its__msn itsmsn

ITISH

AUDIT COMPANY

Audit Details

Contract Name
PWLC

Deployer address

0x91C1F07b7815d68c176321EaD61d7bFaE211d392

Client contacts:

PWLC team

Blockchain

Binance

Project website:

Not Provided By contract

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as 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 below disclaimer below – please make sure to read it in full.

DISCLAIMER: 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 Itish and its affiliates (includingholding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Itish) owe no duty of care towards you or any other person, nor does Itish 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 Itish 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, Itish hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Itish, 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 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.

Background

Itish was commissioned PWLC token to perform an audit of smart contracts:

https://bscscan.com/address/0x91C1F07b7815d68c176321EaD61d7bFaE211d392

The purpose of the audit was to achieve the following:

- Ensure that the smart contract functions as intended.
- Identify potential security issues with the smart contract.

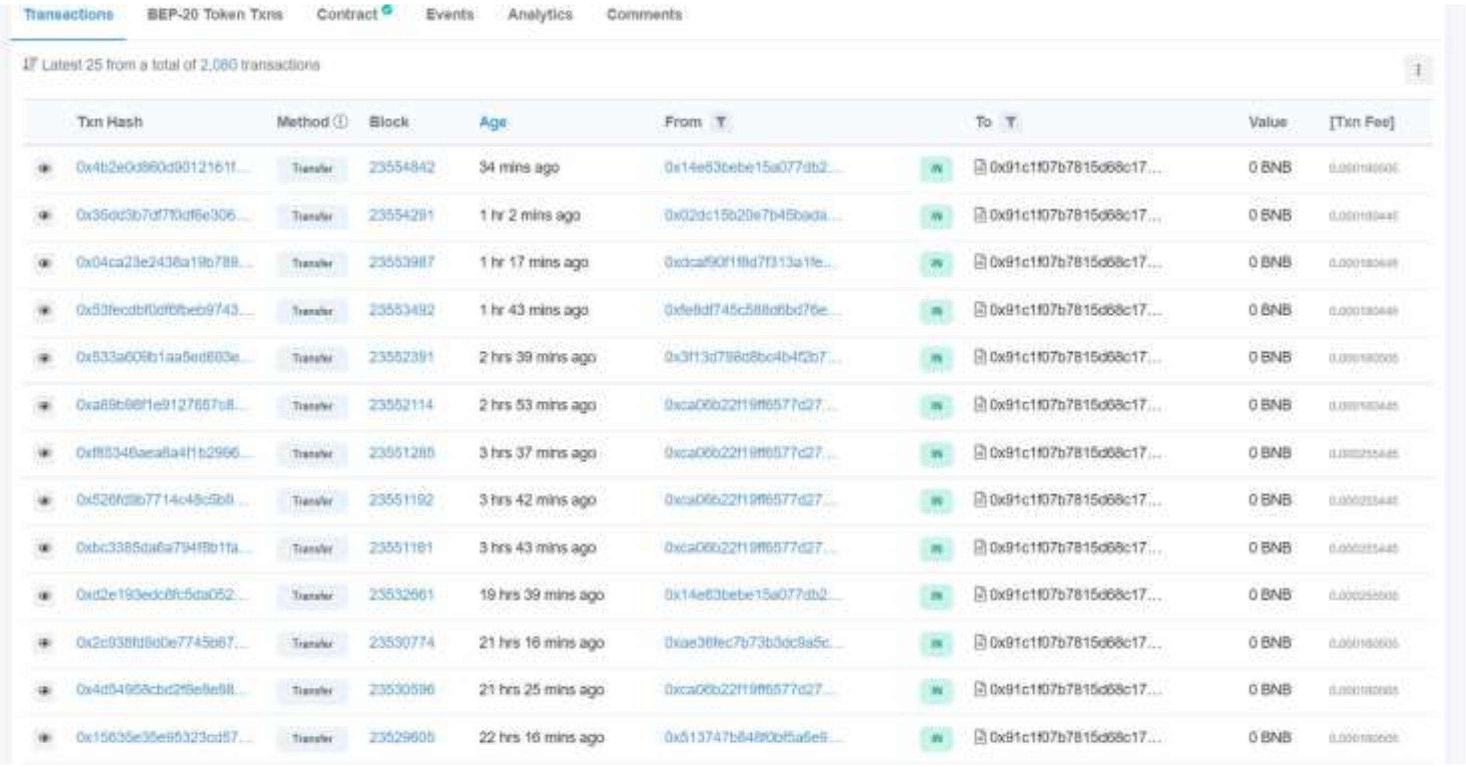
The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

