EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	Pekker Token
Website:	https://pekker.org
Platform:	Ethereum
Language:	Solidity
Date:	June 19th, 2021

Table of contents

Introduction

We were contracted by the Pekker token team to perform the Security audit of the Pekker Token smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on June 19th, 2021.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

Pekker is a community-focused decentralized transaction network. It is an Ethereum Token with a burn and redistribution system.

Audit scope

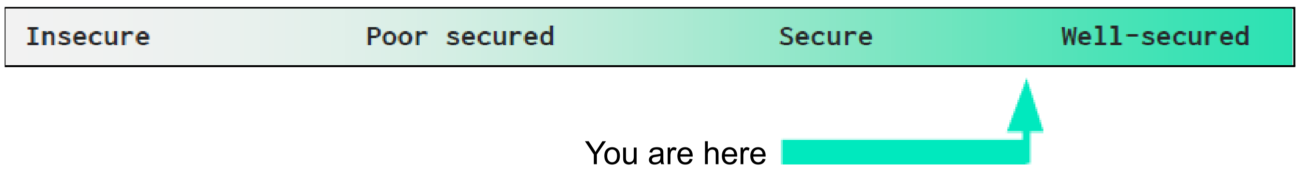
Name	**Code Review and Security Analysis Report for Pekker Token Smart Contract**
Platform	**Etherscan / Solidity**
File	PekkerToken.sol
Smart Contract Online Code	https://etherscan.io/address/0x3c0a9d9b70a253b47443470a28e2d8422b9c882f#code#L1
File MD5 Hash	E7529B37884DF3006CEEA31F75F00DE2
Audit Date	June 19th, 2021

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Name: Pekker Token	**YES, This is valid.**
Symbol: PKR	**YES, This is valid.**
Decimal: 9	**YES, This is valid.**
MaxTxAmount: 100000000000000	**YES, This is valid. Owner can change this fee.**
The owner can access functions like setMaxTxPercent, excludeAccount, includeAccount.	**YES, This is valid. The smart contract owner controls these functions, so the owner must handle the private key of the owner's wallet very securely. Because if the private key is compromised, then it will create problems.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contract is **secured**. These contracts also have owner functions (described in the centralization section below), which does not make everything 100% decentralized. Thus, the owner must execute those smart contract functions as per the business plan.



We found 0 critical, 0 high, 0 medium and 3 low and some very low level issues.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Moderated
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract. This smart contract also contains Libraries, Smart contracts inherits and Interfaces. This is a compact and well written contract.

The libraries in the Pekker Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the PekkerToken.

The Pekker Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Some code parts are **not well** commented on smart contracts.

Documentation

We were given Pekker Token smart contract code in the form of an Etherscan web link. The hashes of that code are mentioned above in the table.

As mentioned above, some code parts are **not well** commented. So it is difficult to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website https://www.pekker.org which provided rich information about the project architecture and tokenomics.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects. And their core code blocks are written well.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Pekker token is a smart contract, having functionality like burn, redistribution, etc.

PakkerToken.sol

(1) Interface

(a) IERC20

(b) IERC20Metadata

(2) Inherited contracts

(a) Context

(b) IERC20

(c) IERC20Metadata

(d) Ownable

(3) Functions

Sl.	Functions	Type	Observation	Conclusion
1	name	read	Passed	No Issue
2	symbol	read	Passed	No Issue
3	decimals	read	Passed	No Issue
4	totalSupply	read	Function state mutability	compiler warning
5	balanceOf	read	Passed	No Issue
6	transfer	write	Passed	No Issue
7	allowance	read	Passed	No Issue
8	approve	write	Passed	No Issue
9	transferFrom	write	Passed	No Issue
10	increaseAllowance	write	Passed	No Issue
11	decreaseAllowance	write	Passed	No Issue
12	isExcluded	read	Passed	No Issue
13	totalFees	write	Passed	No Issue
14	setMaxTxPercent	external	Missing Events	No Issue
15	reflect	write	Passed	No Issue
16	reflectionFromToken	read	Passed	No Issue
17	tokenFromReflection	read	Passed	No Issue
18	excludeAccount	external	Missing Events	No Issue
19	includeAccount	external	Infinite loop possibility	Refer Audit Findings
20	_approve	write	Passed	No Issue

