Quidax

Smart Contract Audit
Final Report

May 07, 2021

Introduction

1. About Quidax

Quidax is a cryptocurrency exchange that gives anyone access to popular cryptocurrencies. It was officially launched in 2018 and currently services users in over 70 countries.

Quidax's vision is ensuring that millions of people around the world are able to access digital assets and upcoming blockchain projects in the BEP20 ecosystem.

Since its launch three years ago, Quidax has achieved important milestones, including 30,000+ weekly users, $8,000,000 USD daily trading volume, 200,000+ downloads since the mobile app launch, and a user-base of 400,000+ registered users.

Visit https://www.quidax.com/ to learn more about.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The Quidax team has not provided any documentation to conduct the audit.

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: Quidax
- Languages: Solidity(Smart contract)
- Github commit hash for audit: ff55a873ad0ae26d4c6a4c9efb676c0daa61f59b
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	-	-	-
Closed	-	-	4

High severity issues

No issues found

Medium severity issues

No issues found

Low severity issues

1. **Comparison to boolean Constant**
   Line no: 419,428, 437, 448, 605, 606, 607, 627, 628, 650, 651, 692, 693, 694,
   Description:
   Boolean constants can directly be used in conditional statements or require statements. Therefore, it's not considered a better practice to explicitly use **TRUE or FALSE** in the **require** statements.

```
437         require(_blacklists[_address] == false,
438         _blacklists[_address] = true;
439         return true;
```

   Recommendation:
   The equality to boolean constants must be removed from the above-mentioned line.

   Amended (May 7th 2021): Issue was fixed by Quidax team and is no longer present in the code.

2. **External Visibility should be preferred**
   Description:
   Those functions that are never called throughout the contract should be marked as **external** visibility instead of **public** visibility.
   This will effectively result in Gas Optimization as well.

   Therefore, the following function must be marked as **external** within the contract:
   - **name**
   - **symbol**
   - **decimals**
   - **totalSupply**
   - **balanceOf**

- **paused**
- **blackListed**
- **pause**
- **unpause**
- **blacklist**
- **whitelist**
- **transfer**
- **mint**
- **burn**

Recommendation:
If the **PUBLIC** visibility of the above-mentioned function is not intended, the function visibility should be changed to **EXTERNAL**.

Amended (May 7th 2021): Issue was fixed by Quidax team and is no longer present in the code.

3. **Absence of Error messages in Require Statements**
Line no - 668
Description:
The **_burnFrom** has a **require** statement in the QuiDax.sol contract that does not include an error message.

```
668         require(amount <= _allowances[account][msg.sender]);
669
```

While this makes it troublesome to detect the reason behind a particular function revert, it also reduces the readability of the code.

Recommendation:
Error Messages must be included in every **require** statement in the contract.

Amended (May 7th 2021): Issue was fixed by Quidax team and is no longer present in the code.

4. **Redundant Initialization of State Variable**
   Line no - 350
   Description:
   The State Variable **_paused** is being initialized to **FALSE** in the constructor.

```
350         _paused = false;
```

   However, boolean state variables are **FALSE** by default and do not require explicit initialization to false.

   Recommendation:
   Unnecessary Initialization of State variables should be avoided in the contract.

   Amended (May 7th 2021): Issue was fixed by Quidax team and is no longer present in the code.

Recommendations

1. **Coding Style Issues in the Contract**
   Description:
   Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

   During the automated testing, it was found that the Quidax contract had quite a few code style issues.

```
Parameter Quidax.blacklisted(address)._address (contracts/Quidax.sol#410) is not in mixedCase
Parameter Quidax.blacklist(address)._address (contracts/Quidax.sol#436) is not in mixedCase
Parameter Quidax.whitelist(address)._address (contracts/Quidax.sol#447) is not in mixedCase
Parameter Quidax.mintToMultipleAddresses(address[],uint256[])._addresses (contracts/Quidax.sol#479) is not in mixedCase
Parameter Quidax.mintToMultipleAddresses(address[],uint256[])._amount (contracts/Quidax.sol#479) is not in mixedCase
```

   Recommendation:
   Therefore, it is highly recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

   Amended (May 7th 2021): Issue was fixed by Quidax team and is no longer present in the code.