Contract Details

Token contract details for 12.02.2022

contract name	**PWLC**
Contract creator	0x91C1F07b7815d68c176321EaD61d7bFaE211d392
Transaction's count	**2080**

Contract TopTransactions



Latest 25 from a total of 2,086 transactions

	Txn Hash	Method	Block	Age	From		To	Value	[Txn Fee]
👁	0x4b2e0d960d9012161f...	Transfer	23554842	34 mins ago	0x14e63bebe15a077db2...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000180605
👁	0x35dd3b7df7f0df6e306...	Transfer	23554291	1 hr 2 mins ago	0x02dc15b20e7b45bada...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000180445
👁	0x04ca23e2436a19b789...	Transfer	23553987	1 hr 17 mins ago	0xdcaf90f1f8d7f313a1fe...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000180449
👁	0x53fecdbf0df6fbeb9743...	Transfer	23553492	1 hr 43 mins ago	0xfe8df745c588d6bd76e...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000180449
👁	0x533a609b1aa5ed603e...	Transfer	23552391	2 hrs 39 mins ago	0x3f13d798d8bc4b4f2b7...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000180605
👁	0xa89b98f1e9127667c8...	Transfer	23552114	2 hrs 53 mins ago	0xca06b22f19ff6577d27...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000180445
👁	0xf85346aea8a4f1b2966...	Transfer	23551285	3 hrs 37 mins ago	0xca06b22f19ff6577d27...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000255445
👁	0x526fd9b7714c48c5b8...	Transfer	23551192	3 hrs 42 mins ago	0xca06b22f19ff6577d27...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000255445
👁	0xbc3385da6a794f8b1fa...	Transfer	23551181	3 hrs 43 mins ago	0xca06b22f19ff6577d27...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000255445
👁	0xd2e193edc8fc5da052...	Transfer	23532661	19 hrs 39 mins ago	0x14e63bebe15a077db2...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000255605
👁	0x2c938fd8d0e7745b67...	Transfer	23530774	21 hrs 16 mins ago	0xae36fec7b73b3dc9a5c...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000180605
👁	0x4d54958cbd2f8e8e88...	Transfer	23530596	21 hrs 25 mins ago	0xca06b22f19ff6577d27...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000180605
👁	0x15635e35e95323cd57...	Transfer	23529605	22 hrs 16 mins ago	0x513747b848f0bf5a6e8...	IN	0x91c1f07b7815d68c17...	0 BNB	0.000180605

Token Functions Details

```
totalSupply()
decimals()
symbol()
name()
getOwner()
balanceOf
transfer
allowance
approve
transferFrom
_msgData()
owner()
renounceOwnership()
```