21	_transfer	write	Passed	No Issue
22	_transferStandard	write	Passed	No Issue
23	_transferToExcluded	write	Passed	No Issue
24	_transferFromExcluded	write	Passed	No Issue
25	_transferBothExcluded	write	Passed	No Issue
26	_reflectFee	write	Passed	No Issue
27	_getValues	read	Passed	No Issue
28	_getTValues	write	Passed	No Issue
29	_getRValues	write	Passed	No Issue
30	_getRate	read	Passed	No Issue
31	_getCurrentSupply	read	Passed	No Issue
32	owner	read	Passed	No Issue
33	onlyOwner	modifier	Passed	No Issue
34	renounceOwnership	write	access only Owner	No Issue
35	transferOwnership	write	access only Owner	No Issue
36	_msgSender	internal	Passed	No Issue
37	_msgData	internal	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical

No critical severity vulnerabilities were found.

High

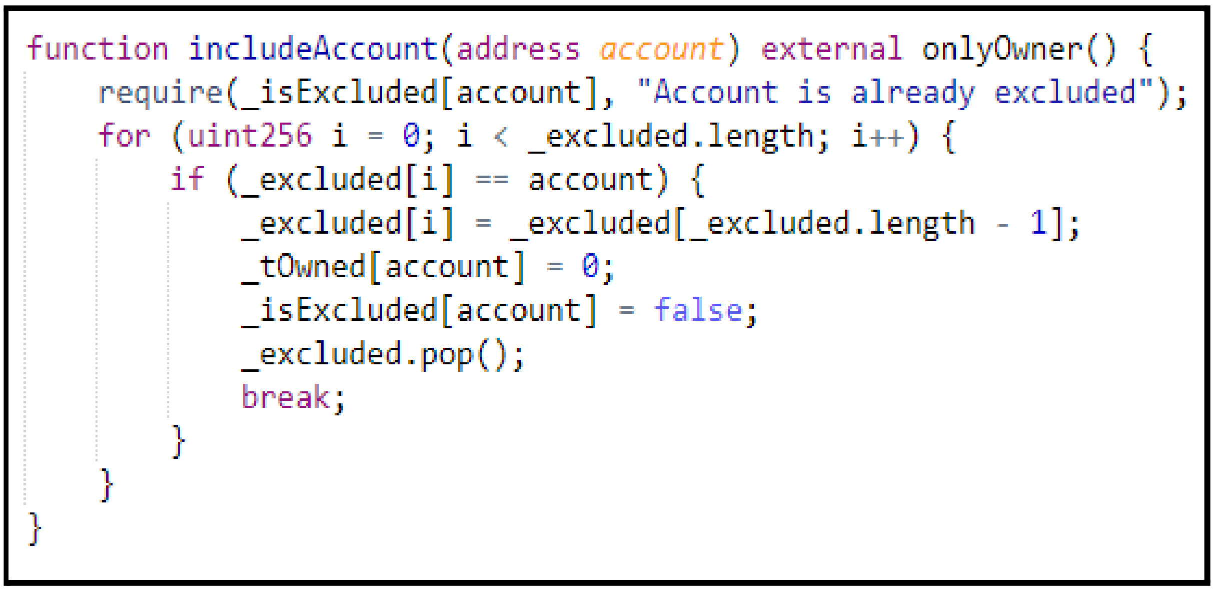
No high severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Infinite loop possibility:

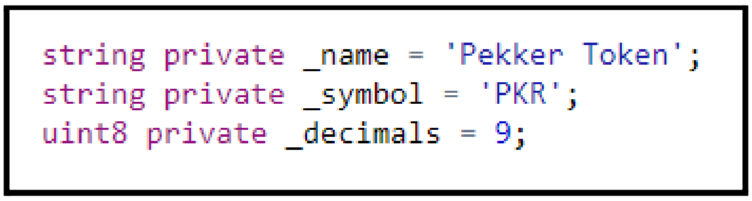


```
function includeAccount(address account) external onlyOwner() {
    require(_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

If there are so many excluded wallets, then this logic will fail, as it might hit the block's gas limit. If there are very limited exceptions, then this will work, but will cost more gas.

<u>Resolution</u>: We suggest excluding limited wallets only.

(2) Make variables constant:



```
string private _name = 'Pekker Token';
string private _symbol = 'PKR';
uint8 private _decimals = 9;
```

Following variables will be unchanged. So, please make it constant. It will save some gas.

- name
- symbol
- decimals

<u>Resolution</u>: Declare those variables as constant. Just put a constant keyword.

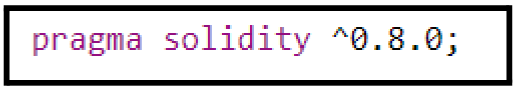
(3) Missing Events:

Missing Events log for some functions:

- excludeAccount
- includeAccount
- setMaxTxPercent

Very Low / Discussion / Best practices:

(1) Use latest solidity version:



```
pragma solidity ^0.8.0;
```

Using the latest solidity will prevent any compiler level bugs.

Resolution: Please use 0.8.5 which is the latest version.

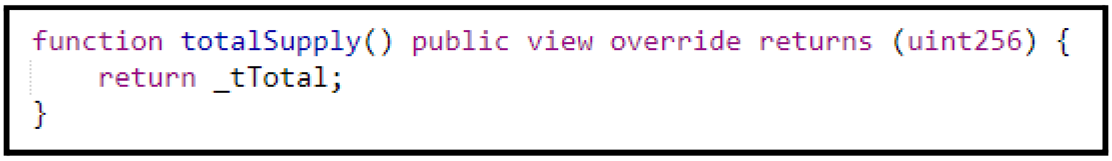
(2) external instead of public:

If any function is not called from inside the smart contract, then it is better to declare it as external instead of public. As it saves some gas as well.

https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

Compiler Warning

(1) Function state mutability:



```
function totalSupply() public view override returns (uint256) {
    return _tTotal;
}
```

Resolution: Function state mutability can be restricted to pure.

Centralization

This smart contract has some functions which can be executed by Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- setMaxTxPercent: Owner can set maximum TAX percentage.
- excludeAccount: Owner can exclude an account.
- includeAccount: Owner can include an account.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues in the smart contracts and those are fixed/acknowledged in the smart contracts. **So it is good to go for the production.**

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract, based on standard audit procedure scope, is **" Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:
In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:
Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

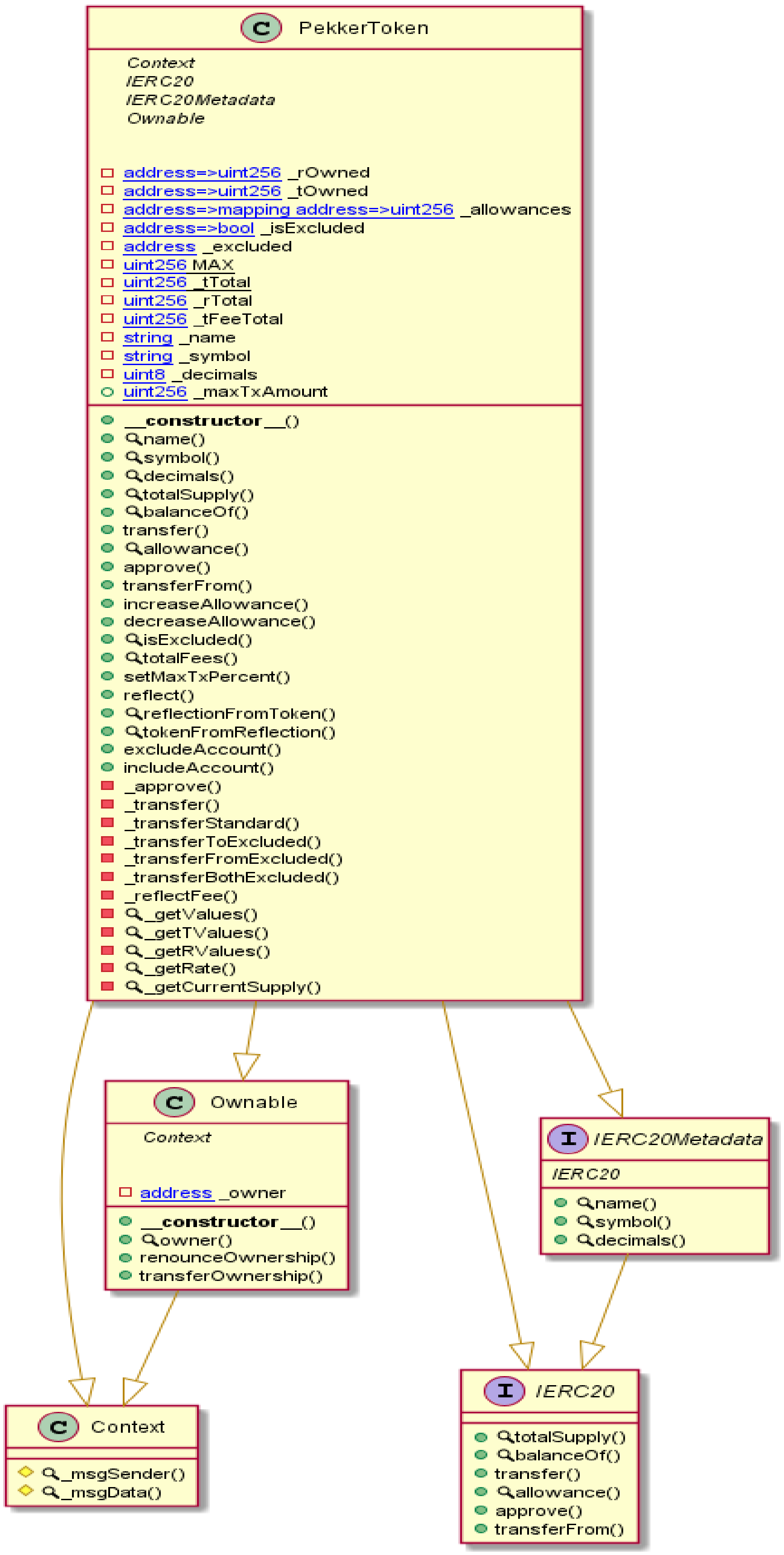
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Pekker Token



Slither Results Log

Slither log >> PekkerToken.sol

