Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	PuppyFinance
Website:	puppyfinance.org
Platform:	Binance Smart Chain
Language:	Solidity
Date:	June 25th, 2021

Table of contents

Introduction 4
Project Background 4
Audit Scope 4
Claimed Smart Contract Features 5
Audit Summary 5
Technical Quick Stats 6
Code Quality 7
Documentation 7
Use of Dependencies 7
AS-IS overview 8
Severity Definitions 10
Audit Findings 10
Conclusion 13
Our Methodology 14
Disclaimers 16
Appendix
- Code Flow Diagram 17
- Slither Report Log 18

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was commissioned by the Puppy Finance on June 25th, 2021 to perform a security audit of the Puppy Token (PUPPY) smart contract.

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

Puppy Finance is a whole new Decentralized Exchange on Binance smart chain known as 2nd Generation yield farming mechanism that allows perpetual price increase with a sustainable and profitable farming yield. Puppy Token is a governance token for the PuppyFinance ecosystem.

Audit scope

Name	**Code Review and Security Analysis Report for Puppy Token (PUPPY) Smart Contract**
Platform	**BSC / Solidity**
File	SeleniumToken.sol
Smart Contract Online Code	https://bscscan.com/address/0x3309338518a85d9653498ea6ffa1a1d8d77304d6#code
File MD5 Hash	681460846DBC10803FA6F459C5FB33D6
Audit Date	June 25th, 2021

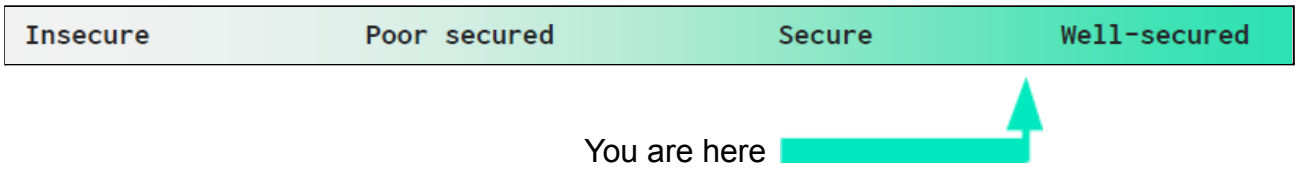
Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Name: Puppy Token	**YES, This is valid.**
Symbol: PUPPY	**YES, This is valid.**
Decimals: 18	**YES, This is valid.**
Max Supply: 2,000,000,000	**No max minting set. Ideally, the owner must be a master chef smart contract.**
Chain: Binance Smart Chain (BEP-20)	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contract is **Well 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 used various tools like MythX, Slither and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 1 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	Passed
	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**

Email: audit@EtherAuthority.io

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 Puppy 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 PuppyFinance Token.

The PuppyFinance 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 **well** commented on smart contracts.

Documentation

We were given Puppy Token smart contract code in the form of a BscScan web link. The hashes of that code are mentioned above in the table.

As mentioned above, some code parts are **well** commented. So it is easy 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://puppyfinance.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

(1) Interface

(a) IBEP20

(2) Inherited contracts

(a) Context

(b) Ownable

(c) IBEP20

(d) BEP20

(e) MasterChef

(3) Struct

(a) UserInfo: Information about UserInfo.

(b) PoolInfo: Information about PoolInfo.

(c) Checkpoint: Information about Checkpoint.

(4) Usages

(a) using SafeMath for uint256;

(b) using Address for address;

(c) using SafeBEP20 for IBEP20;

(5) Events

(a) event Transfer(address indexed from, address indexed to, uint256 value);

(b) event Approval(address indexed owner, address indexed spender, uint256 value);

(c) event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

(d) event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

(e) event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

(f) event Deposit(address indexed user, uint256 indexed pid, uint256 amount);

(g) event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);

(h) event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);

