HashEx

Blockchain Security

PlateauFinance

smart contracts
audit report

hashex.org

contact@hashex.org

Contents

1 Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2 Overview

HashEx was commissioned by the PlateauFinance team to perform an audit of their smart contracts. The audit was conducted between October 18 and October 21, 2021.The code is available at the address 0xeCfE536a209e405Db19887830b366E397f5B917a .

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts.
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

2.1 Summary

Project name	PlateauFinance
URL	https://avascan.info/blockchain/c/address/0xeCfE536a209e405Db19887830b366E397f5B917a
Platform	Avalanche Network
Language	Solidity

2.2 Contracts

Name	Address
PlateauFinance	0xeCfE536a209e405Db19887830b366E397f5B917a

3 Found issues

9
Total issues

🟧	High	1 (11%)
🟪	Medium	5 (56%)
🟦	Low	3 (33%)

PlateauFinance

ID	Title	Severity	Status
01	Input data is not checked	🟧 High	Acknowledged
02	ERC20 standard violation	🟪 Medium	Acknowledged
03	Locked AVAX	🟪 Medium	Acknowledged
04	Hardcoded addresses	🟪 Medium	Acknowledged
05	Missing Approval event	🟪 Medium	Acknowledged
06	No cap for amount of tokens used for adding liquidity	🟪 Medium	Acknowledged
07	Confusing comment	🟦 Low	Acknowledged
08	Unused variables	🟦 Low	Acknowledged

09	Functions may be declared external	■ Low	Acknowledged

4 Contracts

4.1 PlateauFinance

4.1.1 Overview

The contract is an ERC20 token with a commission on transfers. Some addresses may be excluded from the commissions by the contract owner. Commissions are taken for burning and for adding liquidity.

4.1.2 Issues

01. Input data is not checked

■ High ⓘ Acknowledged

The function setFee receives and sets new values BURN_FEE and LP_FEE. The new values aren't checked before setting. Setting a wrong value may completely break token transfers for not excluded addresses.

Recommendation

Since the values represent shares in percents we recommend checking whether that fits into a suitable range of values.

02. ERC20 standard violation

▪ Medium ⓘ Acknowledged

Implementation of the transfer() function does not allow to input zero amount as it’s demanded in ERC20 and BEP20 standards. This issue may break the interaction with smart contracts that rely on full ERC20 support. Also, transfer functions of the reviewed contract don’t throw error messages for the amounts bigger than the sender's balance (like “ERC20: transfer amount exceeds allowance” in OpenZeppelin’s ERC20 implementation) which may confuse users.

03. Locked AVAX

▪ Medium ⓘ Acknowledged

The payable receive() function in makes it possible for the contract to receive ether/bnb. Moreover, addLiquidityAVAX from JoeRouter02 returns any ETH/AVAX leftovers back to the sender. There’s no implemented mechanism for handling this contract’s ETH/AVAX balance.

04. Hardcoded addresses

▪ Medium ⓘ Acknowledged

The addresses of JoeRouter02 and pair are immutable. This may cause a partial malfunction in case of future upgrades of TraiderJoe’s services.

05. Missing Approval event

▪ Medium ⓘ Acknowledged

The token does not emit Approval(address,address,uint256) event which must be triggered on any successful call to approve(address _spender, uint256 _value).

Recommendation

Emit Approval event in the approve() function.

06. No cap for amount of tokens used for adding liquidity

▪ Medium ⓘ Acknowledged

If a big amount of tokens is accumulated on the token contract, adding liquidity will lead to a significant dump in the token price because half of the tokens are sold for AVAX. This may be the case if the maxLPCap parameter is set to a big number or swapAndLiquify functionality is turned off for a long period of time.

07. Confusing comment

▪ Low ⓘ Acknowledged

At the L855 there is a comment //exclude owner and this contract from fee, but only the address of the contract is actually excluded.

08. Unused variables

- Low ⓘ Acknowledged

Values _previousBurnFee and _previousLPFee are only defined and set, but never used.

09. Functions may be declared external

- Low ⓘ Acknowledged

View functions that implement the ERC20 interface should be defined as external instead of public, to save gas.

5. Conclusion

The contract is highly dependent on the owner account. If the owner account is compromised, the token may be completely broken. We recommend securing the owner account by putting it behind a Timelock contract or using a multisig.

