function in the liquidity pool module allows updating the manager of a reward. However, it lacks a basic sanity check to verify that the new manager address is different from the current manager. This oversight permits redundant state updates that consume gas and blockchain resources unnecessarily.

Recommendation: Add a simple `assert!(new_manager != reward_info.manager, ...)` in `update_reward_manager` to block no-op updates and save gas.

2.6 emergency_admin Function Returns Incorrect Address

Severity: Info

Status: Fixed

Location:

1. `sources/config.move`

Description: The `emergency_admin()` view function currently returns the `pool_admin` address instead of the intended `emergency_admin`. This may cause confusion for users or tools expecting the actual emergency admin.

Recommendation: Update the function to correctly return the emergency_admin field from the Config resource.

```
#[view]
public fun emergency_admin(): address acquires Config {
    borrow_global&lt;Config&gt;(@max_dex).emergency_admin
}
```

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