BEOSIN
Web3 Security & Compliance

Dex223

Smart Contract Security Audit

No. 202504300959

Apr 30th, 2025

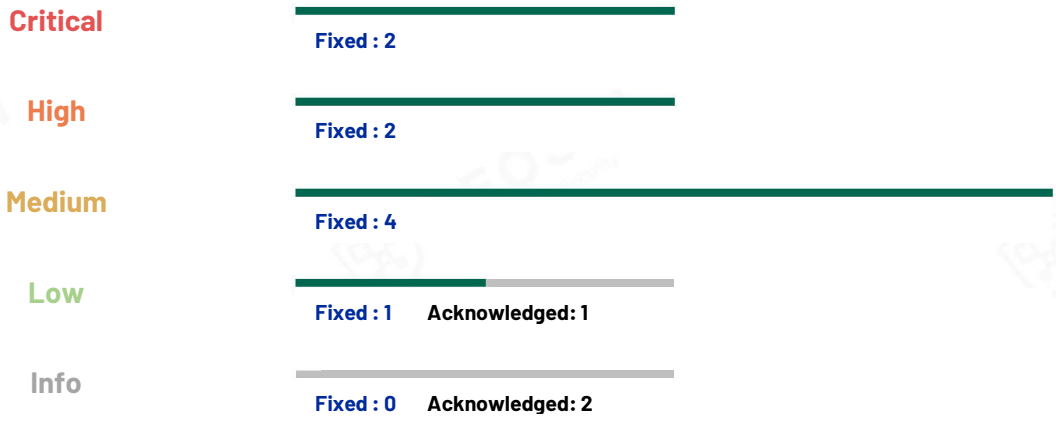
SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing ,2 Critical-risks, 2 High-risks, 4 Medium-risks, 2 Low-risks and 2 Info items were identified in the Dex223 project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:



- **Project Description:**

Dex223 is a decentralized trading protocol built on Uniswap V3, which retains the original architecture and functionalities of Uniswap V3 while introducing support for ERC-223 tokens. This enhancement enables seamless integration of ERC-223 tokens into the liquidity pools and trading mechanisms.

The project follows the Core-Periphery architecture, ensuring compatibility and scalability. The core contracts handle liquidity pool management, trade execution, and fee calculations, while the periphery contracts provide user interaction, price oracles, and additional functionalities, ensuring smooth ERC-223 token usage for both trading and liquidity provisioning. To support ERC-223, the protocol introduces several key improvements:

Seamless Integration Between ERC-20 and ERC-223 Tokens

To achieve interoperability between ERC-20 and ERC-223 tokens, the protocol implements a TokenConverter contract, allowing seamless conversion between original and wrapped tokens. Specifically, an ERC-20 token can be wrapped into a warpERC223 token, and an ERC-223 token can be converted into a warpERC20 token. This ensures that any token can simultaneously exist in both ERC-20 and ERC-223 formats, allowing frictionless exchange and enhanced compatibility.

Liquidity Pool & SwapRouter Compatibility with ERC-223

Since ERC-223 tokens automatically invoke the tokenReceived method when transferred, whereas Uniswap V3 was originally designed for ERC-20 tokens, additional modifications were required. The protocol introduces a custom tokenReceived handler, enabling correct sender recognition and token deposit tracking within liquidity pools. This ensures that ERC-223 tokens can be seamlessly deposited and swapped, while also allowing users to choose their preferred token format (ERC-20 or ERC-223) for output.

Optimized Liquidity Provisioning

With Uniswap V3's NFT-based liquidity positions (ERC-721), each liquidity provider (LP) holds individualized liquidity distributions. The project adjusts the liquidity management logic to ensure that LPs depositing ERC-223 assets can properly allocate their positions without risk of unexpected lock-up when price ranges shift. The Liquidity Manager contract has been extended to support ERC-223 deposits and withdrawals, making it fully compatible with V3 NFT positions while maintaining concentrated liquidity advantages for higher capital efficiency.

Smart Order Routing for Efficient Trade Execution

The Smart Order Routing mechanism has been enhanced to support multi-hop swaps for ERC-223 tokens. The protocol enables users to route trades through multiple paths to achieve optimal price execution. A route recognition algorithm automatically detects the token type and selects the appropriate transfer method for either ERC-20 or ERC-223 tokens. Additionally, the project retains Uniswap V3's multiple fee tiers (0.05%, 0.30%, and 1%), ensuring ERC-223 transactions seamlessly integrate without affecting LP rewards..