(6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	mint	write	Owner must be master chef	No Issue
2	delegates	external	Passed	No Issue
3	delegate	external	Passed	No Issue
4	delegateBySig	external	Handle signature carefully	No Issue
5	getCurrentVotes	external	Passed	No Issue
6	getPriorVotes	external	Gas consuming loop found	Keep array length limited
7	_delegate	internal	Passed	No Issue
8	_moveDelegates	internal	Passed	No Issue
9	_writeCheckpoint	internal	Passed	No Issue
10	safe32	internal	Passed	No Issue
11	getChainId	internal	Passed	No Issue
12	getOwner	external	Passed	No Issue
13	name	read	Passed	No Issue
14	symbol	read	Passed	No Issue
15	decimals	read	Passed	No Issue
16	totalSupply	read	Passed	No Issue
17	balanceOf	read	Passed	No Issue
18	transfer	write	Passed	No Issue
19	allowance	read	Passed	No Issue
20	approve	write	Passed	No Issue
21	transferFrom	write	Passed	No Issue
22	increaseAllowance	write	Passed	No Issue
23	decreaseAllowance	write	Passed	No Issue
24	_transfer	internal	Passed	No Issue
25	_mint	internal	Passed	No Issue
26	_burn	internal	Passed	No Issue
27	_approve	internal	Passed	No Issue
28	_burnFrom	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) Possible gas consuming loop:

```
1045        uint32 lower = 0;
1046        uint32 upper = nCheckpoints - 1;
1047        while (upper > lower) {
1048            uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
1049            Checkpoint memory cp = checkpoints[account][center];
1050            if (cp.fromBlock == blockNumber) {
1051                return cp.votes;
1052            } else if (cp.fromBlock < blockNumber) {
1053                lower = center;
1054            } else {
1055                upper = center - 1;
1056            }
1057        }
```

in the getPriorVotes function, if the upper value is too high than lower, then it will consume a lot of gas. It may possibly hit the block gas limit.

Resolution: nCheckpoints should be kept limited, so it does not execute a lot of code blocks.

Very Low / Discussion / Best practices:

(1) Solidity version

```
pragma solidity 0.6.12;
```

Use the latest solidity version while contract deployment to prevent any compiler version level bugs.
Resolution: This issue is acknowledged.

(2) Max minting is not set by default. Owner can mint as many tokens as he wants. So, ideally, the ownership should be transferred to the master chef smart contract.

(3) Consider specifying function visibility to “external” instead of “public”, if that function is not being called internally. It will save some gas as well.
https://ethereum.stackexchange.com/questions/32353/what-is-the-difference-between-an-internal-external-and-public-private-function/32464

(4) Redundant code: some of the code parts are not used anywhere. such as: masterChef contract code, etc. We suggest removing those unused code blocks to make this token smart contract clean.

(5) Approve of BEP20 standard: This can be used to front-run. From the client side, only use this function to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved). This should be done from the client side.

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:

- renounceOwnership: Owner can give up ownership completely.
- transferOwnership: Owner can transfer the ownership to any other wallet.
- mint: Owner can mint tokens without any limits.

As mentioned above, ownership must be delegated to masterChef smart contract, so it does not have any issues regarding the tokenomics.

Conclusion

We were given contract codes. And we have used all possible tests based on giving 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 "**Well Secured**".

Email: audit@EtherAuthority.io

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).

