QuillAudits

Audit Report
April, 2023

For

Table of Content

Executive Summary

Project Name	Petdia
Overview	The Petdia project contains a single token contract that inherits all standard ERC20 functionality. It mints a total of 3 million tokens to the token creator's address only.
Timeline	21 April, 2023 to 27 April, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze Petdia codebase for quality, security, and correctness.
Contracts in Scope	Petdia.sol *https://drive.google.com/file/d/1aSSH6CeBzcJKh30BbAMd27ATc0EtYN9M/view?usp=share_link*
Fixed In	*https://drive.google.com/file/d/1GPBIZOxH_kbW7Bec2bAPwtg-u2xW8WaI/view?usp=share_link*

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	1	0	1

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

High Severity Issues

No issues were found

Medium Severity Issues

1. Token mint irregularity

Description

The internal mint function is called in the token's constructor to mint a certain number of tokens (3000000000 * 18**18) tokens to the deployer of the contract. This does not match up with the standard value of 10**18 used to denote decimals for ease of transfer and reduction of arithmetic errors.

Recommendation

If this is the intended functionality, include the non-custom step implemented in the documentation, and clearly specify to users the methodology behind this approach.

Status

Resolved

Auditor's Remark: The devs have fixed the mint function with the decimals issue.

Low Severity Issues

2. Privilege and Ownership transfer

Description

The contract does not make use of the Ownable inheritance and can be removed to save deployment costs and reduce unused code.

Recommendation

Remove the Ownable contract if it is to be unused but if it is to be used, it is advised to make ownership and privilege transfer a two-step process OR override the transferOwnership( ) function if it will not be implemented in the scope of this contract. Also, ensure proper management on the owner private key to prevent compromise.

References: Link 1 | Link 2

Status

Acknowledged

Informational Issues

3. Unlocked pragma (pragma solidity >=0.8.0)

Description

Contracts should be deployed with the same compiler version that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation

It is recommended to lock the token contract to a specific version.

Status

Resolved

Functional Testing

Some of the tests performed are mentioned below:

- Should not mint new tokens when inherited
- Should mint 3,000,000,000 tokens when deployed
- Should mint 3,000,000,000 *10**18 tokens
- Should transfer tokens to other users
- Should renounce ownership of contract
- Should transfer ownership of the contract
- Should transfer tokens after ownership of the contract has been renounced

Automated Tests

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.

```
Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.9']
        - ^0.8.0 (Petdia.sol#3)
        - ^0.8.0 (Petdia.sol#15)
        - ^0.8.0 (Petdia.sol#56)
        - ^0.8.0 (Petdia.sol#81)
        - ^0.8.0 (Petdia.sol#100)
        - ^0.8.9 (Petdia.sol#285)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (Petdia.sol#10-12) is never used and should be removed
ERC20._burn(address,uint256) (Petdia.sol#226-242) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (Petdia.sol#3) allows old versions
Pragma version^0.8.0 (Petdia.sol#15) allows old versions
Pragma version^0.8.0 (Petdia.sol#56) allows old versions
Pragma version^0.8.0 (Petdia.sol#81) allows old versions
Pragma version^0.8.0 (Petdia.sol#100) allows old versions
Pragma version^0.8.9 (Petdia.sol#285) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Token.constructor() (Petdia.sol#288-290) uses literals with too many digits:
        - _mint(msg.sender,3000000000 * 18 ** decimals()) (Petdia.sol#289)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Petdia.sol#39-41)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Petdia.sol#44-47)
name() should be declared external:
        - ERC20.name() (Petdia.sol#117-119)
symbol() should be declared external:
        - ERC20.symbol() (Petdia.sol#121-123)
totalSupply() should be declared external:
        - ERC20.totalSupply() (Petdia.sol#132-134)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (Petdia.sol#139-141)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (Petdia.sol#143-147)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (Petdia.sol#153-157)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (Petdia.sol#159-168)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (Petdia.sol#170-174)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (Petdia.sol#176-185)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Closing Summary

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

Some issues of Medium, Low and Informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Petdia Token. 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 Petdia Token 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

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com