1 Overview

1.1 Project Overview

Project Name	Dex223
Project Language	Solidity
Platform	Ethereum
Code Base	https://github.com/EthereumCommonwealth/Dex223-contracts (Exclude contracts:Autolisting.sol, Dex223MarginModule.sol,Dex223Oracle.sol, Dex223OracleTwap.sol)
Commit ID	cd3e2a67fafed3b508c88f0cfcaea8148d768a31 a85f78697dd13c856411ca21fc0962318ad2b79a 7f7af971186d17807d08294ea37803a6df744d5d 1f3375bb6281cd1f08f613225246747a108b5805 b59624fbdbff217250685d06203557bc5f884d78 9e0922852c90ba2caa6544bbecce455c11ca48fe 565cfdb0f0c0a1b3ad2c2663825883af4cc9c426 270bb48f4caf55ddebe8b7836df41866259bfbd9 51c998ce233b3437fc5c6ce0bd831c6707a8e1bb cfa4f71990f25334a9dd52683126ae77f5bd38cc

1.2 Audit Overview

Audit work duration:Feb 20, 2025 – Mar 27, 2025, Apr 30,2025

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
Dex223-01	The visibility of the unwrapWETH9 function is incorrect	Critical	Fixed
Dex223-02	The tokenReceived function may be vulnerable to reentrancy attacks	Critical	Fixed
Dex223-03	Logical flaw in the executeSwapWithDeposit function	High	Fixed
Dex223-04	Token swap may also fail due to reentry lock	High	Fixed
Dex223-05	Incorrect data type	Medium	Fixed
Dex223-06	Improper function permission settings	Medium	Fixed
Dex223-07	The storage data of the proxy mode is disordered	Medium	Fixed
Dex223-08	Improper validation of the call result	Medium	Fixed
Dex223-09	Implementation flaw in the identifyTokens function	Low	Fixed
Dex223-10	Insufficient rigor in token address validation	Low	Acknowledged
Dex223-11	Redundant codes	Info	Acknowledged
Dex223-12	Key functions lack event logging	Info	Acknowledged

Finding Details:

[Dex223-01] The visibility of the unwrapWETH9 function is incorrect

Severity Level	**Critical**
Type	General Vulnerability
Lines	dex-core/Dex223Pool.sol #L479-487
Description	The `unwrapWETH9` function of the `Dex223Pool` contract has a security risk. This is a serious security issue because the function allows any user to withdraw WETH from the pool. Given that `unwrapWETH9` is called by other functions, we suspect that its visibility is improperly set, which may lead to a lack of access control. `function unwrapWETH9(address recipient, address WETH9, uint256 amountOut) public lock payable {`
Recommendation	It is recommended to change the visibility of this function to private to prevent unauthorized access.
Status	**Fixed.** This issue has been resolved in the new version of the code.

[Dex223-02] The tokenReceived function may be vulnerable to reentrancy attacks

Severity Level	**Critical**
Type	General Vulnerability
Lines	dex-core/Dex223Pool.sol #L188-215
Description	The `tokenReceived` function in the contract is designed to receive ERC-223 tokens and utilizes `delegatecall` to invoke other functions within the contract. However, an attacker can craft a malicious payload that forces `delegatecall` to recursively invoke the `tokenReceived` function, leading to a reentrancy attack. This could result in a single transaction being recorded multiple times in the ledger, potentially causing financial loss or state inconsistencies.
Recommendation	It is recommended to add a reentrancy protection mechanism to the `tokenReceived` function. However, care should be taken not to use the contract's lock modifier, as it may cause functionality conflicts.
Status	**Fixed.** In the latest version of the code, a reentrancy protection mechanism has been implemented in this function to prevent `delegatecall` from recursively invoking the `tokenReceived` function.

[Dex223-03] Logical flaw in the executeSwapWithDeposit function

Severity Level	High
Type	Business Security
Lines	dex-periphery/SwapRouter.sol #L228-256
Description	The `executeSwapWithDeposit` function in the contract is designed to handle swaps with ERC-223 tokens as input. The process first transfers tokens to the router and updates `_erc223Deposits`, then calls the corresponding swap function to complete the exchange. However, we have identified an issue where this function does not deduct the user's `_erc223Deposits` after the swap is executed. This is incorrect and may lead to miscalculations of asset balances or potential security risks.
Recommendation	It is recommended that the Router synchronously updates the user's `_erc223Deposits` data when transferring tokens to the `Pool` contract to ensure accurate balance calculations and prevent potential security risks.
Status	**Fixed.**