```
INFO:Detectors:
PekkerToken._getTValues(uint256) (Pekker.sol#413-417) performs a multiplication on the result of a division:
	-tFee = ((tAmount / 100) * 2) (Pekker.sol#414)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
PekkerToken.allowance(address,address).owner (Pekker.sol#247) shadows:
	- Ownable.owner() (Pekker.sol#160-162) (function)
PekkerToken._approve(address,address,uint256).owner (Pekker.sol#335) shadows:
	- Ownable.owner() (Pekker.sol#160-162) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Context._msgData() (Pekker.sol#28-31) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (Pekker.sol#11) necessitates a version too recent to be trusted. Consider deploying
with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Constant PekkerToken._tTotal (Pekker.sol#206) is not in UPPER_CASE_WITH_UNDERSCORES
Variable PekkerToken._maxTxAmount (Pekker.sol#214) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (Pekker.sol#29)" inContext (Pekker.sol#23-32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable PekkerToken._getRValues(uint256,uint256,uint256).rTransferAmount (Pekker.sol#422) is too similar
to PekkerToken._getValues(uint256).tTransferAmount (Pekker.sol#407)
Variable PekkerToken.reflectionFromToken(uint256,bool).rTransferAmount (Pekker.sol#302) is too similar to
PekkerToken._getTValues(uint256).tTransferAmount (Pekker.sol#415)
Variable PekkerToken._transferStandard(address,address,uint256).rTransferAmount (Pekker.sol#366) is too
similar to PekkerToken._transferToExcluded(address,address,uint256).tTransferAmount (Pekker.sol#374)
Variable PekkerToken._getValues(uint256).rTransferAmount (Pekker.sol#409) is too similar to
PekkerToken._transferToExcluded(address,address,uint256).tTransferAmount (Pekker.sol#374)
Variable PekkerToken._transferBothExcluded(address,address,uint256).rTransferAmount (Pekker.sol#392) is
too similar to PekkerToken._transferBothExcluded(address,address,uint256).tTransferAmount
(Pekker.sol#392)
Variable PekkerToken._getRValues(uint256,uint256,uint256).rTransferAmount (Pekker.sol#422) is too similar
to PekkerToken._transferToExcluded(address,address,uint256).tTransferAmount (Pekker.sol#374)
Variable PekkerToken._transferFromExcluded(address,address,uint256).rTransferAmount (Pekker.sol#383)
is too similar to PekkerToken._transferFromExcluded(address,address,uint256).tTransferAmount
(Pekker.sol#383)
Variable PekkerToken._transferToExcluded(address,address,uint256).rTransferAmount (Pekker.sol#374) is
too similar to PekkerToken._transferToExcluded(address,address,uint256).tTransferAmount (Pekker.sol#374)
Variable PekkerToken.reflectionFromToken(uint256,bool).rTransferAmount (Pekker.sol#302) is too similar to
PekkerToken._transferStandard(address,address,uint256).tTransferAmount (Pekker.sol#366)
Variable PekkerToken._transferBothExcluded(address,address,uint256).rTransferAmount (Pekker.sol#392) is
too similar to PekkerToken._transferFromExcluded(address,address,uint256).tTransferAmount
(Pekker.sol#383)
Variable PekkerToken.reflectionFromToken(uint256,bool).rTransferAmount (Pekker.sol#302) is too similar to
PekkerToken._transferFromExcluded(address,address,uint256).tTransferAmount (Pekker.sol#383)
Variable PekkerToken._transferBothExcluded(address,address,uint256).rTransferAmount (Pekker.sol#392) is
too similar to PekkerToken._transferStandard(address,address,uint256).tTransferAmount (Pekker.sol#366)
Variable PekkerToken._transferStandard(address,address,uint256).rTransferAmount (Pekker.sol#366) is too
similar to PekkerToken._getValues(uint256).tTransferAmount (Pekker.sol#407)
```