Audit includes recommendations on code improvement.

PlateauFinance team response: There is only one major issue that is fee should not be set with certain percentage. We determined that this will not affect the contract and believe there is no need for redeployment.

Appendix A. Issues' severity classification

Critical. Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.

High. Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.

Medium. Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.

Low. Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.

Informational. Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

Appendix C. Static Analyzer's output

```
Reentrancy in PlateauFinance._transfer(address,address,uint256)
(downclorox.sol#1098-1154):
 External calls:
 - swapAndLiquify(contractTokenBalance) (downclorox.sol#1120)
  - router.addLiquidityAVAX{value: ethAmount}
(address(this),tokenAmount,0,0,lpReceiverAddress,block.timestamp)
(downclorox.sol#1206-1213)
  - router.swapExactTokensForAVAXSupportingFeeOnTransferTokens(tokenAmount,0,path,a
ddress(this),block.timestamp) (downclorox.sol#1190-1196)
 External calls sending eth:
 - swapAndLiquify(contractTokenBalance) (downclorox.sol#1120)
  - router.addLiquidityAVAX{value: ethAmount}
(address(this),tokenAmount,0,0,lpReceiverAddress,block.timestamp)
(downclorox.sol#1206-1213)
 State variables written after the call(s):
 - _balances[from] = senderBalance - amount (downclorox.sol#1127)
 - _balances[to] += amount (downclorox.sol#1131)
 - _balances[to] += amount.sub(burnFee).sub(lpFee) (downclorox.sol#1138)
 - _balances[address(this)] += lpFee (downclorox.sol#1141)
 - _balances[address(0x000000000000000000000000000000000000dEaD)] += burnFee
(downclorox.sol#1144)
 - swapAndLiquifyEnable = true (downclorox.sol#1121)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#reentrancy-vulnerabilities

PlateauFinance.addLiquidity(uint256,uint256) (downclorox.sol#1201-1214) ignores
return value by router.addLiquidityAVAX{value: ethAmount}
(address(this),tokenAmount,0,0,lpReceiverAddress,block.timestamp)
(downclorox.sol#1206-1213)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-
return

PlateauFinance.allowance(address,address).owner (downclorox.sol#966) shadows:
 - Ownable.owner() (downclorox.sol#742-744) (function)
PlateauFinance._approve(address,address,uint256).owner (downclorox.sol#1073)
shadows:
 - Ownable.owner() (downclorox.sol#742-744) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-
variable-shadowing

PlateauFinance.setFee(uint256,uint256) (downclorox.sol#863-869) should emit an
event for:
 - BURN_FEE = _burnFee (downclorox.sol#864)
 - LP_FEE = _lpFee (downclorox.sol#866)
PlateauFinance.setMaxLPCap(uint256) (downclorox.sol#879-881) should emit an event
for:
 - maxLPCap = cap (downclorox.sol#880)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-
events-arithmetic

PlateauFinance.constructor(address)._lpReceiverAddress (downclorox.sol#840) lacks
a zero-check on :
  - lpReceiverAddress = _lpReceiverAddress (downclorox.sol#844)
PlateauFinance.changeLpReceiverAddress(address).wallet (downclorox.sol#871) lacks
a zero-check on :
  - lpReceiverAddress = wallet (downclorox.sol#872)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-
zero-address-validation

Reentrancy in PlateauFinance.constructor(address) (downclorox.sol#840-860):
 External calls:
 - _pair =
IJoeFactory(_router.factory()).createPair(address(this),_router.WAVAX())
(downclorox.sol#850-851)
 State variables written after the call(s):
 - _excluded[address(this)] = true (downclorox.sol#857)
 - router = _router (downclorox.sol#853)
Reentrancy in PlateauFinance.swapAndLiquify(uint256) (downclorox.sol#1160-1180):
 External calls:
 - swapTokensForEth(half) (downclorox.sol#1172)
  - router.swapExactTokensForAVAXSupportingFeeOnTransferTokens(tokenAmount,0,path,a
ddress(this),block.timestamp) (downclorox.sol#1190-1196)
 - addLiquidity(otherHalf,newBalance) (downclorox.sol#1177)
```

```
  - router.addLiquidityAVAX{value: ethAmount}
(address(this),tokenAmount,0,0,lpReceiverAddress,block.timestamp)
