AUDITMOS

Secure your space

SPARTADEX

audit / code review report

April 10, 2024

TABLE OF CONTENTS

contact@auditmos.com

AUDITMOS

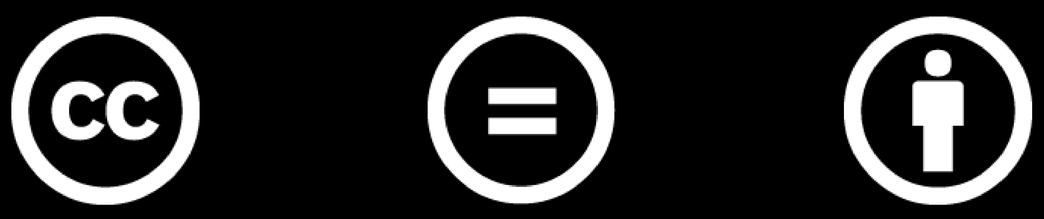
Secure your space

SPARTADEX

audit / code review report

April 10, 2024

LICENSE



contact@auditmos.com

AUDITMOS

Secure your space

SPARTADEX

audit / code review report

April 10, 2024

DISCLAIMER

contact@auditmos.com

AUDITMOS

Secure your space

SPARTADEX

audit / code review report

April 10, 2024

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-evm-contracts/
Branch	develop
Commit hash	2ce885836e7b53fe530756cb74ebefa0a5119aec

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 / code review report

April 10, 2024

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

contact@auditmos.com

AUDITMOS

Secure your space

SPARTADEX

audit / code review report

April 10, 2024

AUDIT SCOPE

1. getting to know the project	✓
2. research into architecture	✓
3. manual code read	✓
4. permissions of state changing functions	✓
5. identify common Solidity vulnerabilities	✓
6. test coverage	✓
7. static analysis	✓
8. storage key overlaps	✓
9. DOS possibilities by malicious attacker	✓
10. steal funds possibilities	✓

contact@auditmos.com

AUDITMOS

Secure your space

SPARTADEX

audit / code review report

April 10, 2024

FINDINGS

Finding	Severity	Status
#1 - It is possible to instantiate WithFees with zero values	MEDIUM	RESOLVED
#2 - Lack of proper validation in ContractRepository	MEDIUM	RESOLVED
#3 - Lack of proper validation in Staking contracts	MEDIUM	RESOLVED

contact@auditmos.com

SPARTADEX

audit / code review report

April 10, 2024

#1 - IT IS POSSIBLE TO INSTANTIATE WITHFEES WITH ZERO VALUES

Severity	Status
MEDIUM	RESOLVED

Allowing instantiation of a smart contract with a 0x0 address and a value of 0 for implemented fees poses significant security risks and can lead to potential vulnerabilities.

1. Invalid Address: Allowing the instantiation of a smart contract with a 0x0 address (or address(0)) means that the contract may not have a legitimate destination for collected fees or transactions. This can result in loss of funds or render the contract unusable.
2. Zero Value for Fees: Implementing fees with a value of 0 means that transactions can be executed without any fees, potentially leading to spam attacks or abuse of the system. Fees are typically used to cover the cost of processing transactions or to incentivize miners to include transactions in the blockchain. Without appropriate fees, the system may become vulnerable to abuse and congestion.

RECOMMENDATION

To fix these issues, you can implement checks in the constructor to ensure that valid addresses and non-zero values are provided for the treasury and fees. Here's how you can do it:

```
constructor(IAccessControl acl_, address treasury_, uint256 value_) {
    require(acl_ != address(0), "AccessControl address cannot be 0x0");
    require(treasury_ != address(0), "Treasury address cannot be 0x0");
    require(value_ > 0, "Fees value must be greater than 0");

    acl = acl_;
    treasury = treasury_;
    fees = value_;
}
```

or use ZeroAddressGuard and ZeroAmountGuard

#2 - LACK OF PROPER VALIDATION IN CONTRACTREPOSITORY

Severity	Status
MEDIUM	RESOLVED

There's a potential issue related to unchecked contract address assignment. While the function setContract is designed to set the address for a given contract ID, it lacks validation mechanisms to ensure that the provided contract address is legitimate. This can lead to various security risks, including the following:

1. **Incorrect or Malicious Contract Address**: Without proper validation, the **contractAddress** parameter could be set to any arbitrary address, including addresses of malicious contracts. This can lead to unintended behavior or exploitation of the system.
2. **Overwriting Existing Contracts**: The function allows overwriting the address associated with a contract ID without any checks. If an attacker gains control over the setContract function, they could potentially overwrite critical contracts with malicious ones, causing disruptions or financial losses.

RECOMMENDATION

To mitigate these risks, you should implement checks to ensure that the provided contract address is valid and meets certain criteria.

```
function setContract(
    bytes32 contractId,
    address contractAddress
) external override onlyRepositoryOwner {
    require(contractAddress != address(0), "Contract address cannot be 0x0");
    require(contractId != bytes32(0), "Contract ID cannot be empty");
    require(contractAddress != address(this), "Cannot set contract address to this contract");

    repository[contractId] = contractAddress;
}
```

or use ZeroAddressGuard and ZeroAmountGuard

SPARTADEX

audit / code review report

April 10, 2024

#3 - LACK OF PROPER VALIDATION IN STAKING CONTRACTS

Severity	Status
MEDIUM	RESOLVED

Since the constructor inherits from **Ownable** and **WithFees**, it's crucial to ensure that these contracts are correctly implemented and that the access control and fee mechanisms they provide are robust and secure.

RECOMMENDATION

To mitigate these risks, you should implement checks to ensure that the provided contract address is valid and meets certain criteria.

Values to fix :

IERC20 stakingToken_, IERC20 rewardToken_ in LinearStaking

auditmos.com

Secure your space

contact@auditmos.com