```
Variable PekkerToken.reflectionFromToken(uint256,bool).rTransferAmount (Pekker.sol#302) is too similar to
PekkerToken._transferBothExcluded(address,address,uint256).tTransferAmount (Pekker.sol#392)
Variable PekkerToken._getRValues(uint256,uint256,uint256).rTransferAmount (Pekker.sol#422) is too similar
to PekkerToken._transferFromExcluded(address,address,uint256).tTransferAmount (Pekker.sol#383)
Variable PekkerToken._transferFromExcluded(address,address,uint256).rTransferAmount (Pekker.sol#383)
is too similar to PekkerToken._transferToExcluded(address,address,uint256).tTransferAmount
(Pekker.sol#374)
Variable PekkerToken._getRValues(uint256,uint256,uint256).rTransferAmount (Pekker.sol#422) is too similar
to PekkerToken._getTValues(uint256).tTransferAmount (Pekker.sol#415)
Variable PekkerToken._transferFromExcluded(address,address,uint256).rTransferAmount (Pekker.sol#383)
is too similar to PekkerToken._getValues(uint256).tTransferAmount (Pekker.sol#407)
Variable PekkerToken._getRValues(uint256,uint256,uint256).rTransferAmount (Pekker.sol#422) is too similar
to PekkerToken._transferBothExcluded(address,address,uint256).tTransferAmount (Pekker.sol#392)
Variable PekkerToken._getRValues(uint256,uint256,uint256).rTransferAmount (Pekker.sol#422) is too similar
to PekkerToken._transferStandard(address,address,uint256).tTransferAmount (Pekker.sol#366)
Variable PekkerToken._transferBothExcluded(address,address,uint256).rTransferAmount (Pekker.sol#392) is
too similar to PekkerToken._getValues(uint256).tTransferAmount (Pekker.sol#407)
Variable PekkerToken._transferStandard(address,address,uint256).rTransferAmount (Pekker.sol#366) is too
similar to PekkerToken._transferStandard(address,address,uint256).tTransferAmount (Pekker.sol#366)
Variable PekkerToken._transferBothExcluded(address,address,uint256).rTransferAmount (Pekker.sol#392) is
too similar to PekkerToken._transferToExcluded(address,address,uint256).tTransferAmount (Pekker.sol#374)
Variable PekkerToken._transferFromExcluded(address,address,uint256).rTransferAmount (Pekker.sol#383)
is too similar to PekkerToken._transferStandard(address,address,uint256).tTransferAmount (Pekker.sol#366)
Variable PekkerToken.reflectionFromToken(uint256,bool).rTransferAmount (Pekker.sol#302) is too similar to
PekkerToken._transferToExcluded(address,address,uint256).tTransferAmount (Pekker.sol#374)
Variable PekkerToken.reflectionFromToken(uint256,bool).rTransferAmount (Pekker.sol#302) is too similar to
PekkerToken._getValues(uint256).tTransferAmount (Pekker.sol#407)
Variable PekkerToken._transferStandard(address,address,uint256).rTransferAmount (Pekker.sol#366) is too
similar to PekkerToken._getTValues(uint256).tTransferAmount (Pekker.sol#415)
Variable PekkerToken._getValues(uint256).rTransferAmount (Pekker.sol#409) is too similar to
PekkerToken._getTValues(uint256).tTransferAmount (Pekker.sol#415)
Variable PekkerToken._transferToExcluded(address,address,uint256).rTransferAmount (Pekker.sol#374) is
too similar to PekkerToken._transferBothExcluded(address,address,uint256).tTransferAmount
(Pekker.sol#392)
Variable PekkerToken._getValues(uint256).rTransferAmount (Pekker.sol#409) is too similar to
PekkerToken._getValues(uint256).tTransferAmount (Pekker.sol#407)
Variable PekkerToken._transferToExcluded(address,address,uint256).rTransferAmount (Pekker.sol#374) is
too similar to PekkerToken._getTValues(uint256).tTransferAmount (Pekker.sol#415)
Variable PekkerToken._transferToExcluded(address,address,uint256).rTransferAmount (Pekker.sol#374) is
too similar to PekkerToken._transferFromExcluded(address,address,uint256).tTransferAmount
(Pekker.sol#383)
Variable PekkerToken._getValues(uint256).rTransferAmount (Pekker.sol#409) is too similar to
PekkerToken._transferStandard(address,address,uint256).tTransferAmount (Pekker.sol#366)
Variable PekkerToken._transferFromExcluded(address,address,uint256).rTransferAmount (Pekker.sol#383)
is too similar to PekkerToken._getTValues(uint256).tTransferAmount (Pekker.sol#415)
Variable PekkerToken._transferToExcluded(address,address,uint256).rTransferAmount (Pekker.sol#374) is
