Quidax

JulPadToken.sol

Smart Contract Audit Report

IMMUNE BYTES

Audits

quidax

May 26, 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
- Contract Name: JulPadToken.sol
- Languages: Solidity(Smart contract)
- Github commit hash for audit/Contract Address:
  - https://bscscan.com/token/0x9e3a9F1612028eeE48F85cA85f8Bed2f37d76848
- Testnet Deployment (Rinkeby)
  - https://rinkeby.etherscan.io/address/0x1A5E45E4F74C25B3762FD9BE675DECED5F86950C
- 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:

Admin/Owner Privileges can be misused either intentionally or unintentionally.

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	-	-	4
Closed	-	-	-

Admin/Owner Privileges

The **admin/owner** of **JulPadToken** has various privileges over the smart contracts. These privileges can be misused either intentionally or unintentionally (in case the admin's private key gets hacked). We assume that these extra rights will always be used appropriately. Some of these admin rights are listed below.

1. **In the JulPadToken contract the MINTER address can mint any amount of tokens.**
   The **JulPadToken** contract contains a **mint**() function by which the **MINTER** account can mint any number of tokens to any ethereum address.

2. **MINTER role can be given to any address/account by the tokenOwner and MINTER of JulPadToken contract.**
   This **tokenOwner and MINTER** of the contract has the right to give a **MINTER** role to any ethereum address. Please note that there can be more than one **MINTER** possible for the **JulPadToken** contract.

3. **PAUSER role can be given to any address/account by the tokenOwner and PAUSER of JulPadToken contract.**
   This **tokenOwner and PAUSER** of the contract has the right to give a **PAUSER** role to any ethereum address. Please note that there can be more than one **PAUSER** possible for the **JulPadToken** contract. The **PAUSER** has the right to pause and unpause all token functionalities.

4. **Owner role can be given to any address/account by the tokenOwner of JulPadToken contract.**
   Using the **setOwner**() function the **tokenOwner** of the contract has the right to give a **Owner** role to any ethereum address. The Owner is also a **MINTER** and **PAUSER**.

Recommendation:
Consider adding some governance for admin rights for smart contracts or use a multi-sig wallet as admin/owner address.

Low severity issues

1. **Ether can get locked in the JulPadToken contract.**
   The **JulPadToken** contract contains a **payable** fallback function which can be used to send Ether to the **JulPadToken** contract. Since there is no way to recover those sent Ethers, those funds get locked inside the contract forever.
   ```
       function() external payable {}
   ```

   Recommendation:
   Consider removing the payable fallback function or implement a logic to pull out the Ether sent to **JulPadToken** contract.

2. **No need to reserve storage space in the contract.**
   In Solidity smart contracts, generally the storage space is reserved for those contracts which are planned to be upgraded in future. Since the **JulPadToken** does not have any code which helps in upgrading the contract logic there is no need to reserve storage space.
   ```
       Reserved storage space to allow for layout changes in the future.
       uint256[50] private ______gap;
   ```

   Recommendation:
   Consider removing all the **______gap** variables.

3. **Redundant logic for contract's ownership present.**
   The **JulPadToken** contract inherits the **Ownable** smart contract which contains all the necessary logic for handling the ownership of smart contract. But the ownership logic is again implemented in the **JulPadToken** contract using the **tokenOwner** variable and **setOwner**() function.

   Recommendation:
   Consider avoiding redundant implementations in the smart contract.

4. **Functions should be declared as external.**
   Most of the functions in **JulPadToken** contract are declared as **public**. Since these functions are not called from inside the **JulPadToken** contract, these functions can be declared as **external**. **External** functions cost less gas than **public** functions.

   Recommendation:
   Consider declaring the functions as **external**.

Automated Auditing

Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our VScode for this analysis. Multiple Linting violations were detected by Solhint, it is recommended to use Solhint's npm package to lint the contracts.

Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to them in real-time. We performed analysis using contract Library on the Rinkeby address of the JulPadToken contract used during manual testing:

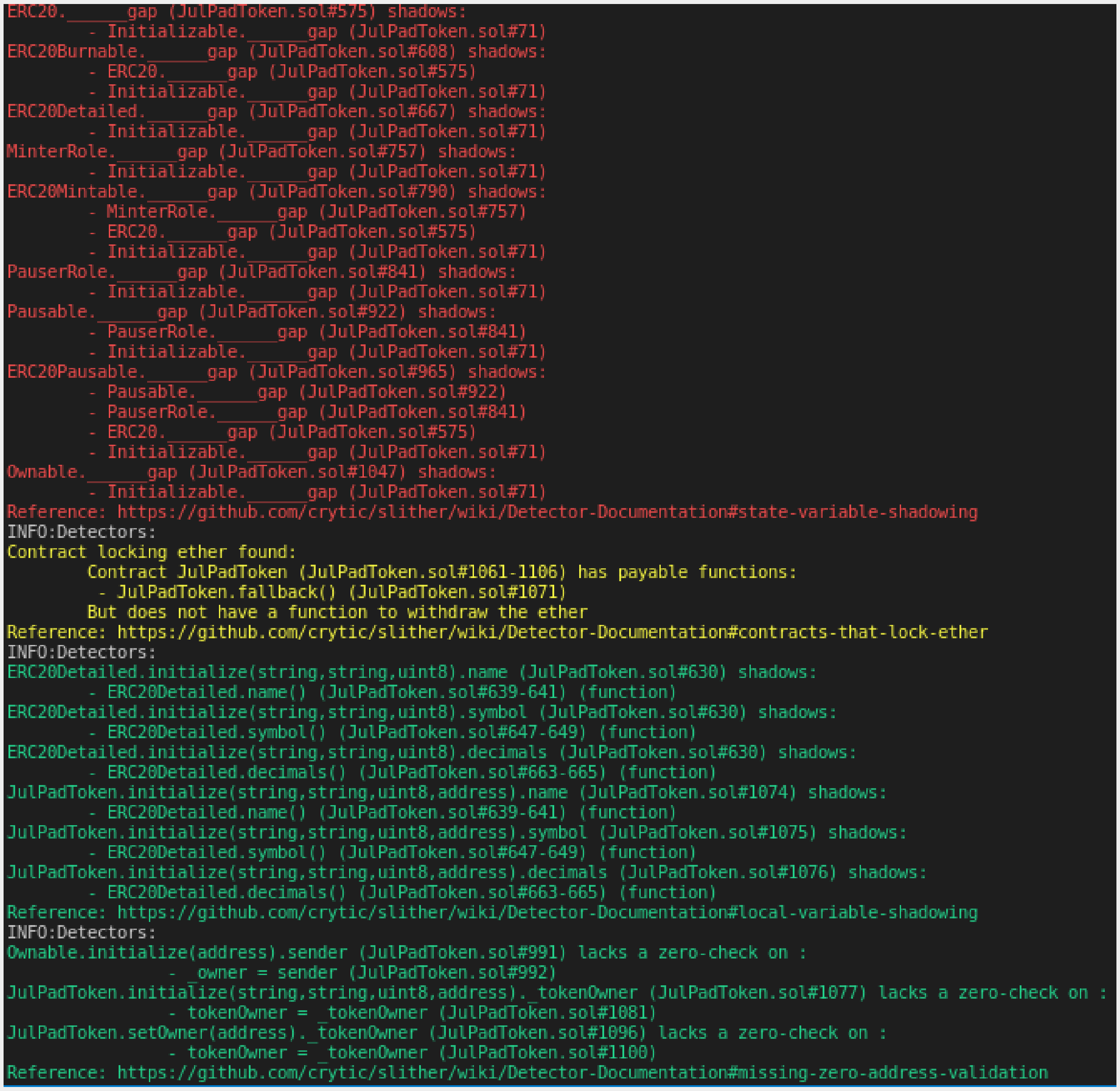
- JulPadToken: 0x1a5E45e4f74c25b3762FD9BE675dECed5f86950c