Contract Interface Details

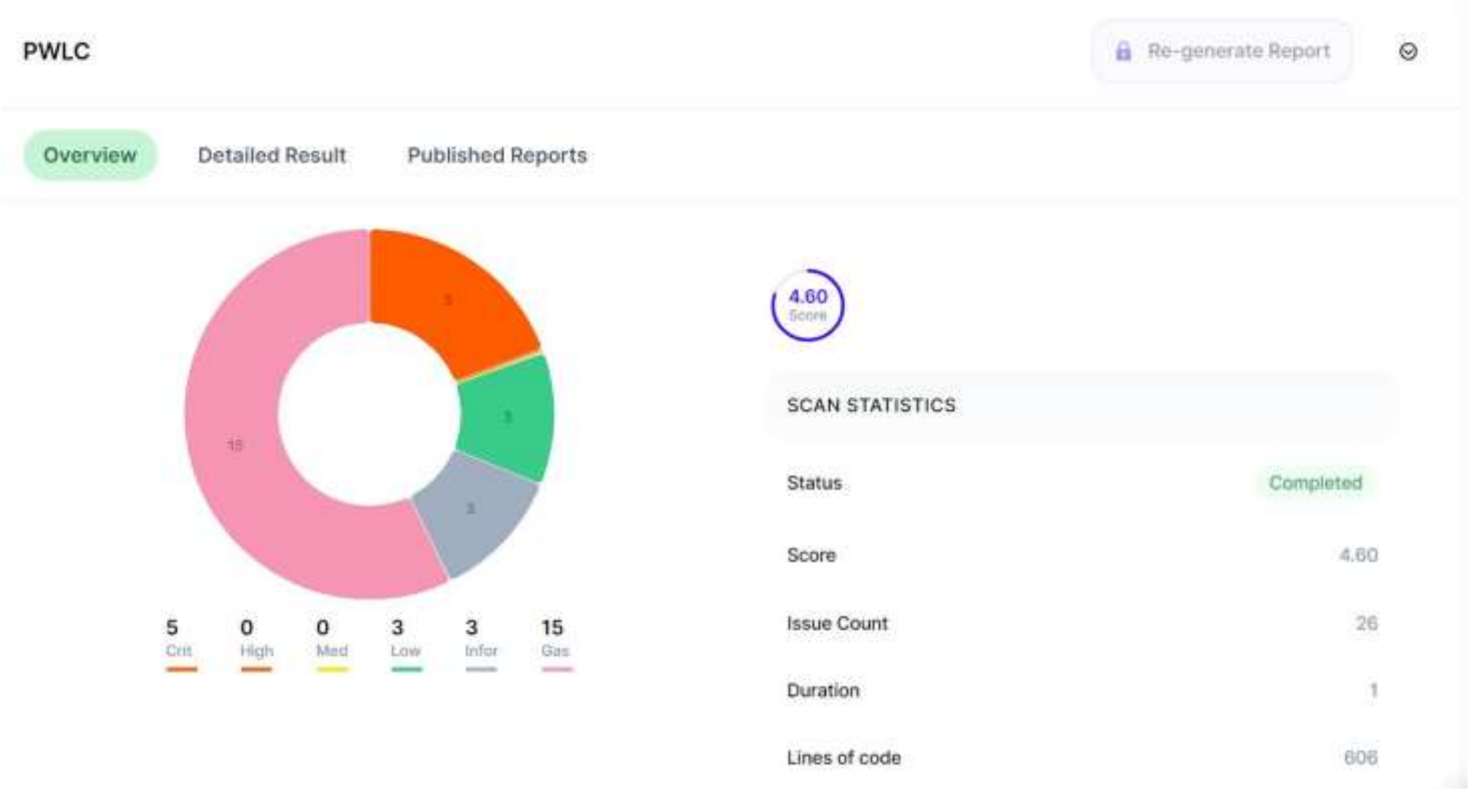
```
interface IERC20
interface IERC20Metadata is IERC20
```

Issues Checking Status

Issue description	Checking status
1. Compiler errors.	Passed
2. Compiler Compatibilities	failed
3. Possible delays in data delivery.	Passed
4. Oracle calls.	Moderate
5. Front running.	Failed
6. Timestamp dependence.	Passed
7. Integer Overflow and Underflow.	Passed
8. DoS with Revert.	Severe
9. DoS with block gas limit.	Moderate
10. Methods execution permissions.	Passed
11. Economy model of the contract.	Passed
12. The impact of the exchange rate on the logic.	Severe
13. Private user data leaks.	Passed
14. Malicious Event log.	Passed
15. Scoping and Declarations.	Passed
16. Uninitialized storage pointers.	Passed
17. Arithmetic accuracy.	passed
18. Design Logic.	poor

19.	**Cross-function race conditions.**	**Passed**
20.	Safe Open Zeppelin contracts implementation and usage.	pass
21.	**Fallback function security.**	**Failed**

