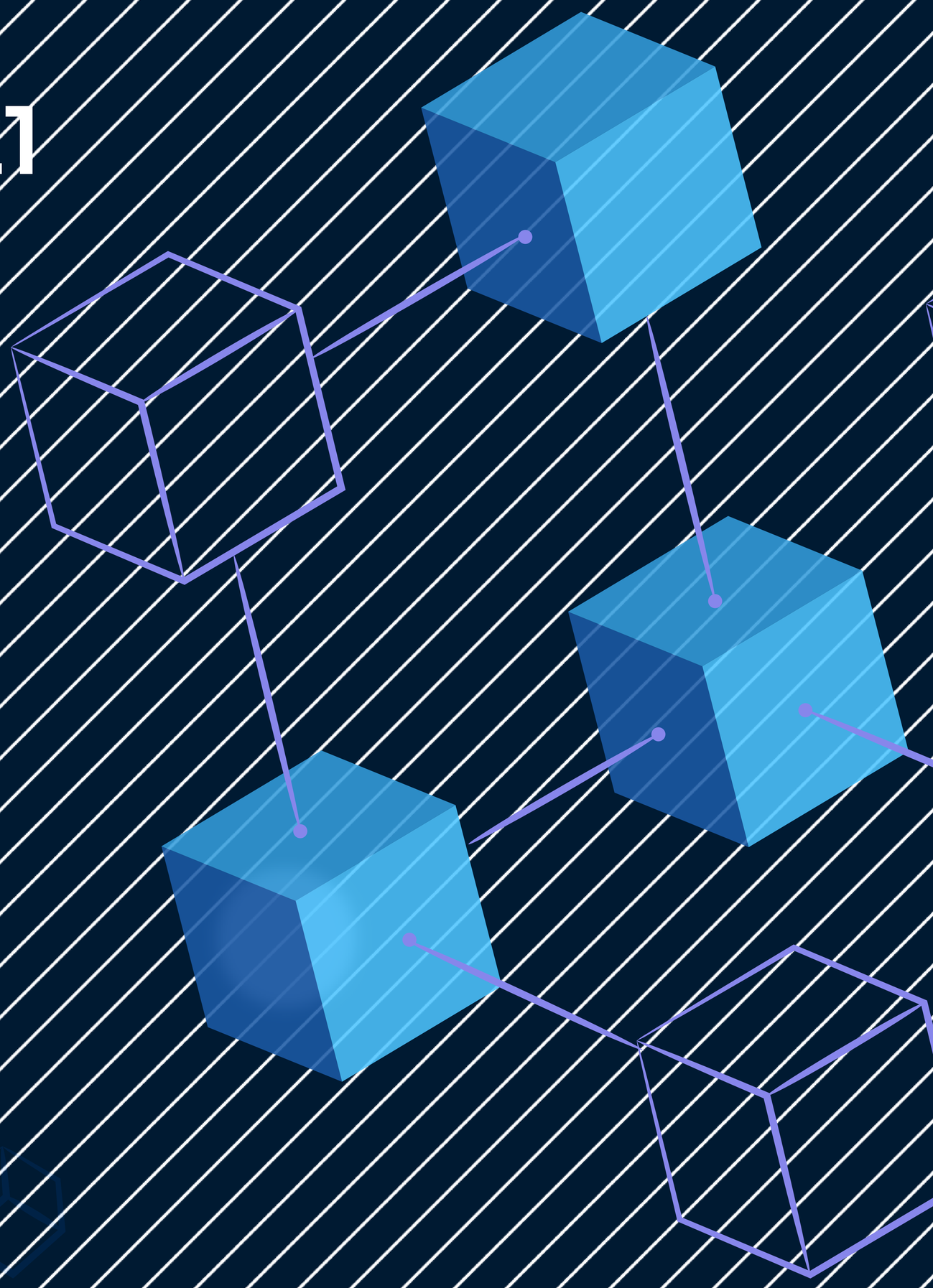
QuillAudits

Audit Report
November, 2021



For

CoinPoker

Contents

Scope of the Audit

The scope of this audit was to analyze and document the CoinPoker Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- 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
- ERC20 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, Theo.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
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.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	3	3
Closed	0	0	0	0

Introduction

During the period of **November 17, 2021, to November 24, 2021** - QuillAudits Team performed a security audit for **CoinPoken** smart contracts.

The code for the audit was taken from following the official link: https://github.com/CoinPokerOfficial/new-chp-smartcontract

Note	Date	Commit hash	Files
Version 1	November	307a05c78daa845bc4a4 902836c06f834b10f468	3

Issues Found

A. Contract – CoinPokerChipsToken

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

1. **Unnecessary import of ERC20, IERC20, SafeMath**

Line	Code
5, 6, 8	import "@openzeppelin/contracts/token/ERC20/ERC20.sol"; import "@openzeppelin/contracts/token/ERC20/IERC20.sol"; import "@openzeppelin/contracts/math/SafeMath.sol";

Description
IERC20 and SafeMath are already imported in SafeERC20. Also, we are importing and using SafeERC20, so importing ERC20 is unnecessary. These imports will increase the stack size of the contract.