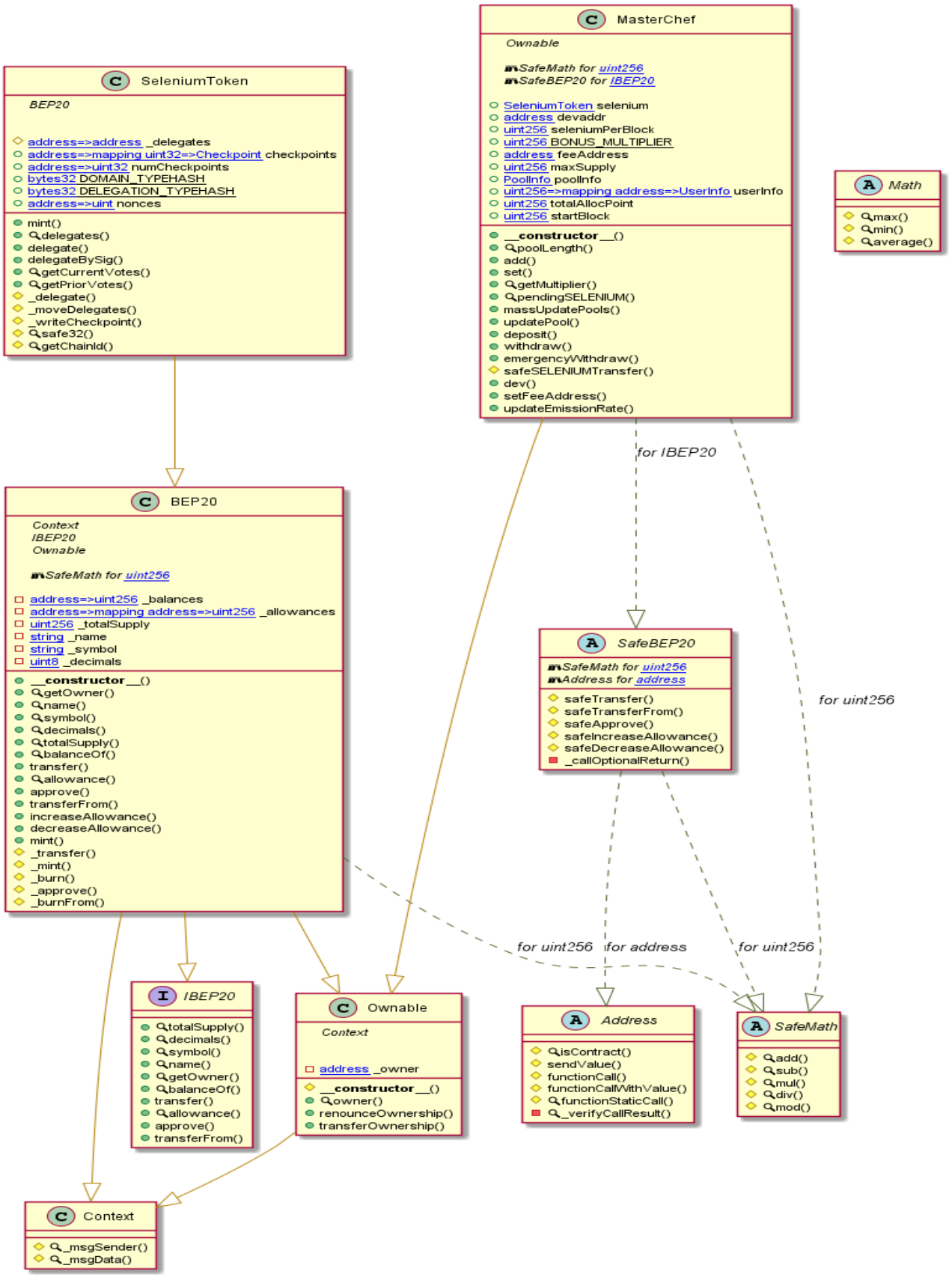
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 - Selenium Token