(downclorox.sol#1206-1213)
 External calls sending eth:
 - addLiquidity(otherHalf,newBalance) (downclorox.sol#1177)
  - router.addLiquidityAVAX{value: ethAmount}
(address(this),tokenAmount,0,0,lpReceiverAddress,block.timestamp)
(downclorox.sol#1206-1213)
 State variables written after the call(s):
 - addLiquidity(otherHalf,newBalance) (downclorox.sol#1177)
  - _allowances[owner][spender] = amount (downclorox.sol#1077)
Reentrancy in PlateauFinance.transferFrom(address,address,uint256)
(downclorox.sol#1004-1012):
 External calls:
 - _transfer(sender,recipient,amount) (downclorox.sol#1005)
  - router.addLiquidityAVAX{value: ethAmount}
(address(this),tokenAmount,0,0,lpReceiverAddress,block.timestamp)
(downclorox.sol#1206-1213)
  - router.swapExactTokensForAVAXSupportingFeeOnTransferTokens(tokenAmount,0,path,a
ddress(this),block.timestamp) (downclorox.sol#1190-1196)
 External calls sending eth:
 - _transfer(sender,recipient,amount) (downclorox.sol#1005)
  - router.addLiquidityAVAX{value: ethAmount}
(address(this),tokenAmount,0,0,lpReceiverAddress,block.timestamp)
(downclorox.sol#1206-1213)
 State variables written after the call(s):
 - _approve(sender,_msgSender(),currentAllowance - amount) (downclorox.sol#1009)
  - _allowances[owner][spender] = amount (downclorox.sol#1077)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#reentrancy-vulnerabilities-2

Reentrancy in PlateauFinance._transfer(address,address,uint256)
(downclorox.sol#1098-1154):
 External calls:
 - swapAndLiquify(contractTokenBalance) (downclorox.sol#1120)
  - router.addLiquidityAVAX{value: ethAmount}
(address(this),tokenAmount,0,0,lpReceiverAddress,block.timestamp)
(downclorox.sol#1206-1213)
```

```
  - router.swapExactTokensForAVAXSupportingFeeOnTransferTokens(tokenAmount,0,path,a
ddress(this),block.timestamp) (downclorox.sol#1190-1196)
 External calls sending eth:
 - swapAndLiquify(contractTokenBalance) (downclorox.sol#1120)
  - router.addLiquidityAVAX{value: ethAmount}
(address(this),tokenAmount,0,0,lpReceiverAddress,block.timestamp)
(downclorox.sol#1206-1213)
 Event emitted after the call(s):
 - Transfer(from,to,amount) (downclorox.sol#1132)
 - Transfer(from,to,amount) (downclorox.sol#1139)
 - Transfer(msg.sender,address(this),lpFee) (downclorox.sol#1142)
 -
Transfer(msg.sender,address(0x000000000000000000000000000000000000dEaD),burnFee)
(downclorox.sol#1146)
Reentrancy in PlateauFinance.swapAndLiquify(uint256) (downclorox.sol#1160-1180):
 External calls:
 - swapTokensForEth(half) (downclorox.sol#1172)
  - router.swapExactTokensForAVAXSupportingFeeOnTransferTokens(tokenAmount,0,path,a
ddress(this),block.timestamp) (downclorox.sol#1190-1196)
 - addLiquidity(otherHalf,newBalance) (downclorox.sol#1177)
  - router.addLiquidityAVAX{value: ethAmount}
(address(this),tokenAmount,0,0,lpReceiverAddress,block.timestamp)
(downclorox.sol#1206-1213)
 External calls sending eth:
 - addLiquidity(otherHalf,newBalance) (downclorox.sol#1177)
  - router.addLiquidityAVAX{value: ethAmount}
(address(this),tokenAmount,0,0,lpReceiverAddress,block.timestamp)
(downclorox.sol#1206-1213)
 Event emitted after the call(s):
 - SwapAndLiquify(half,newBalance,otherHalf) (downclorox.sol#1179)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#reentrancy-vulnerabilities-3

Address.isContract(address) (downclorox.sol#519-528) uses assembly
 - INLINE ASM (downclorox.sol#526)
Address._functionCallWithValue(address,bytes,uint256,string)
(downclorox.sol#612-633) uses assembly
```

```
 - INLINE ASM (downclorox.sol#625-628)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-
usage

PlateauFinance._transfer(address,address,uint256) (downclorox.sol#1098-1154)
compares to a boolean constant:
 -swapAndLiquifyEnable == true && overMinTokenBalance && from != _pair
(downclorox.sol#1114-1116)
PlateauFinance._transfer(address,address,uint256) (downclorox.sol#1098-1154)
compares to a boolean constant:
 -_excluded[from] == true (downclorox.sol#1130)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-
equality

Different versions of Solidity is used:
