CexiSwap

Smart Contract Review of Dex.sol

January 2022

Casaba Security SE Asia PTE LTD

www.casaba.com

US-CXS-2021-01

Table of Contents

Executive Summary

In January of 2022, CexiSwap engaged Casaba Security SE Asia("Casaba") to conduct a smart contract review of a smart contract intended for use in its CexiSwap platform. CexiSwap is a decentralized exchange (Dex) used for trading Ethereum and ERC20 tokens without a brokerage acting as an intermediary.

The scope of this engagement focused solely on a single solidity file (dex.sol) containing CexiSwap's on-chain mechanism for swapping tokens. The project did not include any review of CexiSwap's application(app.cexiswap.io) or any other code related to the project. Activities included in this assessment were:

- Static analysis of source contract using automated tools
- Manual review and dynamic analysis of the smart contract in an offline testing environment.

Casaba found the code to be professionally written with no apparent security issues. The functionality of the smart contract is straightforward, and no logical errors arose during testing. We found that the contract performs as intended without any misconfiguration or potential for issues. Casaba does not have an opinion on the security of the CexiSwap application as a whole due to limited scope of this engagement. Further testing of the entire CexiSwap application, including how Dex.sol is being called, should be performed prior to the code entering production.

Goals

Casaba performed a code review and audit of the CexiSwap's smart contracts. This engagement included the following tasks:

Design and Syntax

- Are the contracts written according to best practices?
- Are there any immediately apparent vulnerabilities such as confusing naming conventions or incorrect formatting that would cause the contracts to deviate from expected behavior?
- Does the code implement any unsafe design patterns or contain any logical issues which could cause it to behave in ways other than intended?

Automated Analysis

- Run the code through several automated security tools and frameworks to discover known vulnerabilities

Functional Testing

- Package and deploy the contracts to Casaba's test environment
- Build a test harness to analyze the contract at a functional level
- Perform functional verification of each method in the code
- Develop and run through test cases that confirm expected behavior
- Manually attempt to circumvent expected behavior
- Attempt to exploit any discovered flaws

Scenario and Analysis

Assessment Type and Approach

The engagement was conducted using a white box methodology. Casaba had full access to the repository containing the in-scope code. Other code and documentation within the repository was referenced to better understand the contract's functionality but none of these items were subject to security review. The goal of reviewing source code is to identify potential issues before an exploit can be written. Both manual and automated techniques were used to determine if the code contained any potential issues.

Dex.sol

The primary functionality of the contract resides within the Swap() function. The function takes in two sets of data:

- The transaction details including
  - ↳ Amount of each token to be swapped
  - ↳ Ethereum addresses detailing the contract for each token being swapped
  - ↳ The address where the tokens are to be sent.
- An {r,s,v} signature of the transaction signed by an address designated to be the "price quote signer" used to confirm that the swap transaction is valid

The swap takes place in four steps:

1. A transaction hash is generated using the data provided as input.
2. The computed hash and the provided {r,s,v} signature are put through a function to determine the address which generated the signature.
3. The generated signature is compared to the approve "price quote signer" to determine the validity of the quote.
4. The swap is initiated

The remaining functionality of the contract serves to manage the internal variables of the contract. Access to those functions is limited using role-based access controls.

Analysis

Casaba created a test environment consisting of a deployed instance of the Dex contract, two ERC-20 compliant tokens, and separate addresses assigned to each role specified by the contract. The contract makes use of OpenZepplin's framework for deploying upgradeable contracts. Casaba confirmed the ability to upgrade the contract was limited to the address assigned the "UPGRADE_ROLE" but did not attempt to perform any upgrades. Testing of the core functionality was completed without identifying any security issues.

Dex.sol makes excellent use of exception handling to ensure correct execution of transactions. The contract makes liberal use of "*require*" statements prior to making any changes to state. Require statements are superior to throwing an error in Solidity because their conditions are checked prior to

execution of any code. An invalid opcode (0xfd) is sent, and an error is raised if any condition fails. The following excerpt of the priceQuoteIsValid() function is a good example of the contract's error handling:

```
require(getTime() <= deadline, 'PriceQuote expired');
require(inTokenAddr != outTokenAddr, 'same in/out token');
require(inTokenAddr == token0 || inTokenAddr == token1, 'invalid inTokenAddr');
require(outTokenAddr == token0 || outTokenAddr == token1, 'invalid outTokenAddr');
require(!usedNonces[toAddr][nonce], 'used nonce');
```

The function is comparing multiple variables such as block time and transaction nonce to prevent attempts at replay attacks while also checking that the tokens being swapped can actually be handled by the contract.

