AUDITMOS

Secure your space

SPARTADEX

audit / code review report

July 10, 2023

TABLE OF CONTENTS

contact@auditmos.com

AUDITMOS

Secure your space

SPARTADEX

audit / code review report

July 10, 2023

LICENSE

contact@auditmos.com

AUDITMOS

Secure your space

SPARTADEX

audit / code review report

July 10, 2023

DISCLAIMER

contact@auditmos.com

AUDITMOS

Secure your space

SPARTADEX

audit / code review report

July 10, 2023

APPROACH AND METHODOLOGY

PURPOSE

1. Determine the correct operation of the protocol, according to the design specification.
2. Identify possible vulnerabilities that could be exploited by an attacker.
3. Detect errors in the smart contract that could lead to unexpected behavior.
4. Analyze whether best practices were followed during development.
5. Make recommendations to improve security and code readability.

CODEBASE

Repository	https://github.com/SpartaDEX/sdex-smart-contracts
Branch	main
Commit hash	0369f48e3b2a655616dc75afdf87b9915f64d720

METHODOLOGY

1. Reading the available documentation and understanding the code.
2. Doing automated code analysis and reviewing dependencies.
3. Checking manually source code line by line for security vulnerabilities.
4. Following guildlines and recommendations.
5. Preparing this report.

contact@auditmos.com

AUDITMOS
Secure your space

SPARTADEX

audit report

July 10, 2023

DESCRIPTION

Issues Categories:

Severity	Description
CRITICAL	vulnerability that can lead to loss of funds, failure to recover blocked funds, or catastrophic denial of service.
HIGH	vulnerability that can lead to incorrect contract state or unpredictable operation of the contract.
MEDIUM	failure to adhere to best practices, incorrect usage of primitives, without major impact on security.
LOW	recommendations or potential optimizations which can lead to better user experience or readability.

Each issue can be in the following state:

State	Description
PENDING	still waiting for resolving
ACKNOWLEDGED	know but not planned to resolve for some reasons
RESOLVED	fixed and deployed

Auditor

bytes032

https://bytes032.xyz/

AUDITMOS

Secure your space

SPARTADEX

audit report

July 10, 2023

FINDINGS

Finding	Severity	Status
#1 - A malicious user can drain all the funds from the TokenVesting contract	CRITICAL	RESOLVED
#2 - _getLPTokenPrice can be manipulated through a flash loan	CRITICAL	RESOLVED
#3 - An adversary can lock user funds by spamming vesting schedules	CRITICAL	RESOLVED
#4 - Excessive user allocations can DoS the calculateTotalReward function and deprive the user from rewards	CRITICAL	RESOLVED
#5 - Unclaimed tokens are locked forever in the Airdrop.sol contract	HIGH	RESOLVED
#6 - addTargetLiquidity and removeSourceLiquidity` has no slippage protection, which can result in in loss of funds	HIGH	RESOLVED
#7 - LockdropPhase2 lacks token recovery mechanism	HIGH	RESOLVED
#8 - exchangeTokens lacks slippage control	HIGH	RESOLVED
#9 - addLiquidity and removeLiquidity is called without expiration	HIGH	RESOLVED
#10 - Potential mismatch between claimable amounts and actual token balance	MEDIUM	RESOLVED

AUDITMOS

Secure your space

SPARTADEX

audit report

July 10, 2023

FINDINGS

Finding	Severity	Status
#11 - Potential manipulation of airdrop claimable amounts in setClaimableAmounts function	MEDIUM	RESOLVED
#12 - Duplicate locking expiration timestamps can dillute the rewards	MEDIUM	RESOLVED
#13 - Missing sanity checks for _claimStartTimestamp in Airdrop contract	MINOR	RESOLVED
#14 - EPossibility for division by zero in LockdropPhase1	MINOR	RESOLVED
#15 - LockdropPhase2 is prone to reentrancy	MINOR	RESOLVED
#16 - Violation of the Check-Effects-Interaction pattern in getRewardAndSendOnVesting	MINOR	RESOLVED
#17 - Potential for Cross-Chain Replay attacks due to hard-coded Domain Separator	MINOR	RESOLVED
#18 - Inconsistency in 'LiquidityProvided' event parameters	MINOR	RESOLVED

contact@auditmos.com

AUDITMOS

Secure your space

SPARTADEX

audit report

July 10, 2023

#1 - A MALICIOUS USER CAN DRAIN ALL THE FUNDS FROM THE TOKENVESTING CONTRACT

Severity	Status
CRITICAL	RESOLVED

This vulnerability allows a malicious actor to drain the entire TokenVesting contract of its tokens prematurely. The issue arises from the lack of a proper check for whether the vesting schedule duration has elapsed.