Slither Results Log

Slither log >> SeleniumToken.sol

INFO:Detectors:
MasterChef.safeSELENIUMTransfer(address,uint256) (SeleniumToken.sol#1380-1387) ignores return value by selenium.transfer(_to,seleniumBal) (SeleniumToken.sol#1383)
MasterChef.safeSELENIUMTransfer(address,uint256) (SeleniumToken.sol#1380-1387) ignores return value by selenium.transfer(_to,_amount) (SeleniumToken.sol#1385)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
MasterChef.pendingSELENIUM(uint256,address) (SeleniumToken.sol#1275-1288) performs a multiplication on the result of a division:
	-seleniumReward = multiplier.mul(seleniumPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (SeleniumToken.sol#1282)
	-accSELENIUMPerShare = accSELENIUMPerShare.add(seleniumReward.mul(1e12).div(lpSupply)) (SeleniumToken.sol#1285)
MasterChef.updatePool(uint256) (SeleniumToken.sol#1299-1323) performs a multiplication on the result of a division:
	-seleniumReward = multiplier.mul(seleniumPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (SeleniumToken.sol#1311)
	-pool.accSELENIUMPerShare = pool.accSELENIUMPerShare.add(seleniumReward.mul(1e12).div(lpSupply)) (SeleniumToken.sol#1321)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
SeleniumToken._writeCheckpoint(address,uint32,uint256,uint256) (SeleniumToken.sol#1093-1111) uses a dangerous strict equality:
	- nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (SeleniumToken.sol#1103)
MasterChef.updatePool(uint256) (SeleniumToken.sol#1299-1323) uses a dangerous strict equality:
	- lpSupply == 0 || pool.allocPoint == 0 || seleniumPerBlock == 0 (SeleniumToken.sol#1305)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in MasterChef.add(uint256,IBEP20,uint16,bool) (SeleniumToken.sol#1239-1253):
	External calls:
	- massUpdatePools() (SeleniumToken.sol#1242)
		- selenium.mint(devaddr,devReward) (SeleniumToken.sol#1319)
		- selenium.mint(address(this),seleniumReward) (SeleniumToken.sol#1320)
	State variables written after the call(s):
	- poolInfo.push(PoolInfo(_lpToken,_allocPoint,lastRewardBlock,0,_depositFeeBP)) (SeleniumToken.sol#1246-1252)
	- totalAllocPoint = totalAllocPoint.add(_allocPoint) (SeleniumToken.sol#1245)
Reentrancy in MasterChef.deposit(uint256,uint256) (SeleniumToken.sol#1326-1348):
	External calls:
	- updatePool(_pid) (SeleniumToken.sol#1329)
		- selenium.mint(devaddr,devReward) (SeleniumToken.sol#1319)
		- selenium.mint(address(this),seleniumReward) (SeleniumToken.sol#1320)
	- safeSELENIUMTransfer(msg.sender,pending) (SeleniumToken.sol#1333)
		- selenium.transfer(_to,seleniumBal) (SeleniumToken.sol#1383)
		- selenium.transfer(_to,_amount) (SeleniumToken.sol#1385)
	- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (SeleniumToken.sol#1337)
	- pool.lpToken.safeTransfer(feeAddress,depositFee) (SeleniumToken.sol#1340)
	State variables written after the call(s):
	- user.amount = user.amount.add(_amount).sub(depositFee) (SeleniumToken.sol#1341)
Reentrancy in MasterChef.deposit(uint256,uint256) (SeleniumToken.sol#1326-1348):
	External calls:
	- updatePool(_pid) (SeleniumToken.sol#1329)

- selenium.mint(devaddr,devReward) (SeleniumToken.sol#1319)
- selenium.mint(address(this),seleniumReward) (SeleniumToken.sol#1320)
- safeSELENIUMTransfer(msg.sender,pending) (SeleniumToken.sol#1333)
- selenium.transfer(_to,seleniumBal) (SeleniumToken.sol#1383)
- selenium.transfer(_to,_amount) (SeleniumToken.sol#1385)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (SeleniumToken.sol#1337)

State variables written after the call(s):
- user.amount = user.amount.add(_amount) (SeleniumToken.sol#1343)

Reentrancy in MasterChef.set(uint256,uint256,uint16,bool) (SeleniumToken.sol#1256-1264):

External calls:
- massUpdatePools() (SeleniumToken.sol#1259)
- selenium.mint(devaddr,devReward) (SeleniumToken.sol#1319)
- selenium.mint(address(this),seleniumReward) (SeleniumToken.sol#1320)

State variables written after the call(s):
- poolInfo[_pid].allocPoint = _allocPoint (SeleniumToken.sol#1262)
- poolInfo[_pid].depositFeeBP = _depositFeeBP (SeleniumToken.sol#1263)
- totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint) (SeleniumToken.sol#1261)