Automated Test Results

```
Quidax.pause() (contracts/Quidax.sol#418-422) compares to a boolean constant:
        -require(bool,string)(_paused == false,Quidax: token transfer is unavailable) (contracts/Quidax.sol#41
Quidax.unpause() (contracts/Quidax.sol#427-431) compares to a boolean constant:
        -require(bool,string)(_paused == true,Quidax: token transfer is available) (contracts/Quidax.sol#428)
Quidax.blacklist(address) (contracts/Quidax.sol#436-440) compares to a boolean constant:
        -require(bool,string)(_blacklists[_address] == false,Quidax: account already blacklisted) (contracts/Q
Quidax.whitelist(address) (contracts/Quidax.sol#447-451) compares to a boolean constant:
        -require(bool,string)(_blacklists[_address] == true,Quidax: account already whitelisted) (contracts/Qu
Quidax._transfer(address,address,uint256) (contracts/Quidax.sol#602-614) compares to a boolean constant:
        -require(bool,string)(_blacklists[recipient] == false,Quidax: sender account already blacklisted) (cont
Quidax._transfer(address,address,uint256) (contracts/Quidax.sol#602-614) compares to a boolean constant:
        -require(bool,string)(_blacklists[sender] == false,Quidax: sender account already blacklisted) (contra
Quidax._transfer(address,address,uint256) (contracts/Quidax.sol#602-614) compares to a boolean constant:
        -require(bool,string)(_paused == false,Quidax: token contract is not available) (contracts/Quidax.sol#
Quidax._mint(address,uint256) (contracts/Quidax.sol#625-635) compares to a boolean constant:
        -require(bool,string)(_paused == false,Quidax: token contract is not available) (contracts/Quidax.sol#
Quidax._mint(address,uint256) (contracts/Quidax.sol#625-635) compares to a boolean constant:
        -require(bool,string)(_blacklists[account] == false,Quidax: account to mint to already blacklisted) (c
Quidax._burn(address,uint256) (contracts/Quidax.sol#648-658) compares to a boolean constant:
        -require(bool,string)(_blacklists[account] == false,Quidax: account to burn from already blacklisted)
Quidax._burn(address,uint256) (contracts/Quidax.sol#648-658) compares to a boolean constant:
        -require(bool,string)(_paused == false,Quidax: token contract is not available) (contracts/Quidax.sol#
```

```
Quidax.constructor() (contracts/Quidax.sol#341-352) uses literals with too many digits:
        - _mint(_msgSender(),50000000000000000000000000000) (contracts/Quidax.sol#346)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

```
blacklist(address) should be declared external:
        - Quidax.blacklist(address) (contracts/Quidax.sol#436-440)
whitelist(address) should be declared external:
        - Quidax.whitelist(address) (contracts/Quidax.sol#447-451)
transfer(address,uint256) should be declared external:
        - Quidax.transfer(address,uint256) (contracts/Quidax.sol#46
mint(address,uint256) should be declared external:
        - Quidax.mint(address,uint256) (contracts/Quidax.sol#471-47
burn(uint256) should be declared external:
        - Quidax.burn(uint256) (contracts/Quidax.sol#494-497)
burnFrom(address,uint256) should be declared external:
        - Quidax.burnFrom(address,uint256) (contracts/Quidax.sol#50
approve(address,uint256) should be declared external:
        - Quidax.approve(address,uint256) (contracts/Quidax.sol#528
transferFrom(address,address,uint256) should be declared external:
        - Quidax.transferFrom(address,address,uint256) (contracts/Q
increaseAllowance(address,uint256) should be declared external:
```

Concluding Remarks

While conducting the audits of Quidax smart contract - Quidax.sol, it was observed that the contracts contain only Low severity issues, along with a few areas of recommendations.

Our auditors suggest that Low severity issues should be resolved by the Quidax developers. Resolving the areas of recommendations are up to the team's discretion. The recommendations given will improve the operations of the smart contract.

Note: Quidax team has fixed the Medium and Low issues based on the auditor's recommendation.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Quidax platform or its product neither this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes Pvt Ltd.