RECOMMENDATION

To fix this vulnerability, a check should be added to ensure the elapsed time does not exceed the vesting duration before calculating the vested amount. This can be done by adding a conditional statement to verify if block.timestamp is greater than vestingSchedule.endTime.

contact@auditmos.com

SPARTADEX

audit report

July 10, 2023

#2 - GETLPTOKENPRICE CAN BE MANIPULATED THROUGH A FLASH LOAN

Severity	Status
CRITICAL	RESOLVED

This vulnerability allows for manipulation of the Total Value Locked (TVL) and Liquidity Provider (LP) token price. If exploited, it could lead to incorrect pricing of LP tokens and potentially significant financial loss to users and the platform itself. It could also damage the platform's reputation and user trust.

RECOMMENDATION

To mitigate this vulnerability, the following recommendations are proposed:

1. Update the getLPTokenPrice algorithm to derive ideal reserve values from K and the underlying prices
2. Implement safeguards against rapid and large changes in the TVL that could be indicative of manipulative actions.
3. Consider introducing limits or controls on the amount of tokens a user can add to the pool at a given time.
4.

Additionaly, you can take inspiration from this UniswapV2Oracle by Alpha Homora.

SPARTADEX

audit report

July 10, 2023

#3 - AN ADVERSARY CAN LOCK USER FUNDS BY SPAMMING VESTING SCHEDULES

Severity	Status
CRITICAL	RESOLVED

The way the claim function is currently implemented, it iterates over all vesting schedules of a user, regardless of the scheduleIds specified. This could potentially lead to a Denial of Service (DoS) attack if a malicious actor spams the schedules of an arbitrary user with a large number of vesting schedules. As a result, the targeted user's claim function call will run out of gas, effectively preventing them from claiming any of their vested tokens.

RECOMMENDATION

To mitigate this potential issue, the claim function should be refactored to loop over only the scheduleIds specified by the user, rather than all vesting schedules associated with the user. This can be achieved by changing the scheduleIdsLength variable to be the length of the scheduleIds array, rather than the length of the user's schedules array.

SPARTADEX

audit report

July 10, 2023

#4 - EXCESSIVE USER ALLOCATIONS CAN DOS THE CALCULATETOTALREWARD FUNCTION AND DEPRIVE THE USER FROM REWARDS

Severity	Status
CRITICAL	RESOLVED

The presence of this vulnerability may discourage users from making numerous allocations, as it could potentially leave them unable to access their rewards. This issue stems from a scenario where a user has a significant number of allocations, leading to the exhaustion of gas in the calculateTotalReward function.

RECOMMENDATION

The calculateTotalReward function should be refactored to handle a large number of allocations without exceeding the gas limit. One approach could be to design the function so it can process allocations in ranges. This would allow it to handle a large number of allocations over multiple transactions, rather than attempting to process all allocations in a single transaction.

SPARTADEX

audit report

July 10, 2023

#5 - UNCLAIMED TOKENS ARE LOCKED FOREVER IN THE AIRDROP.SOL CONTRACT

Severity	Status
HIGH	RESOLVED

As a result of this issue, tokens that are not claimed during the designated timeframe, or "dust" tokens left over due to small discrepancies in calculations, remain stuck in the Airdrop.sol contract indefinitely. This means these tokens are effectively lost, decreasing the total available supply and potentially distorting the token's market dynamics.

RECOMMENDATION

Implement a token recovery function that can be executed only by onlyAirdropManagerRole, but only after the airdrop period ends.
This will allow addresses which have the onlyAirdropManagerRole to recover excess tokens from the contract.

SPARTADEX

audit report

July 10, 2023

#6 - ADDTARGETLIQUIDITY AND REMOVESOURCELIQUIDITY HAS NO SLIPPAGE PROTECTION, WHICH CAN RESULT IN IN LOSS OF FUNDS

Severity	Status
HIGH	RESOLVED

The current implementation of the removeSourceLiquidity() function in the smart contract lacks slippage protection. This omission might expose users to potential price manipulation, where malicious actors could intentionally cause price slippage, leading to significant losses when users remove liquidity. As a result, users could lose a significant portion of their funds when they remove liquidity from the pool, which could erode trust in the smart contract.

RECOMMENDATION

To mitigate the risk of losing funds due to price slippage, it's recommended to add slippage protection to both functions. This protection can be added by specifying the minimum amounts of tokens (amountAMin and amountBMin) that should be received when removing liquidity and pass them to router.removeLiquidity and router.addLiquidity respectively.