Remediation
We recommend importing only SafeERC20 and removing IERC20, ERC20, and SafeMath

Status: Acknowledged

2. No need to index the amount param

Line	Code
26	event RecoverToken(address indexed token, address indexed destination, uint256 indexed amount);

Description
The `amount` parameter in the RecoverToken event will not be used to search for the transaction event because the value will not be unique for that transaction log.

Remediation
Remove the `indexed` keyword for `amount` parameter.

Status: Acknowledged

3. setGovernance method can be made external

Line	Code

43-45	function setGovernance(address _governance) public onlyGovernance {
	governance = _governance;
	}

Description
Function setGovernance method can be made external if we are not going to import the `CoinPokerChipsToken` contract.

Remediation
In all public functions, EVM immediately copies array arguments to memory, while external functions can read directly from call data. Memory allocation is expensive, whereas reading from call data is cheap.

We recommend making it external to save gas costs.

Status: Acknowledged

Informational issues

4. Insufficient error message

Line	Code
59	require(token != destination, "Invalid address");
60	require(IERC20(token).transfer(destination, amount), "Retrieve failed");

Description
The above error messages do not describe the error correctly.

Remediation
Change the error messages to something that describes the error better, and follow the correct guide. Eg. `Contract Name: Revert message`

Status: Acknowledged

5. Need to add comments on every method and state variable

Description
Add comments on the following methods and state variables-
- MAX_CAP
- governance
- onlyGovernance()
- constructor()
- _nonces

Remediation
Add comments on the above-mentioned methods and state variables

Status: Acknowledged

6. Follow correct Natspec for all the functions params

Line	Code
44-58	function recoverToken(address token, address destination, uint256 amount)

Description
The function parameters of the `recoverToken` method aren’t following the correct Natspec.

Remediation
We recommend you to add `_`(underscore) before the param name, similar to the `setGovernance` method.

Status: Acknowledged

Functional test

Contract Address:
0x6d4B557cFf108f22F3C37911b96E1f8aEc6dBE98

Transaction
setGovernance:
0x1e078ee25d1a8eee8bc9d4622295c3ef72cb4a9d3c7422947176a998569e2823
recoverToken:
0x53c4dcd2de4a8925a4e04f9cf241e76199f204d3e62fa6c2a4368ef6c151a73f

