DeHacker

# Code Security Assessment

# BIDZ COIN

October 2nd, 2024

# Contents

# Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best prac tices. Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow/underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service/logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

# Issue Categories

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

## Critical severity issues

A vulnerability that can disrupt the contract functioning in a number of scenarios or creates a risk that the contract may be broken.

## Major severity issues

A vulnerability that affects the desired outcome when using a contract or provides the opportunity to use a contract in an unintended way.

## Medium severity issues

A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

## Minor severity issues

A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

## Informational

A vulnerability that has informational character but is not affecting any of the code.

# Overview

## Project Summary

| | |
|---|---|
| **Project Name** | BIDZ COIN |
| **Platform** | Ethereum |
| **Website** | bidz.store/index.html |
| **Type** | DeFi |
| **Language** | Solidity |
| **Codebase** | https://bscscan.com/address/0x20dE6118C3672659E488D1d45279CDF77391FBdc |

## Vulnerability Summary

| Vulnerability Level | Total | Mitigated | Declined | Acknowledged | Partially Resolved | Resolved |
|---|---|---|---|---|---|---|
| Critical | 0 | 0 | 0 | 0 | 0 | 0 |
| Major | 2 | 0 | 0 | 2 | 0 | 0 |
| Medium | 1 | 0 | 0 | 0 | 1 | 0 |
| Minor | 0 | 0 | 0 | 0 | 0 | 0 |
| Informational | 0 | 0 | 0 | 0 | 0 | 0 |
| Discussion | 0 | 0 | 0 | 0 | 0 | 0 |

## Audit scope

| ID | File | SHA256 Checksum |
|---|---|---|
| CTS | CoinToken.sol | 3e7155da4ae58183ad21539b2020e3ee040846ddbcadade5a95cec3116275aca |

# Findings

| ID | Title | Severity | Status |
|---|---|---|---|
| CTS-01 | Centralized Token Holding Position Risk | Major | Acknowledged |
| CTS-02 | Centralization Risks In CoinToken.Sol | Major | Acknowledged |
| CTS-03 | Missing Zero Address Validation InupdateLiquidity() Function | Medium | Partially Resolved |

# MAJOR

## CTS-01| Centralized Token Holding Position Risk

| Issue | Severity | Location | Status |
|---|---|---|---|
| Centralization / Privilege | Major | CoinToken.sol (v2) | Acknowledged |

### Description

As of September 19, 2024, the contract address 0x0C89C0407775dd89b12918B9c0aa42Bf96518820 has acquired ownership of over 70% of the total token supply for the token at address 0x20dE6118C3672659E488D1d45279CDF77391FBdc. A detailed inspection reveals that all these tokens are held in a locked state by the deployer. The deployer locked tokens in two transactions:

- 0xc8f93ee442572ca5d2f11220ac7d75b3704f914b92ba1179ae5695e53296f00c: 10 billion tokens are locked and will be unlocked on Jun 21, 2027.
- 0xd6c92047fbcaee55b55795b3d0bb92b0b3aa3c8516dd3ef3f4a5631e6b692cac: 900 million tokens are locked and will be unlocked on Jun 21, 2030.

This is a centralization risk because the address can distribute tokens without obtaining the consensus of the community once the locks mature. Any compromise to the addresses may allow a hacker to steal and sell tokens on the market, resulting in severe damage to the project.

### Recommendation

It is recommended that the team be transparent regarding the token distribution process. The token distribution plan should be published in a public location that the community can access. The team should make efforts to restrict access to the private keys of the deployer account or EOAs. A multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. Deanonymize the project team with a third-party KYC provider to create greater accountability.

# MAJOR

## CTS-02| Centralization Risks in CoinToken.sol

| Issue | Severity | Location | Status |
|---|---|---|---|
| Centralization / Privilege | Major | CoinToken.sol (v2): 1094, 1103, 1128, 1263, 1272, 1287, 1292 | Acknowledged |

### Description

Token address 0x20dE6118C3672659E488D1d45279CDF77391FBdc:
In the contract CoinToken, the role _owner has authority over the functions shown in the diagram below. Any compromise to the _owner account may allow the hacker to take advantage of this authority and mint tokens to an account, check account's liquidity status, finish the minting process, and set a transfer blacklist.

In the contract Ownable, the role _owner has authority over the functions shown in the diagram below. Any compromise to the _owner account may allow the hacker to take advantage of this authority and renounce contract ownership or transfer the ownership.

In the contract TokenRecover, the role _owner has authority over the functions shown in the diagram below. Any compromise to the _owner account may allow the hacker to take advantage of this authority and transfer ERC20 tokens from the contract.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

**Short Term:**

Timelock and Multi sign (⅔, ⅗) combination mitigate by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

**Long Term:**

Timelock and DAO, the combination, mitigate by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Recommendation

**Permanent:**

Renouncing the ownership or removing the function can be considered fully resolved.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

# MEDIUM

## CTS-03| Missing Zero Address Validation in updateLiquidity() Function

| Issue | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | Medium | CoinToken.sol (v2): 1287 | Partially Resolved |

## Description

In the CoinToken contract, the _liquidity mapping functions as a blacklist. If _liquidity[account] is set to true, the account is prevented from transferring or receiving tokens. The updateLiquidity() function, which can only be called by the _owner, updates the _liquidity list but does not include a check for the zero address. If the zero address (address(0)) is added to the _liquidity list, the minting and burning functions will fail to operate correctly.

```
modifier notLiquidity(address account) {
        require(!_liquidity[account], "Address is in array");
        require(!_liquidity[msg.sender],"Sender is in array");
        _;
    }


    function _beforeTokenTransfer(address from, address to, uint256 amount) internal overrid
notLiquidity(from) notLiquidity(to) {
        super._beforeTokenTransfer(from, to, amount);
    }

    function updateLiquidity(address account, bool liquidty) external onlyOwner {
        _liquidity[account] = liquidty;
        emit LiquidityUpdated(account, liquidty);
    }
```

## Recommendation

Add a zero address check in the updateLiquidity() function.

# Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

# Appendix

## Finding Categories

**Centralization / Privilege**
Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

**Coding Style**
Coding Style findings usually do not affect the generated bytecode but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

**Volatile Code**
Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

**Logical Issue**
Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block. timestamp works.

## Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux “sha256sum" command against the target file.

# About

DeHacker is a team of auditors and white hat hackers who perform security audits and assessments. With decades of experience in security and distributed systems, our experts focus on the ins and outs of system security. Our services follow clear and prudent industry standards. Whether it's reviewing the smallest modifications or a new platform, we'll provide an in-depth security survey at every stage of your company's project. We provide comprehensive vulnerability reports and identify structural inefficiencies in smart contract code, combining high-end security research with a real-world attacker mindset to reduce risk and harden code.

### BLOCKCHAIINS

- Ethereum
- Cosmos
- Eos
- Substrate

### TECH STACK

- Python
- Solidity
- Rust
- c++

### CONTACTS

- https://dehacker.io
- https://twitter.com/dehackerio
- https://github.com/dehacker/audits_public
- https://t.me/dehackerio
- https://blog.dehacker.io/


October 2nd 2024