too similar to PekkerToken._transferStandard(address,address,uint256).tTransferAmount (Pekker.sol#366)
Variable PekkerToken._transferToExcluded(address,address,uint256).rTransferAmount (Pekker.sol#374) is
too similar to PekkerToken._getValues(uint256).tTransferAmount (Pekker.sol#407)
Variable PekkerToken._transferFromExcluded(address,address,uint256).rTransferAmount (Pekker.sol#383)
is too similar to PekkerToken._transferBothExcluded(address,address,uint256).tTransferAmount
(Pekker.sol#392)
Variable PekkerToken._transferStandard(address,address,uint256).rTransferAmount (Pekker.sol#366) is too
similar to PekkerToken._transferBothExcluded(address,address,uint256).tTransferAmount (Pekker.sol#392)
Variable PekkerToken._getValues(uint256).rTransferAmount (Pekker.sol#409) is too similar to
PekkerToken._transferFromExcluded(address,address,uint256).tTransferAmount (Pekker.sol#383)
Variable PekkerToken._getValues(uint256).rTransferAmount (Pekker.sol#409) is too similar to
PekkerToken._transferBothExcluded(address,address,uint256).tTransferAmount (Pekker.sol#392)
Variable PekkerToken._transferStandard(address,address,uint256).rTransferAmount (Pekker.sol#366) is too
similar to PekkerToken._transferFromExcluded(address,address,uint256).tTransferAmount (Pekker.sol#383)
Variable PekkerToken._transferBothExcluded(address,address,uint256).rTransferAmount (Pekker.sol#392) is
too similar to PekkerToken._getTValues(uint256).tTransferAmount (Pekker.sol#415)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
PekkerToken.slitherConstructorVariables() (Pekker.sol#196-443) uses literals with too many digits:
	- _maxTxAmount = 100000000 * 10 ** 6 * 10 ** 9 (Pekker.sol#214)
PekkerToken.slitherConstructorConstantVariables() (Pekker.sol#196-443) uses literals with too many digits:
	- _tTotal = 100000000000 * 10 ** 6 * 10 ** 9 (Pekker.sol#206)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
PekkerToken._decimals (Pekker.sol#212) should be constant
PekkerToken._name (Pekker.sol#210) should be constant
PekkerToken._symbol (Pekker.sol#211) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
renounceOwnership() should be declared external:
	- Ownable.renounceOwnership() (Pekker.sol#179-182)
transferOwnership(address) should be declared external:
	- Ownable.transferOwnership(address) (Pekker.sol#188-192)
name() should be declared external:
	- PekkerToken.name() (Pekker.sol#221-223)
symbol() should be declared external:
	- PekkerToken.symbol() (Pekker.sol#225-227)
decimals() should be declared external:
	- PekkerToken.decimals() (Pekker.sol#229-231)
totalSupply() should be declared external:
	- PekkerToken.totalSupply() (Pekker.sol#233-235)
balanceOf(address) should be declared external:
	- PekkerToken.balanceOf(address) (Pekker.sol#237-240)
transfer(address,uint256) should be declared external:
	- PekkerToken.transfer(address,uint256) (Pekker.sol#242-245)
allowance(address,address) should be declared external:
	- PekkerToken.allowance(address,address) (Pekker.sol#247-249)
approve(address,uint256) should be declared external:
	- PekkerToken.approve(address,uint256) (Pekker.sol#251-254)
transferFrom(address,address,uint256) should be declared external:
	- PekkerToken.transferFrom(address,address,uint256) (Pekker.sol#256-261)
increaseAllowance(address,uint256) should be declared external:
	- PekkerToken.increaseAllowance(address,uint256) (Pekker.sol#263-266)
decreaseAllowance(address,uint256) should be declared external:
	- PekkerToken.decreaseAllowance(address,uint256) (Pekker.sol#268-272)
isExcluded(address) should be declared external:
	- PekkerToken.isExcluded(address) (Pekker.sol#274-276)
totalFees() should be declared external:
	- PekkerToken.totalFees() (Pekker.sol#278-280)
reflect(uint256) should be declared external:
	- PekkerToken.reflect(uint256) (Pekker.sol#287-294)
reflectionFromToken(uint256,bool) should be declared external:
	- PekkerToken.reflectionFromToken(uint256,bool) (Pekker.sol#296-305)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Pekker.sol analyzed (5 contracts with 75 detectors), 73 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```