Automated Tests

Slither

```
'npx hardhat compile --force' running
Compiling 13 files with 0.7.4
Compilation finished successfully
Creating Typechain artifacts in directory typechain for target ethers-v5
Successfully generated Typechain artifacts!

@openzeppelin/contracts/access/Ownable.sol:26:5: Warning: Visibility for constructor is ignored. If you want the contract to be non-deployable, making it "abstract" is sufficient.
    constructor () internal {
    ^ (Relevant source part starts here and spans across multiple lines).

@openzeppelin/contracts/token/ERC20/ERC20.sol:55:5: Warning: Visibility for constructor is ignored. If you want the contract to be non-deployable, making it "abstract" is sufficient.
    constructor (string memory name_, string memory symbol_) public {
    ^ (Relevant source part starts here and spans across multiple lines).

contracts/ERC20Permit.sol:19:5: Warning: Visibility for constructor is ignored. If you want the contract to be non-deployable, making it "abstract" is sufficient.
    constructor() internal {
    ^ (Relevant source part starts here and spans across multiple lines).



CoinPokerChipsToken.setGovernance(address) (contracts/CoinPokerChipsToken.sol#43-45) should emit an event for:
        - governance = _governance (contracts/CoinPokerChipsToken.sol#44)
MockTetherToken.setGovernance(address) (contracts/mock/MockTetherToken.sol#30-32) should emit an event for:
        - governance = _governance (contracts/mock/MockTetherToken.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

CoinPokerChipsToken.setGovernance(address)._governance (contracts/CoinPokerChipsToken.sol#43) lacks a zero-check on :
                - governance = _governance (contracts/CoinPokerChipsToken.sol#44)
MockTetherToken.setGovernance(address)._governance (contracts/mock/MockTetherToken.sol#30) lacks a zero-check on :
                - governance = _governance (contracts/mock/MockTetherToken.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in CoinPokerChipsToken.recoverToken(address,address,uint256) (contracts/CoinPokerChipsToken.sol#54-62):
        External calls:
        - require(bool,string)(IERC20(token).transfer(destination,amount),Retrieve failed) (contracts/CoinPokerChipsToken.sol#60)
        Event emitted after the call(s):
        - RecoverToken(token,destination,amount) (contracts/CoinPokerChipsToken.sol#61)
Reentrancy in MockTetherToken.recoverToken(address,address,uint256) (contracts/mock/MockTetherToken.sol#34-42):
        External calls:
        - require(bool,string)(IERC20(token).transfer(destination,amount),Retrieve failed) (contracts/mock/MockTetherToken.sol#40)
        Event emitted after the call(s):
        - RecoverToken(token,destination,amount) (contracts/mock/MockTetherToken.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (contracts/ERC20Permit.sol#40-62) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,Permit: expired deadline) (contracts/ERC20Permit.sol#49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/Address.sol#26-35) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#33)
Address._verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#147-164) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#156-159)
ERC20Permit.constructor() (contracts/ERC20Permit.sol#19-34) uses assembly
        - INLINE ASM (contracts/ERC20Permit.sol#21-23)
```

```
ERC20Permit.constructor() (contracts/ERC20Permit.sol#19-34) uses assembly
        - INLINE ASM (contracts/ERC20Permit.sol#21-23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity is used:
        - Version used: ['0.7.4', '>=0.6.0<0.8.0', '>=0.6.2<0.8.0', '^0.7.0']
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/GSN/Context.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/math/SafeMath.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20Burnable.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/SafeERC20.sol#3)
        - >=0.6.2<0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/Counters.sol#3)
        - 0.7.4 (contracts/CoinPokerChipsToken.sol#3)
        - 0.7.4 (contracts/ERC20Permit.sol#3)
        - 0.7.4 (contracts/IERC2612Permit.sol#3)
        - ^0.7.0 (contracts/mock/MockTetherToken.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/GSN/Context.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/math/SafeMath.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20Burnable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/SafeERC20.sol#3) is too complex
Pragma version>=0.6.2<0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/Counters.sol#3) is too complex
Pragma version0.7.4 (contracts/CoinPokerChipsToken.sol#3) allows old versions
Pragma version0.7.4 (contracts/ERC20Permit.sol#3) allows old versions
Pragma version0.7.4 (contracts/IERC2612Permit.sol#3) allows old versions
Pragma version^0.7.0 (contracts/mock/MockTetherToken.sol#3) allows old versions
solc-0.7.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#53-59):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-121):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#139-145):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#143)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter CoinPokerChipsToken.setGovernance(address)._governance (contracts/CoinPokerChipsToken.sol#43) is not in mixedCase
Variable ERC20Permit.DOMAIN_SEPARATOR (contracts/ERC20Permit.sol#17) is not in mixedCase
Parameter MockTetherToken.setGovernance(address)._governance (contracts/mock/MockTetherToken.sol#30) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (node_modules/@openzeppelin/contracts/GSN/Context.sol#21)" inContext (node_modules/@openzeppelin/contracts/GSN/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

ERC20.constructor(string,string) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#55-61) uses literals with too many digits:
        - _mint(msg.sender,10000000000000000000000000000000000) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#60)
```

```
ERC20.constructor(string,string) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#55-61) uses literals with too many digits:
        - _mint(msg.sender,10000000000000000000000000000000) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#60)
MockTetherToken.slitherConstructorVariables() (contracts/mock/MockTetherToken.sol#11-43) uses literals with too many digits:
        - tokenSupply = 10000000000 * (10 ** 18) (contracts/mock/MockTetherToken.sol#12)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

MockTetherToken.tokenSupply (contracts/mock/MockTetherToken.sol#12) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

owner() should be declared external:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#35-37)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#54-57)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#63-67)
symbol() should be declared external:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#74-76)
decimals() should be declared external:
        - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#91-93)
totalSupply() should be declared external:
        - ERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#98-100)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#105-107)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#117-120)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#136-139)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#154-158)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#172-175)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#191-194)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20Burnable.sol#21-23)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20Burnable.sol#36-41)
setGovernance(address) should be declared external:
        - CoinPokerChipsToken.setGovernance(address) (contracts/CoinPokerChipsToken.sol#43-45)
permit(address,address,uint256,uint256,uint8,bytes32,bytes32) should be declared external:
        - ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (contracts/ERC20Permit.sol#40-62)
nonces(address) should be declared external:
        - ERC20Permit.nonces(address) (contracts/ERC20Permit.sol#67-69)
setGovernance(address) should be declared external:
        - MockTetherToken.setGovernance(address) (contracts/mock/MockTetherToken.sol#30-32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
. analyzed (13 contracts with 75 detectors), 53 result(s) found
```

Results

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

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract, but relying on other contracts might cause Reentrancy Vulnerability.

3 Low Severity issues were discovered during the audit, which has been Acknowledged by the CoinPokerOfficial Team.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **CoinPokerOfficial** 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 **CoinPokerOfficial** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
November, 2021

For

CoinPoker

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com