[Dex223-04] Token swap may also fail due to reentry lock

Severity Level	High
Type	Business Security
Lines	dex-core/Dex223Pool.sol #L479
Description	The `unwrapWETH9` function is decorated with the lock modifier, while the `swapExactInput` function also uses the lock modifier. This could cause a failure when attempting to call the `unwrapWETH9` function due to a lock conflict.
Recommendation	It is recommended to remove the lock modifier in the `unwrapWETH9` function.
Status	**Fixed.**

[Dex223-05] Incorrect data type

Severity Level	**Medium**
Type	General Vulnerability
Lines	dex-core/Dex223Pool.sol #L466
Description	The swap function calls the swap function of the `pool_lib` contract through `delegatecall` and obtains the return value. However, when the call fails, the error code of type uint256 is received, which is wrong. The error code should be of type string.

```
        (bool success, bytes memory retdata) =
pool_lib.delegatecall(abi.encodeWithSignature("swap(address,bool,int
256,uint160,bool,bytes)", recipient, zeroForOne, amountSpecified,
sqrtPriceLimitX96, prefer223, data));
        if (success) {
            (amount0, amount1) = abi.decode(retdata, (int256, int256));
        } else {
            uint256 val = abi.decode(retdata, (uint256));
            assembly {
                let ptr := mload(0x40)
                mstore(ptr, val)
                revert(ptr, 32)
            }
        }
```

Recommendation	It is recommended to use string type to receive the error code from the call.
Status	**Fixed.** This issue has been resolved in the new version of the code.

```
string memory val = abi.decode(retdata, (string));
```

[Dex223-06] Improper function permission settings

Severity Level	**Medium**
Type	General Vulnerability
Lines	dex-periphery/base/PeripheryPayments.sol #L74-103
Description	There is a `sweepToken` function in the `SwapRouter` contract, which is used to extract excess tokens in the contract. In theory, the `SwapRouter` contract should not hold token assets, so this is reasonable. However, due to the characteristics of ERC-223, that is, transferring tokens before performing operations, there may be excess tokens left in the contract. Therefore, if this function is retained, other users may use it to extract these extra ERC-223 tokens.
Recommendation	It is recommended to delete the `sweepToken` function. Users can use the withdraw function to withdraw their excess tokens.
Status	**Fixed.** The `sweepToken` function has been deleted in the new version of the code.

[Dex223-07] The storage data of the proxy mode is disordered

Severity Level	**Medium**
Type	Coding Conventions
Lines	contracts\dex-core\Dex223PoolLib.sol
Description	The Dex223Pool contract of the project will use `delegatecall` to call the related functions of the `Dex223PoolLib` contract. However, during the project code update, the `erc223ReentrancyLock` variable was added to the `Dex223Pool` contract, but it was not added synchronously in the Dex223PoolLib contract, which may cause confusion in the data obtained when calling `delegatecall`.
Recommendation	It is recommended to add the `erc223ReentrancyLock` variable to the corresponding position of the `Dex223PoolLib` contract for storage space.
Status	**Fixed.**

[Dex223-08] Improper validation of the call result

Severity Level	**Medium**
Type	Business Security
Lines	dex-core\Dex223PoolLib.sol #L382 dex-periphery\SwapRouter.sol #L248

Description	(1) In the `Dex223PoolLib` contract, the `transfer` function of the target token contract is called and the result is used to determine whether the transfer is successful, but it should be noted that the USDT token has no return value for its transfer. Therefore, according to the current code logic, even if the USDT token transfer is successful, its `tokenNotExist` is true, and it is considered that the token does not exist, and it is unreasonable to perform token exchange.
	`(bool success, bytes memory data) = _token.call(abi.encodeWithSelector(IERC20Minimal.transfer.selector, _recipient, _amount));`
	`bool tokenNotExist = (success && data.length == 0);`
	(2) Similarly, in the `SwapRouter` contract, the `balanceOf` function of the target token contract is called to obtain the address balance. However, the result of `tokenNotExist` is true when needed, which is unreasonable.
	`(bool success, bytes memory resdata) = _tokenOut.call(abi.encodeWithSelector(IERC20.balanceOf.selector, recipient));`
	`bool tokenNotExist = (success && resdata.length == 0);`

Recommendation	For the transfer function in the `Dex223PoolLib` contract, we recommend that token exchange be performed when the call result is false. The `balanceOf` function of the `SwapRouter` contract needs to check the call result and the return value length is 32. If it is, it means that the balance is obtained successfully.
Status	**Fixed.** This has been fixed in commit 270bb48f.