SPARTADEX

audit report

July 10, 2023

#7 - LOCKDROPPHASE2 LACKS TOKEN RECOVERY

Severity	Status
HIGH	RESOLVED

MECHANISM

The lack of a proper token recovery mechanism in the LockdropPhase2 contract for unclaimed tokens presents a significant financial risk. In the current setup, any unclaimed "sparta" and "stable" tokens become irretrievable,

RECOMMENDATION

Implement a token recovery function that can be executed only after a certain state and only by the owner of the contract.
This will allow the owners of the protocol to recover excess tokens from the contract.

SPARTADEX

audit report

July 10, 2023

#8 - EXCHANGETOKENS LACKS SLIPPAGE CONTROL

Severity	Status
HIGH	RESOLVED

Without slippage protection, there's a significant risk of unfavorable trade outcomes. The lack of guarantee that spartaTotalLocked will equal the stable coins required can lead to substantial financial loss. This becomes particularly concerning during periods of high market volatility.

RECOMMENDATION

To mitigate the risk of losing funds due to price slippage, it's recommended to add slippage protection to both functions. This protection can be added by specifying the minimum amounts of tokens (amountAMin and amountBMin) that should be received.

SPARTADEX

audit report

July 10, 2023

#9 - ADDLIQUIDITY AND REMOVELIQUIDITY IS CALLED WITHOUT EXPIRATION

Severity	Status
HIGH	RESOLVED

The current implementation can lead to significant risks including unfavorable trade outcomes and potential financial loss. Without a user-specified deadline, transactions can remain in the mempool for an extended period, resulting in execution at a potentially disadvantageous time. Moreover, by setting the deadline to block.timestamp, a validator can hold the transaction without any time constraints, further exposing users to the risk of price fluctuations.

RECOMMENDATION

To mitigate this issue, it's recommended to add deadline parameters to all functions interacting with AMMs and allow users to specify their preferred deadlines. These user-specified deadlines should then be passed on to the respective AMM function calls. This modification will give users more control over their transactions, reducing the associated risks.

AUDITMOS

Secure your space

SPARTADEX

audit report

July 10, 2023

#10 - POTENTIAL MISMATCH BETWEEN CLAIMABLE AMOUNTS AND ACTUAL TOKEN BALANCE

Severity	Status
MEDIUM	RESOLVED

The setClaimableAmounts function could potentially allow the administrators to set claimable token amounts for users that exceed the actual token balance of the contract. This could lead to a situation where users are unable to claim their tokens despite being told they have a certain amount available. This discrepancy can undermine trust in the contract, the airdrop process, and the overall project.

RECOMMENDATION

It's recommended to add a check to ensure the total claimable amounts do not exceed the actual token balance of the contract. This can be done by calculating the sum of all claimable amounts and comparing it to the contract's token balance.

contact@auditmos.com

AUDITMOS

Secure your space

SPARTADEX

audit report

July 10, 2023

#11 - POTENTIAL MANIPULATION OF AIRDROP CLAIMABLE AMOUNTS IN SETCLAIMABLEAMOUNTS FUNCTION

Severity	Status
MEDIUM	RESOLVED

The setClaimableAmounts function sets the amount of tokens that specific users are eligible to claim from an airdrop. The function is only accessible by addresses with the onlyAirdropManagerRole. This function takes in two arrays, users and amounts, representing the addresses of the users and the respective amounts they can claim.

RECOMMENDATION

Use += instead of =

contact@auditmos.com

AUDITMOS

Secure your space

SPARTADEX

audit report

July 10, 2023

#12 - DUPLICATE LOCKING EXPIRATION TIMESTAMPS CAN DILLUTE THE REWARDS

Severity	Status
MEDIUM	RESOLVED

The current implementation of the calculateRewardRates() function in the LockdropPhase1.sol smart contract has a potential vulnerability where rewards might get diluted. This dilution could occur if there are any duplicate locking expiration timestamps, which could lead to a duration of zero during reward calculation. As a result, users may receive less reward than they are supposed to, impacting the fairness and trustworthiness of the reward distribution mechanism.

RECOMMENDATION

It is recommended to add a check during the constructor time to ensure that the lockingExpirationTimestamps array doesn't contain any duplicate values.

contact@auditmos.com

AUDITMOS
Secure your space

SPARTADEX

audit report

July 10, 2023

#13 - MISSING SANITY CHECKS FOR _CLAIMSTARTTIMESTAMP IN AIRDROP CONTRACT

Severity	Status
MINOR	RESOLVED

The absence of sanity checks for the _claimStartTimestamp parameter in the Airdrop contract can lead to potential vulnerabilities and unexpected behavior. It may allow the contract to be initialized with a claim start timestamp that is not in the future, which could undermine the intended functionality and fairness of the airdrop.