Reentrancy in MasterChef.updateEmissionRate(uint256) (SeleniumToken.sol#1401-1404):

External calls:
- massUpdatePools() (SeleniumToken.sol#1402)
- selenium.mint(devaddr,devReward) (SeleniumToken.sol#1319)
- selenium.mint(address(this),seleniumReward) (SeleniumToken.sol#1320)

State variables written after the call(s):
- seleniumPerBlock = _seleniumPerBlock (SeleniumToken.sol#1403)

Reentrancy in MasterChef.updatePool(uint256) (SeleniumToken.sol#1299-1323):

External calls:
- selenium.mint(devaddr,devReward) (SeleniumToken.sol#1319)
- selenium.mint(address(this),seleniumReward) (SeleniumToken.sol#1320)

State variables written after the call(s):
- pool.accSELENIUMPerShare = pool.accSELENIUMPerShare.add(seleniumReward.mul(1e12).div(lpSupply)) (SeleniumToken.sol#1321)
- pool.lastRewardBlock = block.number (SeleniumToken.sol#1322)

Reentrancy in MasterChef.withdraw(uint256,uint256) (SeleniumToken.sol#1351-1366):

External calls:
- updatePool(_pid) (SeleniumToken.sol#1355)
- selenium.mint(devaddr,devReward) (SeleniumToken.sol#1319)
- selenium.mint(address(this),seleniumReward) (SeleniumToken.sol#1320)
- safeSELENIUMTransfer(msg.sender,pending) (SeleniumToken.sol#1358)
- selenium.transfer(_to,seleniumBal) (SeleniumToken.sol#1383)
- selenium.transfer(_to,_amount) (SeleniumToken.sol#1385)

State variables written after the call(s):
- user.amount = user.amount.sub(_amount) (SeleniumToken.sol#1361)

Reentrancy in MasterChef.withdraw(uint256,uint256) (SeleniumToken.sol#1351-1366):

External calls:
- updatePool(_pid) (SeleniumToken.sol#1355)
- selenium.mint(devaddr,devReward) (SeleniumToken.sol#1319)
- selenium.mint(address(this),seleniumReward) (SeleniumToken.sol#1320)
- safeSELENIUMTransfer(msg.sender,pending) (SeleniumToken.sol#1358)
- selenium.transfer(_to,seleniumBal) (SeleniumToken.sol#1383)
- selenium.transfer(_to,_amount) (SeleniumToken.sol#1385)
- pool.lpToken.safeTransfer(address(msg.sender),_amount) (SeleniumToken.sol#1362)

State variables written after the call(s):
- user.rewardDebt = user.amount.mul(pool.accSELENIUMPerShare).div(1e12) (SeleniumToken.sol#1364)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

INFO:Detectors:

BEP20.constructor(string,string).name (SeleniumToken.sol#640) shadows:
- BEP20.name() (SeleniumToken.sol#656-658) (function)
- IBEP20.name() (SeleniumToken.sol#186) (function)

BEP20.constructor(string,string).symbol (SeleniumToken.sol#640) shadows:
- BEP20.symbol() (SeleniumToken.sol#664-666) (function)
- IBEP20.symbol() (SeleniumToken.sol#181) (function)

