QuillAudits

Audit Report
April, 2023

For

indu4.0

Table of Content

Executive Summary

Project Name	Indu4.0
Overview	Indu40 is an ERC20 token contract that inherits from the openzeppelin library; ERC20Burnable, ERC20Capped, Ownable, and ERC20Permit. The contract allows the owner to mint tokens until it meets the set cap.
Project URL	*https://www.indu40.io/*
Timeline	19th April, 2023 to 21st April, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze Indu40 codebase for quality, security, and correctness.
Contracts in Scope	*https://github.com/indu40/indu-erc-20-token-contract/blob/main/contracts/Indu40.sol* Branch: Main Commit hash: 1278a41c8343bbe27d552d2838911c8daada16a9

High Medium

Low Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- BEP20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - Indu40.sol

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

No issues were found

Functional Testing

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

```
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#117)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#126)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#105)
        - result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Contract locking ether found:
        Contract Indu40 (contracts/Indu4.sol#20-45) has payable functions:
         - Indu40.constructor(address,uint256,uint256) (contracts/Indu4.sol#24-32)
        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:
ERC20Permit.constructor(string).name (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#44) shadows:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64) (function)
        - IERC20Metadata.name() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#17) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#49-68)
 uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20P
ermit.sol#58)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Strings.toString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#18-38) uses assembly
```

```
        - ^0.8.19 (contracts/Indu4.sol#2)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Counters.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Counters.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#4) allows old versions
Pragma version0.8.19 (contracts/Indu4.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function ERC20Permit.DOMAIN_SEPARATOR() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#81-83) is not in mixedCase
Variable ERC20Permit._PERMIT_TYPEHASH_DEPRECATED_SLOT (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#37) is not in mixedCase
Function IERC20Permit.DOMAIN_SEPARATOR() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol#59) is not in mixedCase
Variable EIP712._CACHED_DOMAIN_SEPARATOR (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#31) is not in mixedCase
Variable EIP712._CACHED_CHAIN_ID (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#32) is not in mixedCase
Variable EIP712._CACHED_THIS (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#33) is not in mixedCase
Variable EIP712._HASHED_NAME (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#35) is not in mixedCase
Variable EIP712._HASHED_VERSION (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#36) is not in mixedCase
Variable EIP712._TYPE_HASH (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#37) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:. analyzed (15 contracts with 85 detectors), 40 result(s) found
```

Functional Testing

Some of the tests performed are mentioned below:

- Should get the name of the token
- Should get the symbol of the token
- Should get the decimal of the token
- Should get the capped volume size of the token contract.
- Should allow the contract owner to mint any amount of token to a recipient.
- Should fail when minting after the capped size has been met.
- Should allow token holders burn their tokens.
- Should allow spender burn allowed tokens with burnFrom function.

Closing Summary

In this report, we have considered the security of the Indu40 codebase. We performed our audit according to the procedure described above.

No Issues were Found During the Course of the Audit.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Indu40 Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Indu40 Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

700+
Audits Completed

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
April, 2023

For

indu4.0

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com