Error checking is combined with EIP-712 compliant signature validation to confirm that the transaction originates from an approved address(image).

```
bytes32 digest = keccak256(
    abi.encodePacked(
        '\x19\x01',
        domainSeparator,
        keccak256(
            abi.encodePacked(
                keccak256(
                    'PriceQuote(uint256 amountIn,uint256 amountOut,address toAddr,uint256 nonce,uint256 deadline,address inTokenAddr,address outTokenAddr)'
                ),
                amountIn,
                amountOut,
                toAddr,
                nonce,
                deadline,
                inTokenAddr,
                outTokenAddr
            )
        )
    )
);
address recovered = ecrecover(digest, v, r, s);
require(recovered == priceQuoteSigner, 'PriceQuote sig wrong');
return true;
```

The design pattern is excellent since it limits the vectors for an attacker to submit fraudulent transactions in the event that the off-chain component of CexiSwap is compromised. A successful breach of the application will not be able to make unauthorized swaps provided the keys to the priceQuoteSigner address are stored within a separate transaction signing environment.

The tight error handling combined with the use of access control roles heavily limit the vectors for abuse. Any attempts to make an unauthorized transaction were rejected. Trying to drain the contract of funds through fraudulent transactions were similarly unsuccessful. The withdraw() function is inaccessible to any address which has not been assigned the "WITHDRAW_ROLE". Similarly, the ability to change variables within the contract are properly secured to the "ADMIN_ROLE". The contract is hardened against unauthorized use, and it has the capability to be secure from insider threats provided the existing roles are properly assigned and controlled.

Recommendations

Casaba raised two observational findings as a result of this engagement. The findings do not represent direct threats to the contract but are areas that Casaba believes would be beneficial for CexiSwap to review. The first finding is straightforward. The priceQuoteisValid() function uses signature verification to confirm that any quote has been signed by the approved key. As noted previously the pattern protects against unauthorized transactions. It could further be improved by adding an explicit check that the recovered key is valid. The additional check helps protect against certain potential attack vectors while also creating additional clarity since it will be possible to distinguish a fraudulently signed transaction from a transaction with an invalid signature.

The second area for investigation has to do with segregation of roles when the smart contract enters production. The contract as written makes use of OpenZepplin's Access Control framework. The framework allows for the use of RBAC to govern access to smart contract functionality. Its efficacy hinges on an organization's ability to properly assign those roles. The default behavior of Dex.sol is to assign all three roles to a single address specified during initialization. This introduces risk because a single account has full rights to remove tokens from the contract. CexiSwap should ensure that deployment of the contract includes creation of separate keypairs for each role along with adequate protection for all private keys. Any addresses with the ADMIN_ROLE or WITHDRAW_ROLE roles should be protected to the same degree as the keys for the transaction signing account.

To summarize:

- Add an explicit check for a valid ECDSA signature in priceQuoteIsValid()
- Ensure proper assignment of roles and management of the related private keys

Detailed Findings

The following findings represent issues or areas of interest which our team wishes to highlight.

ID	Severity	Title
CXS-01	Info	Consider adding check for invalid ECDSA signature in priceQuoteIsValid()
CXS-02	Iinfo	Review assignment of access control rules to ensure adequate segregation of privileges

At a minimum, Casaba recommends all vulnerabilities of Medium severity or greater be mitigated. Also, we strongly encourage all rated Low and Observations to be reviewed, as they may call attention to or address significant underlying risks, to determine if further action is necessary.

1.1 Finding Taxonomy

Casaba's detailed findings follow a standard format. This format includes the necessary details to completely describe a vulnerability, including its impact if left unmitigated, how it was identified and how it may be reproduced. Vulnerabilities are given one or more applicable STRIDE categories and scored according to version 3 of the Common Vulnerability Scoring System (CVSS) standard. The qualitative severity rating is derived from the CVSS score. The qualitative severity is a well-understood industry standard can be easily mapped to other severity risk rating systems. The various fields and their meanings are detailed below.

Title: The name of the finding.

Severity: The qualitative severity level of the finding. Four values are possible: Critical, High, Medium, and Low.