BEP20.allowance(address,address).owner (SeleniumToken.sol#705) shadows:

```
        - Ownable.owner() (SeleniumToken.sol#555-557) (function)
BEP20._approve(address,address,uint256).owner (SeleniumToken.sol#864) shadows:
        - Ownable.owner() (SeleniumToken.sol#555-557) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
MasterChef.constructor(SeleniumToken,address,address,uint256,uint256,uint256)._devaddr
(SeleniumToken.sol#1219) lacks a zero-check on :
            - devaddr = _devaddr (SeleniumToken.sol#1226)
MasterChef.constructor(SeleniumToken,address,address,uint256,uint256,uint256)._feeAddress
(SeleniumToken.sol#1220) lacks a zero-check on :
            - feeAddress = _feeAddress (SeleniumToken.sol#1227)
MasterChef.dev(address)._devaddr (SeleniumToken.sol#1390) lacks a zero-check on :
            - devaddr = _devaddr (SeleniumToken.sol#1392)
MasterChef.setFeeAddress(address)._feeAddress (SeleniumToken.sol#1395) lacks a zero-check on :
            - feeAddress = _feeAddress (SeleniumToken.sol#1397)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in MasterChef.deposit(uint256,uint256) (SeleniumToken.sol#1326-1348):
        External calls:
        - updatePool(_pid) (SeleniumToken.sol#1329)
            - selenium.mint(devaddr,devReward) (SeleniumToken.sol#1319)
            - selenium.mint(address(this),seleniumReward) (SeleniumToken.sol#1320)
        - safeSELENIUMTransfer(msg.sender,pending) (SeleniumToken.sol#1333)
            - selenium.transfer(_to,seleniumBal) (SeleniumToken.sol#1383)
            - selenium.transfer(_to,_amount) (SeleniumToken.sol#1385)
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount)
(SeleniumToken.sol#1337)
        - pool.lpToken.safeTransfer(feeAddress,depositFee) (SeleniumToken.sol#1340)
        Event emitted after the call(s):
        - Deposit(msg.sender,_pid,_amount) (SeleniumToken.sol#1347)
Reentrancy in MasterChef.emergencyWithdraw(uint256) (SeleniumToken.sol#1369-1377):
        External calls:
        - pool.lpToken.safeTransfer(address(msg.sender),amount) (SeleniumToken.sol#1375)
        Event emitted after the call(s):
        - EmergencyWithdraw(msg.sender,_pid,amount) (SeleniumToken.sol#1376)
Reentrancy in MasterChef.withdraw(uint256,uint256) (SeleniumToken.sol#1351-1366):
        External calls:
        - updatePool(_pid) (SeleniumToken.sol#1355)
            - selenium.mint(devaddr,devReward) (SeleniumToken.sol#1319)
            - selenium.mint(address(this),seleniumReward) (SeleniumToken.sol#1320)
        - safeSELENIUMTransfer(msg.sender,pending) (SeleniumToken.sol#1358)
            - selenium.transfer(_to,seleniumBal) (SeleniumToken.sol#1383)
            - selenium.transfer(_to,_amount) (SeleniumToken.sol#1385)
        - pool.lpToken.safeTransfer(address(msg.sender),_amount) (SeleniumToken.sol#1362)
        Event emitted after the call(s):
        - Withdraw(msg.sender,_pid,_amount) (SeleniumToken.sol#1365)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
SeleniumToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32)
(SeleniumToken.sol#959-1000) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now <= expiry,TOKEN::delegateBySig: signature expired)
(SeleniumToken.sol#998)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (SeleniumToken.sol#283-292) uses assembly
        - INLINE ASM (SeleniumToken.sol#290)
Address._verifyCallResult(bool,bytes,string) (SeleniumToken.sol#404-421) uses assembly
        - INLINE ASM (SeleniumToken.sol#413-416)
SeleniumToken.getChainId() (SeleniumToken.sol#1118-1122) uses assembly
        - INLINE ASM (SeleniumToken.sol#1120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['0.6.12', '>=0.4.0', '>=0.6.0<0.8.0', '>=0.6.2<0.8.0', '>=0.6.4']
```

- >=0.6.0<0.8.0 (SeleniumToken.sol#7)
- >=0.6.4 (SeleniumToken.sol#165)
- >=0.6.2<0.8.0 (SeleniumToken.sol#260)
- >=0.6.0<0.8.0 (SeleniumToken.sol#426)
- >=0.6.0<0.8.0 (SeleniumToken.sol#499)
- >=0.6.0<0.8.0 (SeleniumToken.sol#524)
- >=0.4.0 (SeleniumToken.sol#592)
- 0.6.12 (SeleniumToken.sol#884)
- 0.6.12 (SeleniumToken.sol#1151)

Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Address.functionCall(address,bytes) (SeleniumToken.sol#336-338) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (SeleniumToken.sol#361-363) is never used and should be removed
Address.functionStaticCall(address,bytes) (SeleniumToken.sol#386-388) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (SeleniumToken.sol#396-402) is never used and should be removed
Address.sendValue(address,uint256) (SeleniumToken.sol#310-316) is never used and should be removed
BEP20._burn(address,uint256) (SeleniumToken.sol#843-849) is never used and should be removed
BEP20._burnFrom(address,uint256) (SeleniumToken.sol#878-881) is never used and should be removed
Context._msgData() (SeleniumToken.sol#516-519) is never used and should be removed
Math.average(uint256,uint256) (SeleniumToken.sol#1144-1147) is never used and should be removed
Math.max(uint256,uint256) (SeleniumToken.sol#1129-1131) is never used and should be removed
SafeBEP20.safeApprove(IBEP20,address,uint256) (SeleniumToken.sol#457-466) is never used and should be removed
SafeBEP20.safeDecreaseAllowance(IBEP20,address,uint256) (SeleniumToken.sol#473-476) is never used and should be removed
SafeBEP20.safeIncreaseAllowance(IBEP20,address,uint256) (SeleniumToken.sol#468-471) is never used and should be removed
SafeMath.mod(uint256,uint256) (SeleniumToken.sol#143-145) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (SeleniumToken.sol#159-162) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version>=0.6.0<0.8.0 (SeleniumToken.sol#7) is too complex
Pragma version>=0.6.4 (SeleniumToken.sol#165) allows old versions
Pragma version>=0.6.2<0.8.0 (SeleniumToken.sol#260) is too complex
Pragma version>=0.6.0<0.8.0 (SeleniumToken.sol#426) is too complex
Pragma version>=0.6.0<0.8.0 (SeleniumToken.sol#499) is too complex
Pragma version>=0.6.0<0.8.0 (SeleniumToken.sol#524) is too complex
Pragma version>=0.4.0 (SeleniumToken.sol#592) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (SeleniumToken.sol#310-316):
- (success) = recipient.call{value: amount}() (SeleniumToken.sol#314)

Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (SeleniumToken.sol#371-378):
- (success,returndata) = target.call{value: value}(data) (SeleniumToken.sol#376)

Low level call in Address.functionStaticCall(address,bytes,string) (SeleniumToken.sol#396-402):
- (success,returndata) = target.staticcall(data) (SeleniumToken.sol#400)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter SeleniumToken.mint(address,uint256)._to (SeleniumToken.sol#889) is not in mixedCase
Parameter SeleniumToken.mint(address,uint256)._amount (SeleniumToken.sol#889) is not in mixedCase
Variable SeleniumToken._delegates (SeleniumToken.sol#901) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._allocPoint (SeleniumToken.sol#1239) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._lpToken (SeleniumToken.sol#1239) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._depositFeeBP (SeleniumToken.sol#1239) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._withUpdate (SeleniumToken.sol#1239) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._pid (SeleniumToken.sol#1256) is not in mixedCase