 - Version used: ['>=0.6.2', '^0.8.0']
 - >=0.6.2 (downclorox.sol#5)
 - >=0.6.2 (downclorox.sol#164)
 - ^0.8.0 (downclorox.sol#223)
 - ^0.8.0 (downclorox.sol#447)
 - ^0.8.0 (downclorox.sol#473)
 - ^0.8.0 (downclorox.sol#637)
 - ^0.8.0 (downclorox.sol#711)
 - ^0.8.0 (downclorox.sol#778)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-
pragma-directives-are-used

Address._functionCallWithValue(address,bytes,uint256,string)
(downclorox.sol#612-633) is never used and should be removed
Address.functionCall(address,bytes) (downclorox.sol#572-574) is never used and
should be removed
Address.functionCall(address,bytes,string) (downclorox.sol#582-584) is never used
and should be removed
Address.functionCallWithValue(address,bytes,uint256) (downclorox.sol#597-599) is
never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string)
(downclorox.sol#607-610) is never used and should be removed
```

```
Address.isContract(address) (downclorox.sol#519-528) is never used and should be
removed
Address.sendValue(address,uint256) (downclorox.sol#546-552) is never used and
should be removed
Context._msgData() (downclorox.sol#490-493) is never used and should be removed
SafeERC20._callOptionalReturn(IERC20,bytes) (downclorox.sol#695-705) is never used
and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (downclorox.sol#668-677) is never
used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (downclorox.sol#684-687)
is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (downclorox.sol#679-682)
is never used and should be removed
SafeERC20.safeTransfer(IERC20,address,uint256) (downclorox.sol#653-655) is never
used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256)
(downclorox.sol#657-659) is never used and should be removed
SafeMath.add(uint256,uint256) (downclorox.sol#237-242) is never used and should be
removed
SafeMath.mod(uint256,uint256) (downclorox.sol#347-349) is never used and should be
removed
SafeMath.mod(uint256,uint256,string) (downclorox.sol#363-366) is never used and
should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

PlateauFinance._previousBurnFee (downclorox.sol#822) is set pre-construction with
a non-constant function or state variable:
 - BURN_FEE
PlateauFinance._previousLPFee (downclorox.sol#823) is set pre-construction with a
non-constant function or state variable:
 - LP_FEE
PlateauFinance.maxLPCap (downclorox.sol#824) is set pre-construction with a non-
constant function or state variable:
 - 1000 * 10 ** decimals()
PlateauFinance.MAX_SUPPLY (downclorox.sol#831) is set pre-construction with a non-
constant function or state variable:
 - 1000000000 * 10 ** decimals()
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-
initializing-state

Pragma version>=0.6.2 (downclorox.sol#5) allows old versions
Pragma version>=0.6.2 (downclorox.sol#164) allows old versions
Pragma version^0.8.0 (downclorox.sol#223) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (downclorox.sol#447) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (downclorox.sol#473) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (downclorox.sol#637) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (downclorox.sol#711) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (downclorox.sol#778) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.7 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) (downclorox.sol#546-552):
 - (success) = recipient.call{value: amount}() (downclorox.sol#550)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string)
(downclorox.sol#612-633):
 - (success,returndata) = target.call{value: weiValue}(data) (downclorox.sol#616)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-
calls

PlateauFinance (downclorox.sol#807-1216) should inherit from IERC20Metadata
(downclorox.sol#455-470)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-
inheritance

Function IJoeRouter01.WAVAX() (downclorox.sol#10) is not in mixedCase
Parameter PlateauFinance.setFee(uint256,uint256)._burnFee (downclorox.sol#863) is
not in mixedCase
```

```
Parameter PlateauFinance.setFee(uint256,uint256)._lpFee (downclorox.sol#863) is