[Dex223-09] Implementation flaw in the identifyTokens function

Severity Level	Low
Type	Business Security
Lines	dex-core/Dex223Factory.sol #L182
Description	As shown in the following code, when the predictWrapperAddress function is called, the parameter _token223 must be a Warp223 token, so when its code size is greater than 0, it means that it has been created. In this case, you only need to compare the obtained Origin ERC20 token with _token, and there is no need to use predictWrapperAddress for address prediction. On the other hand, the predictWrapperAddress function also predicts the wrapper address based on the Origin token, but the parameter passed here is _token223, which is unreasonable because _token223 must be the wrapped token at this time.
Recommendation	It is recommended to remove the call check of the predictWrapperAddress function from the code here, because it is unnecessary.
Status	**Fixed.**

```
        if(!converter.isWrapper(_token))
        {
            uint256 _code_size;
            assembly { _code_size := extcodesize(_token223) }
            if(_code_size > 0)
            {
                (bool success, bytes memory data) =
_token223.staticcall(abi.encodeWithSelector(0x5a3b7e42)); // call
`standard() returns uint32`
                require(success && abi.decode(data,(uint32)) ==
uint32(223));
                require((converter.getERC20OriginFor(_token223) ==
_token) || converter.predictWrapperAddress(_token223, false) ==
_token);
                return; // All checks passed for scenario 1.
            }
```

[Dex223-10] Insufficient rigor in token address validation

Severity Level	Low
Type	Business Security
Lines	dex-core/Dex223Pool.sol #L515
Description	When the user calls the `swapExactInput` function, he can specify `unwrapETH` as true, indicating that he wants to get ETH instead of WETH. Then the contract will automatically exchange WETH for ETH. This is a reasonable business logic, but now there may be an abnormal situation: the user's exchange path is WETH to token A, but `unwrapETH` is also set to true. Then the contract will call the unwrapWETH9 function to unwrap WETH with the amount of token A obtained. Therefore, if (1) the A token contract must implement the withdraw function, because the `unwrapWETH9` function will call this function; (2) the contract must have enough ETH for the `unwrapWETH9` function to withdraw. Then the ETH of the contract can be withdrawn. These conditions seem harsh, but it is still recommended to determine that WETH9 in the `unwrapWETH9` function is the specified WETH token address.
Recommendation	It is recommended to strictly check that the `WETH9` parameter in the unwrapWETH9 function is the specified WETH token address.
Status	**Acknowledged.**

[Dex223-11] Redundant codes

Severity Level	Low
Type	Coding Conventions
Lines	dex-core/Dex223Factory.sol #L22 converter/TokenConverter.sol #L115 dex-core/Dex223Pool.sol #L108

Description	The contract contains some interfaces and variable declarations that are unused, which constitute redundant code.
	(1) The `standardIntrospection` variable in the `Dex223Factory` contract
	`ITokenStandardIntrospection public standardIntrospection;`
	(2) The IERC20WrapperToken interface declares standard, but the ERC20WrapperToken contract does not inherit this interface and does not implement the standard function.
	(3) The logic code related to `protocolFees` of the Dex223Pool contract has been deleted, making it redundant code.
	`ProtocolFees public override protocolFees;`

Recommendation	It is recommended to remove redundant code that has no practical significance to improve the readability and maintainability of the contract. Redundant code not only increases complexity but may also introduce unnecessary security risks. Therefore, cleaning up unused or non-functional code can make the contract more concise, efficient, and easier to maintain.
Status	**Acknowledged.**

[Dex223-12] Key functions lack event logging

Severity Level	Info
Type	Coding Conventions
Lines	converter/TokenConverter.sol
Description	In the mint and burn functions of the `ERC223WrapperToken` and `ERC20WrapperToken` contracts, there is a lack of event triggers for updating token balances. This is a bad practice that is detrimental to off-chain record-keeping of contract data. In particular, some blockchain explorers rely on events to update contract status, which may lead to inaccurate display of data on the explorer.
Recommendation	It is recommended to trigger the Transfer event in the mint and burn functions.
Status	**Acknowledged.**

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Critical**

Critical impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other Critical and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.3 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.4 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Deprecated Items
		Redundant Code
		require/assert Usage
		Default Values
2	General Vulnerability	Insufficient Address Validation
		Lack Of Address Normalization
		Variable Override
		DoS (Denial Of Service)
		Function Call Permissions
		Call/Delegatecall Security
		Tx.origin Usage
		Returned Value Security
		Mathematical Risk
		Overriding Variables
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Arbitrage Attack
		Access Control

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Rust language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Web3 Security & Compliance

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

X
https://x.com/Beosin_com

Email
service@beosin.com

LinkedIn
https://www.linkedin.com/company/beosin/