It raises no major concern for the contract.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

The concerns slither raises have already been covered in the manual audit section.



```
ERC20.______gap (JulPadToken.sol#575) shadows:
        - Initializable.______gap (JulPadToken.sol#71)
ERC20Burnable.______gap (JulPadToken.sol#608) shadows:
        - ERC20.______gap (JulPadToken.sol#575)
        - Initializable.______gap (JulPadToken.sol#71)
ERC20Detailed.______gap (JulPadToken.sol#667) shadows:
        - Initializable.______gap (JulPadToken.sol#71)
MinterRole.______gap (JulPadToken.sol#757) shadows:
        - Initializable.______gap (JulPadToken.sol#71)
ERC20Mintable.______gap (JulPadToken.sol#790) shadows:
        - MinterRole.______gap (JulPadToken.sol#757)
        - ERC20.______gap (JulPadToken.sol#575)
        - Initializable.______gap (JulPadToken.sol#71)
PauserRole.______gap (JulPadToken.sol#841) shadows:
        - Initializable.______gap (JulPadToken.sol#71)
Pausable.______gap (JulPadToken.sol#922) shadows:
        - PauserRole.______gap (JulPadToken.sol#841)
        - Initializable.______gap (JulPadToken.sol#71)
ERC20Pausable.______gap (JulPadToken.sol#965) shadows:
        - Pausable.______gap (JulPadToken.sol#922)
        - PauserRole.______gap (JulPadToken.sol#841)
        - ERC20.______gap (JulPadToken.sol#575)
        - Initializable.______gap (JulPadToken.sol#71)
Ownable.______gap (JulPadToken.sol#1047) shadows:
        - Initializable.______gap (JulPadToken.sol#71)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
Contract locking ether found:
        Contract JulPadToken (JulPadToken.sol#1061-1106) has payable functions:
         - JulPadToken.fallback() (JulPadToken.sol#1071)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
ERC20Detailed.initialize(string,string,uint8).name (JulPadToken.sol#630) shadows:
        - ERC20Detailed.name() (JulPadToken.sol#639-641) (function)
ERC20Detailed.initialize(string,string,uint8).symbol (JulPadToken.sol#630) shadows:
        - ERC20Detailed.symbol() (JulPadToken.sol#647-649) (function)
ERC20Detailed.initialize(string,string,uint8).decimals (JulPadToken.sol#630) shadows:
        - ERC20Detailed.decimals() (JulPadToken.sol#663-665) (function)
JulPadToken.initialize(string,string,uint8,address).name (JulPadToken.sol#1074) shadows:
        - ERC20Detailed.name() (JulPadToken.sol#639-641) (function)
JulPadToken.initialize(string,string,uint8,address).symbol (JulPadToken.sol#1075) shadows:
        - ERC20Detailed.symbol() (JulPadToken.sol#647-649) (function)
JulPadToken.initialize(string,string,uint8,address).decimals (JulPadToken.sol#1076) shadows:
        - ERC20Detailed.decimals() (JulPadToken.sol#663-665) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Ownable.initialize(address).sender (JulPadToken.sol#991) lacks a zero-check on :
                - _owner = sender (JulPadToken.sol#992)
JulPadToken.initialize(string,string,uint8,address)._tokenOwner (JulPadToken.sol#1077) lacks a zero-check on :
                - tokenOwner = _tokenOwner (JulPadToken.sol#1081)
JulPadToken.setOwner(address)._tokenOwner (JulPadToken.sol#1096) lacks a zero-check on :
                - tokenOwner = _tokenOwner (JulPadToken.sol#1100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

Concluding Remarks

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

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

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 nor 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.