not in mixedCase
Parameter PlateauFinance.calculateBurnFee(uint256)._amount (downclorox.sol#891) is
not in mixedCase
Parameter PlateauFinance.calculateLiquidityFee(uint256)._amount
(downclorox.sol#905) is not in mixedCase
Variable PlateauFinance.BURN_FEE (downclorox.sol#820) is not in mixedCase
Variable PlateauFinance.LP_FEE (downclorox.sol#821) is not in mixedCase
Variable PlateauFinance._previousBurnFee (downclorox.sol#822) is not in mixedCase
Variable PlateauFinance._previousLPFee (downclorox.sol#823) is not in mixedCase
Variable PlateauFinance._excluded (downclorox.sol#827) is not in mixedCase
Variable PlateauFinance._pair (downclorox.sol#830) is not in mixedCase
Variable PlateauFinance.MAX_SUPPLY (downclorox.sol#831) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (downclorox.sol#491)" inContext
(downclorox.sol#485-494)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-
statements

Variable IJoeRouter01.addLiquidity(address,address,uint256,uint256,uint256,uint256,
address,uint256).amountADesired (downclorox.sol#15) is too similar to IJoeRouter01.
addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amoun
tBDesired (downclorox.sol#16)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-
names-are-too-similar

PlateauFinance._transfer(address,address,uint256) (downclorox.sol#1098-1154) uses
literals with too many digits:
 - _balances[address(0x000000000000000000000000000000000000dEaD)] += burnFee
(downclorox.sol#1144)
PlateauFinance._transfer(address,address,uint256) (downclorox.sol#1098-1154) uses
literals with too many digits:
 -
Transfer(msg.sender,address(0x000000000000000000000000000000000000dEaD),burnFee)
(downclorox.sol#1146)
```

```
PlateauFinance.slitherConstructorVariables() (downclorox.sol#807-1216) uses
literals with too many digits:
 - MAX_SUPPLY = 1000000000 * 10 ** decimals() (downclorox.sol#831)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-
digits

renounceOwnership() should be declared external:
 - Ownable.renounceOwnership() (downclorox.sol#761-764)
transferOwnership(address) should be declared external:
 - Ownable.transferOwnership(address) (downclorox.sol#770-774)
setFee(uint256,uint256) should be declared external:
 - PlateauFinance.setFee(uint256,uint256) (downclorox.sol#863-869)
changeLpReceiverAddress(address) should be declared external:
 - PlateauFinance.changeLpReceiverAddress(address) (downclorox.sol#871-873)
exclude(address,bool) should be declared external:
 - PlateauFinance.exclude(address,bool) (downclorox.sol#875-877)
setMaxLPCap(uint256) should be declared external:
 - PlateauFinance.setMaxLPCap(uint256) (downclorox.sol#879-881)
setSwapAndLiquify(bool) should be declared external:
 - PlateauFinance.setSwapAndLiquify(bool) (downclorox.sol#898-900)
name() should be declared external:
 - PlateauFinance.name() (downclorox.sol#918-920)
symbol() should be declared external:
 - PlateauFinance.symbol() (downclorox.sol#926-928)
totalSupply() should be declared external:
 - PlateauFinance.totalSupply() (downclorox.sol#950-952)
allowance(address,address) should be declared external:
 - PlateauFinance.allowance(address,address) (downclorox.sol#966-968)
approve(address,uint256) should be declared external:
 - PlateauFinance.approve(address,uint256) (downclorox.sol#977-981)
transfer(address,uint256) should be declared external:
 - PlateauFinance.transfer(address,uint256) (downclorox.sol#985-988)
transferFrom(address,address,uint256) should be declared external:
 - PlateauFinance.transferFrom(address,address,uint256) (downclorox.sol#1004-1012)
increaseAllowance(address,uint256) should be declared external:
 - PlateauFinance.increaseAllowance(address,uint256) (downclorox.sol#1026-1029)
decreaseAllowance(address,uint256) should be declared external:
 - PlateauFinance.decreaseAllowance(address,uint256) (downclorox.sol#1045-1051)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-
function-that-could-be-declared-external
. analyzed (11 contracts with 75 detectors), 84 result(s) found
```

HashEx
Blockchain Security