Categorization: This is the general categorization of the finding. Values are any of the six STRIDE threat categories, with the addition of Attack Surface Reduction and Observation. A finding will always have at least one category, but it is possible to have several. For details on the STRIDE categories, please refer to The STRIDE Threat Model. Findings categorized as Attack Surface Reduction represent an opportunity to reduce the exposure of the system to the risk of attack by removing potentially vulnerable components. Findings categorized as Observations are not vulnerabilities but represent areas of continuing interest.

Summary: This is the technical description of the bug.

Impact: This describes the problems that may arise if the finding is left unmitigated.

Recommendation: The steps required to mitigate the issue are detailed here.

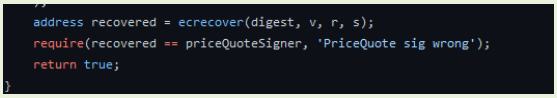
Reference Info: Links to relevant reference materials, including CVE entries at the National Vulnerability Database, MSDN technical articles, etc.

Affected Resources: Client resources and assets that affected by the finding are detailed here. Affected resources may be described as IP address/port tuples, URLs and/or file/line tuples.

Notes: Additional details germane to the finding will be called out here. Notes may include assigned bug tracking system IDs, the rationale behind CVSS scoring details, etc.

[CXS-01] Consider adding check for invalid ECDSA signature in priceQuoteIsValid()

Severity	Info
Categorization	Observation
Summary	The current implementation for error handling does not check for an invalid signature.
Impact	Adding a check for an invalid signature would better align the code to the EIP-712 standard and add additional protection against unauthorized transactions while also adding clarity when debugging any issues.
Recommendation	Consider appending an addition check explicitly confirming that the recovered signature is valid after confirming that the address matches the priceQuoteSigner. E.g.: *require(recovered != address(0), "Invalid Signature")*

Reference Info	Dex.sol:123
	 
	`address recovered = ecrecover(digest, v, r, s);`
	`require(recovered == priceQuoteSigner, 'PriceQuote sig wrong');`
	`return true;`
	`}`

Affected Resource(s)	Dex.sol
Notes	None

[CXS-02] Review assignment of access control rules to ensure adequate segregation of privileges

Severity	Info
Categorization	Observation
Summary	The scope of this engagement did not cover how the distinct roles defined in dex.sol will be assigned in a production environment. Casaba confirmed that the RBAC functions but lacks any visibility into how CexiSwap manages the private keys for the various functionality.
Impact	The contract uses the following access control roles to limit who can execute certain functionality. These roles are: • ADMIN-ROLE ○ Change the approved price quote signer

	○ Manage the whitelist of addresses allowed to withdraw ETH and ERC-20 tokens from the contract ○ Add and revoke other roles • WITHDRAW_ROLE ○ Address allowed to send tokens from the contract address to be whitelisted addresses • UPGRADE_ROLE ○ Used to upgrade the contract using OpenZepplin's plugin The contract assigns all three roles to the single address passed in the "admin" parameter. Securing the admin role becomes essential since it is functionally the owner of the contract. The role should not be used by itself in production as doing so defeats the purpose of using access controls in the first place.
Recommendation	Casaba recommends that CexiSwap ensures that each role is assigned to a discrete address generated from separate private keys. The keys should then be securely stored within CexiSwap's infrastructure according to their importance. An example for key management could be: • The ADMIN_ROLE key should be stored offline or in a secure keystore for emergency access. • The UPGRADE_ROLE key should be stored securely within CexiSwap's development environment • The WITHDRAW_ROLE key should be stored either offline or within CexiSwap's secure signing environment similar to the priceQuoteSigner key

Reference Info	Dex.sol:58-63
	`58 _setRoleAdmin(ADMIN_ROLE, ADMIN_ROLE);`
	`59 _setRoleAdmin(WITHDRAW_ROLE, ADMIN_ROLE);`
	`60 _setRoleAdmin(UPGRADE_ROLE, ADMIN_ROLE);`
	`61 _setupRole(ADMIN_ROLE, admin);`
	`62 _setupRole(WITHDRAW_ROLE, admin);`
	`63 _setupRole(UPGRADE_ROLE, admin);`

Affected Resource(s)	Dex.sol
Notes	None

Tools Used

The following tools and environments were among those used by our team during this engagement:

- Remix IDE
- Ganache
- Mithril
- Slither

Scope

The following code was reviews as part of this assessment:

File	Language	Commit	LoC
https://github.com/cexiswap/cexiswap-contracts/blob/main/contracts/Dex.sol	solidity	a89f84d	254