RECOMMENDATION

To mitigate this vulnerability, it is crucial to implement sanity checks and ensure that the _claimStartTimestamp value provided in the constructor is in the future. For example, you can add a validation check within the constructor to ensure that _claimStartTimestamp is greater than the current block timestamp.
This can be done using a condition such as require(_claimStartTimestamp > block.timestamp, "Claim start time must be in the future");.

contact@auditmos.com

AUDITMOS

Secure your space

SPARTADEX

audit report

July 10, 2023

#14 - POSSIBILITY FOR DIVISION BY ZERO IN LOCKDROPPHASE1

Severity	Status
MINOR	RESOLVED

The vulnerability in the LockdropPhase1.sol contract can potentially lead to a division by zero error. This error can cause the function execution to revert, resulting in unexpected behaviour or denial of service for users interacting with the contract.

RECOMMENDATION

To address this, make sure to sanitize the lockingExpirationTimestamps for erronous values before calculating the reward rate.

contact@auditmos.com

SPARTADEX

audit report

July 10, 2023

#15 - LOCKDROPPHASE2 IS PRONE TO REENTRANCY

Severity	Status
MINOR	RESOLVED

The code block performs a token transfer before updating the internal state (stableTotalLocked, walletStableLocked[msg.sender]). This violates the check-effects-interaction pattern, which is a best practice in smart contract development. If the token used is an ERC20 compatible ERC777 or has similar re-entrancy capabilities, the receiver of the transfer call could re-enter the contract and interact with it while its internal state is not yet updated, leading to to unlimited withdrawals.

RECOMMENDATION

To respect the check-effects-interaction pattern, the internal state should be updated before interacting with external contracts (i.e., before calling the transfer function).

SPARTADEX

audit report

July 10, 2023

#16 - LOCKDROPPHASE2 IS PRONE TO REENTRANCY

Severity	Status
MINOR	RESOLVED

This vulnerability can lead to potential unlimited claims of the reward if the reward token adheres to the ERC777 standard, which is backward compatible with ERC20. Although it's currently minor as the reward token is supposedly the sparta token, it can be a major issue if any other tokens are used in the future.
The getRewardAndSendOnVesting function is designed to distribute rewards to a user in a two-fold manner: immediate transfer and vesting. The function first validates if the second phase of locking has ended, then calculates the total reward for the user, divides it in half, and sends one half immediately while the other half is designated for vesting.
The issue is that it does not follow the Check-Effects-Interaction (CEI) pattern. Specifically, it attempts to transfer tokens before updating the critical userRewardWithdrawn state variable. This sequence can potentially allow a reentrancy attack, exploiting the token's hooks to call getRewardAndSendOnVesting again before the state variable is updated.

RECOMMENDATION

The solution to this vulnerability is to follow the Check-Effects-Interaction pattern by updating the state (`userRewardWithdrawn[msg.sender] = reward`) before attempting to transfer the tokens.

AUDITMOS

Secure your space

SPARTADEX

audit report

July 10, 2023

#17 - POTENTIAL FOR CROSS-CHAIN REPLAY ATTACKS DUE TO HARD-CODED DOMAIN SEPARATOR

Severity	Status
MINOR	RESOLVED

This vulnerability could lead to unauthorized transactions on a new chain after a split, posing significant security risks to users. In the worst case, assets could be unintentionally transferred or manipulated on the unintended chain.

RECOMMENDATION

To mitigate this vulnerability, it is recommended that the chain ID be computed dynamically rather than being hard-coded into the `DOMAIN_SEPARATOR` during initialization. This can be done using the `chainid()` function within the EVM

contact@auditmos.com

AUDITMOS

Secure your space

SPARTADEX

audit report

July 10, 2023

#18 - POTENTIAL FOR CROSS-CHAIN REPLAY ATTACKS DUE TO HARD-CODED DOMAIN SEPARATOR

Severity	Status
MINOR	RESOLVED

The inconsistent usage of event parameters in the LiquidityProvided event can lead to confusion and incorrect data representation

RECOMMENDATION

1. Update the event declaration in the ILockdropPhase1 contract to match the usage in the LockdropPhase1 contract. If the last parameter represents the _value, revise the event declaration to reflect this.
2. Alternatively, if the intended usage is to represent the duration, modify the event emission in the LockdropPhase1 contract to include the correct duration value instead of _value.

contact@auditmos.com

auditmos.com

AUDITMOS

Secure your space

contact@auditmos.com