Parameter MasterChef.set(uint256,uint256,uint16,bool)._allocPoint (SeleniumToken.sol#1256) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._depositFeeBP (SeleniumToken.sol#1256) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._withUpdate (SeleniumToken.sol#1256) is not in mixedCase
Parameter MasterChef.getMultiplier(uint256,uint256)._from (SeleniumToken.sol#1267) is not in mixedCase
Parameter MasterChef.getMultiplier(uint256,uint256)._to (SeleniumToken.sol#1267) is not in mixedCase
Parameter MasterChef.pendingSELENIUM(uint256,address)._pid (SeleniumToken.sol#1275) is not in mixedCase
Parameter MasterChef.pendingSELENIUM(uint256,address)._user (SeleniumToken.sol#1275) is not in mixedCase
Parameter MasterChef.updatePool(uint256)._pid (SeleniumToken.sol#1299) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256)._pid (SeleniumToken.sol#1326) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256)._amount (SeleniumToken.sol#1326) is not in mixedCase
Parameter MasterChef.withdraw(uint256,uint256)._pid (SeleniumToken.sol#1351) is not in mixedCase
Parameter MasterChef.withdraw(uint256,uint256)._amount (SeleniumToken.sol#1351) is not in mixedCase
Parameter MasterChef.emergencyWithdraw(uint256)._pid (SeleniumToken.sol#1369) is not in mixedCase
Parameter MasterChef.safeSELENIUMTransfer(address,uint256)._to (SeleniumToken.sol#1380) is not in mixedCase
Parameter MasterChef.safeSELENIUMTransfer(address,uint256)._amount (SeleniumToken.sol#1380) is not in mixedCase
Parameter MasterChef.dev(address)._devaddr (SeleniumToken.sol#1390) is not in mixedCase
Parameter MasterChef.setFeeAddress(address)._feeAddress (SeleniumToken.sol#1395) is not in mixedCase
Parameter MasterChef.updateEmissionRate(uint256)._seleniumPerBlock (SeleniumToken.sol#1401) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (SeleniumToken.sol#517)" inContext (SeleniumToken.sol#511-520)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (SeleniumToken.sol#574-577)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (SeleniumToken.sol#583-587)
symbol() should be declared external:
- BEP20.symbol() (SeleniumToken.sol#664-666)
decimals() should be declared external:
- BEP20.decimals() (SeleniumToken.sol#671-673)
totalSupply() should be declared external:
- BEP20.totalSupply() (SeleniumToken.sol#678-680)
transfer(address,uint256) should be declared external:
- BEP20.transfer(address,uint256) (SeleniumToken.sol#697-700)
allowance(address,address) should be declared external:
- BEP20.allowance(address,address) (SeleniumToken.sol#705-707)
approve(address,uint256) should be declared external:
- BEP20.approve(address,uint256) (SeleniumToken.sol#716-719)
transferFrom(address,address,uint256) should be declared external:
- BEP20.transferFrom(address,address,uint256) (SeleniumToken.sol#733-741)
increaseAllowance(address,uint256) should be declared external:
- BEP20.increaseAllowance(address,uint256) (SeleniumToken.sol#755-758)
decreaseAllowance(address,uint256) should be declared external:
- BEP20.decreaseAllowance(address,uint256) (SeleniumToken.sol#774-777)
mint(uint256) should be declared external:
- BEP20.mint(uint256) (SeleniumToken.sol#787-790)
mint(address,uint256) should be declared external:
- SeleniumToken.mint(address,uint256) (SeleniumToken.sol#889-892)
add(uint256,IBEP20,uint16,bool) should be declared external:
- MasterChef.add(uint256,IBEP20,uint16,bool) (SeleniumToken.sol#1239-1253)
set(uint256,uint256,uint16,bool) should be declared external:
- MasterChef.set(uint256,uint256,uint16,bool) (SeleniumToken.sol#1256-1264)
deposit(uint256,uint256) should be declared external:
- MasterChef.deposit(uint256,uint256) (SeleniumToken.sol#1326-1348)

withdraw(uint256,uint256) should be declared external:
	- MasterChef.withdraw(uint256,uint256) (SeleniumToken.sol#1351-1366)
emergencyWithdraw(uint256) should be declared external:
	- MasterChef.emergencyWithdraw(uint256) (SeleniumToken.sol#1369-1377)
dev(address) should be declared external:
	- MasterChef.dev(address) (SeleniumToken.sol#1390-1393)
setFeeAddress(address) should be declared external:
	- MasterChef.setFeeAddress(address) (SeleniumToken.sol#1395-1398)
updateEmissionRate(uint256) should be declared external:
	- MasterChef.updateEmissionRate(uint256) (SeleniumToken.sol#1401-1404)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:SeleniumToken.sol analyzed (10 contracts with 75 detectors), 103 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Email: audit@EtherAuthority.io