Security Issues



SCAN STATISTICS

Status	Completed
Score	4.60
Issue Count	26
Duration	1
Lines of code	606

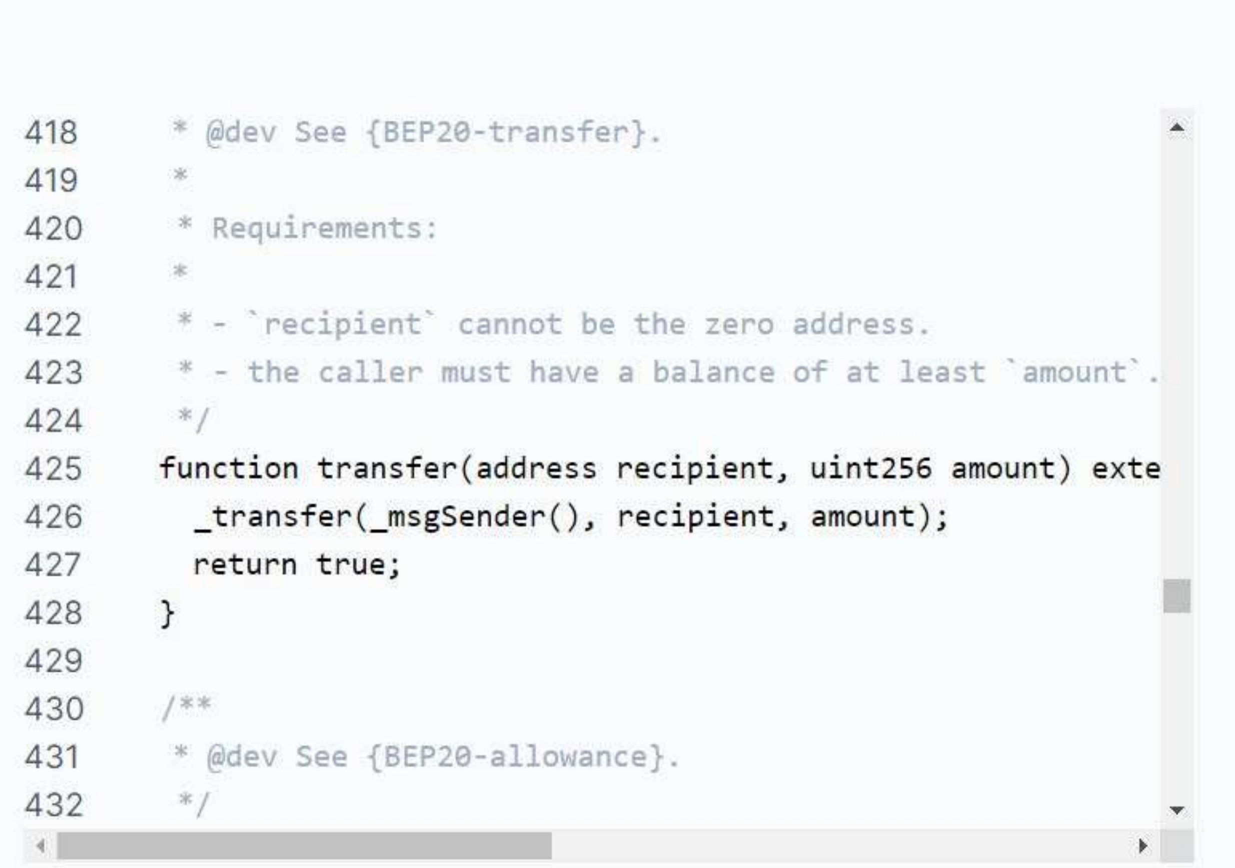
Critical Security Issues

Issue # 1:

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract `PWLC` is importing an access control library `@openzeppelin/contracts/access/Ownable.sol` but the function `transfer` is missing the modifier `onlyOwner`.



```
  * @dev See {BEP20-transfer}.
  *
  * Requirements:
  *
  * - `recipient` cannot be the zero address.
  * - the caller must have a balance of at least `amount`.
  */
 function transfer(address recipient, uint256 amount) exte
   _transfer(_msgSender(), recipient, amount);
   return true;
 }

 /**
  * @dev See {BEP20-allowance}.
  */
```

Remediation # 1:

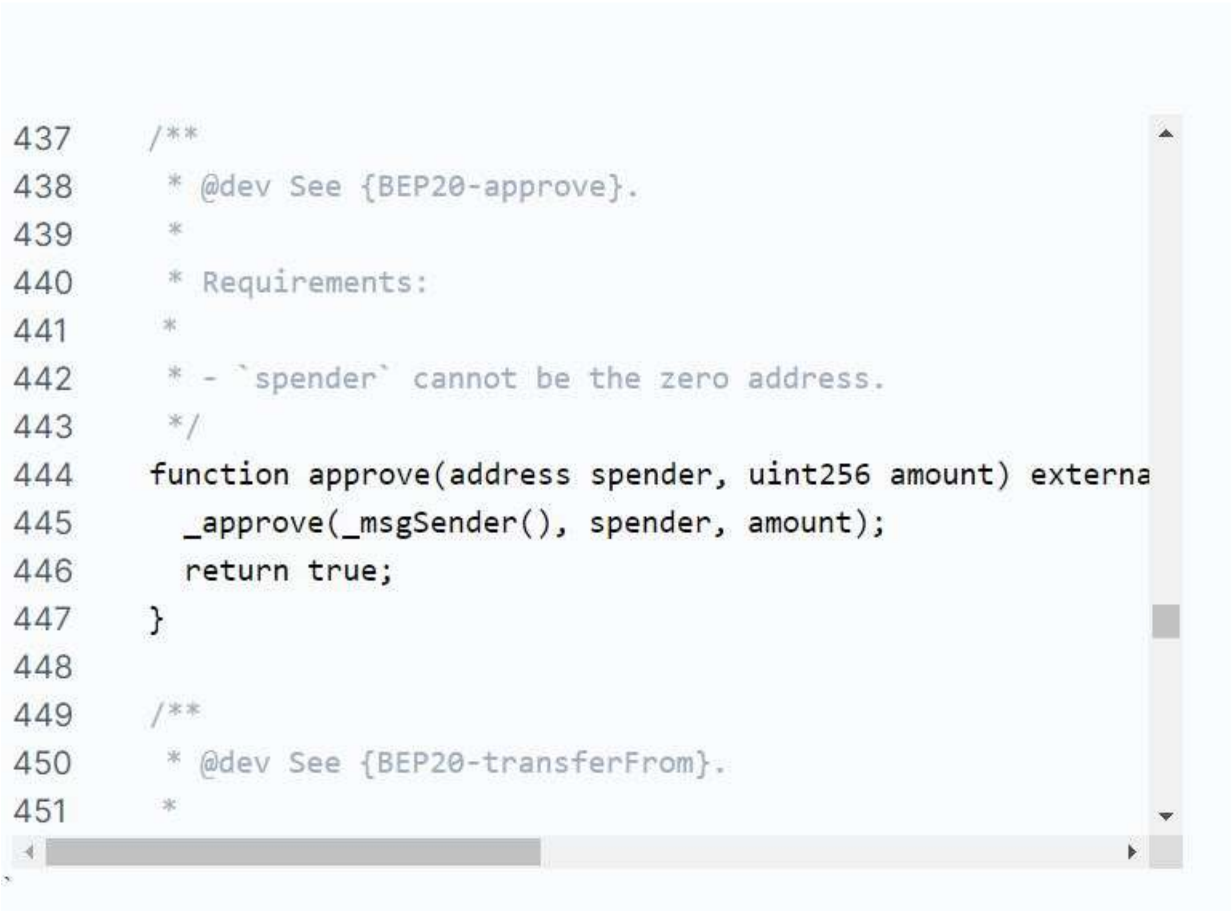
It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Type 2:

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract `PWLC` is importing an access control library `@openzeppelin/contracts/access/Ownable.sol` but the function `approve` is missing the modifier `onlyOwner`.



```
/**
 * @dev See {BEP20-approve}.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function approve(address spender, uint256 amount) externa
  _approve(_msgSender(), spender, amount);
  return true;
}

/**
 * @dev See {BEP20-transferFrom}.
 *
```

Remediation # 1:

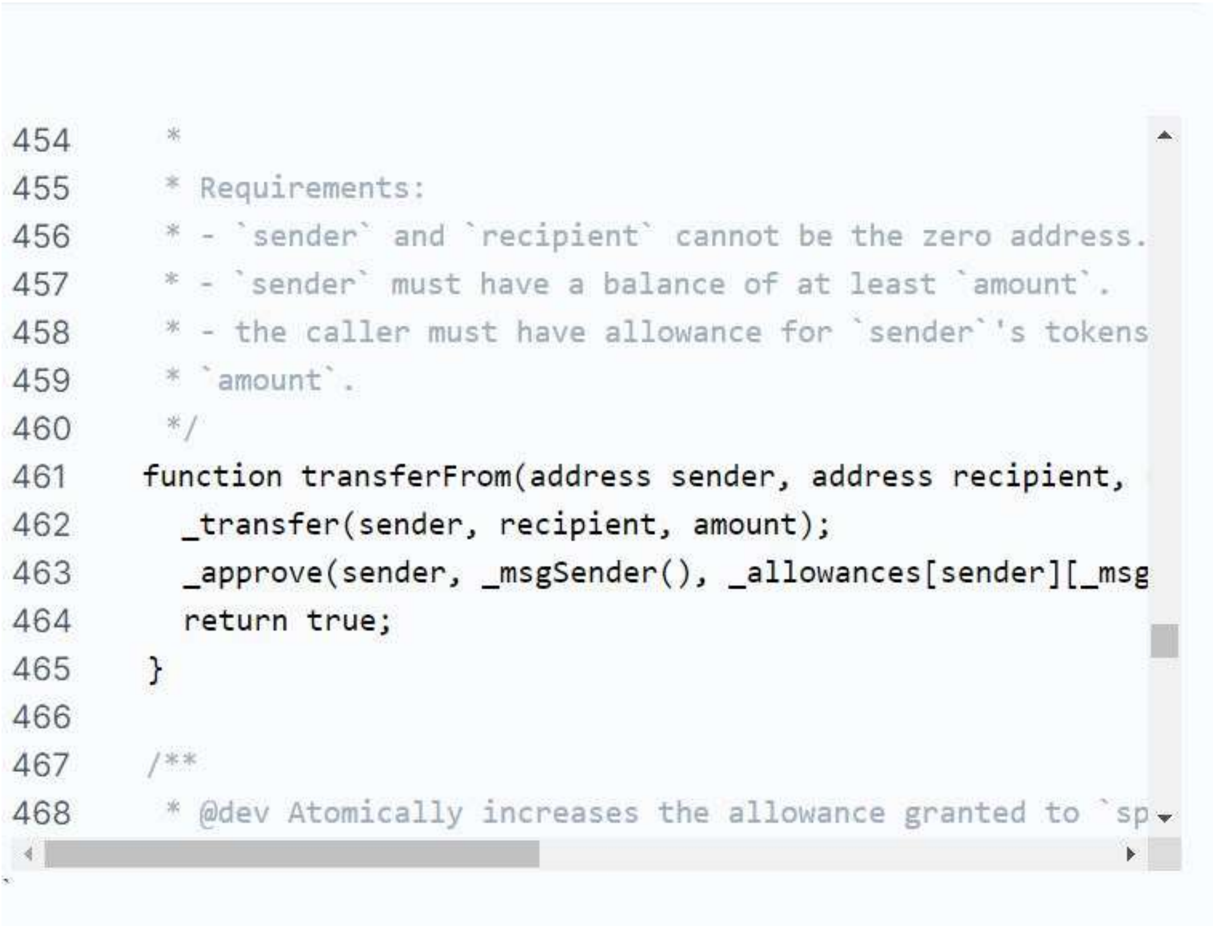
It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Type 3:

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract `PWLC` is importing an access control library `@openzeppelin/contracts/access/Ownable.sol` but the function `transferFrom` is missing the modifier `onlyOwner`.



```
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for `sender`'s tokens
     * `amount`.
     */
    function transferFrom(address sender, address recipient,
      _transfer(sender, recipient, amount);
      _approve(sender, _msgSender(), _allowances[sender][_msg
      return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `sp
```

Remediation # 1:

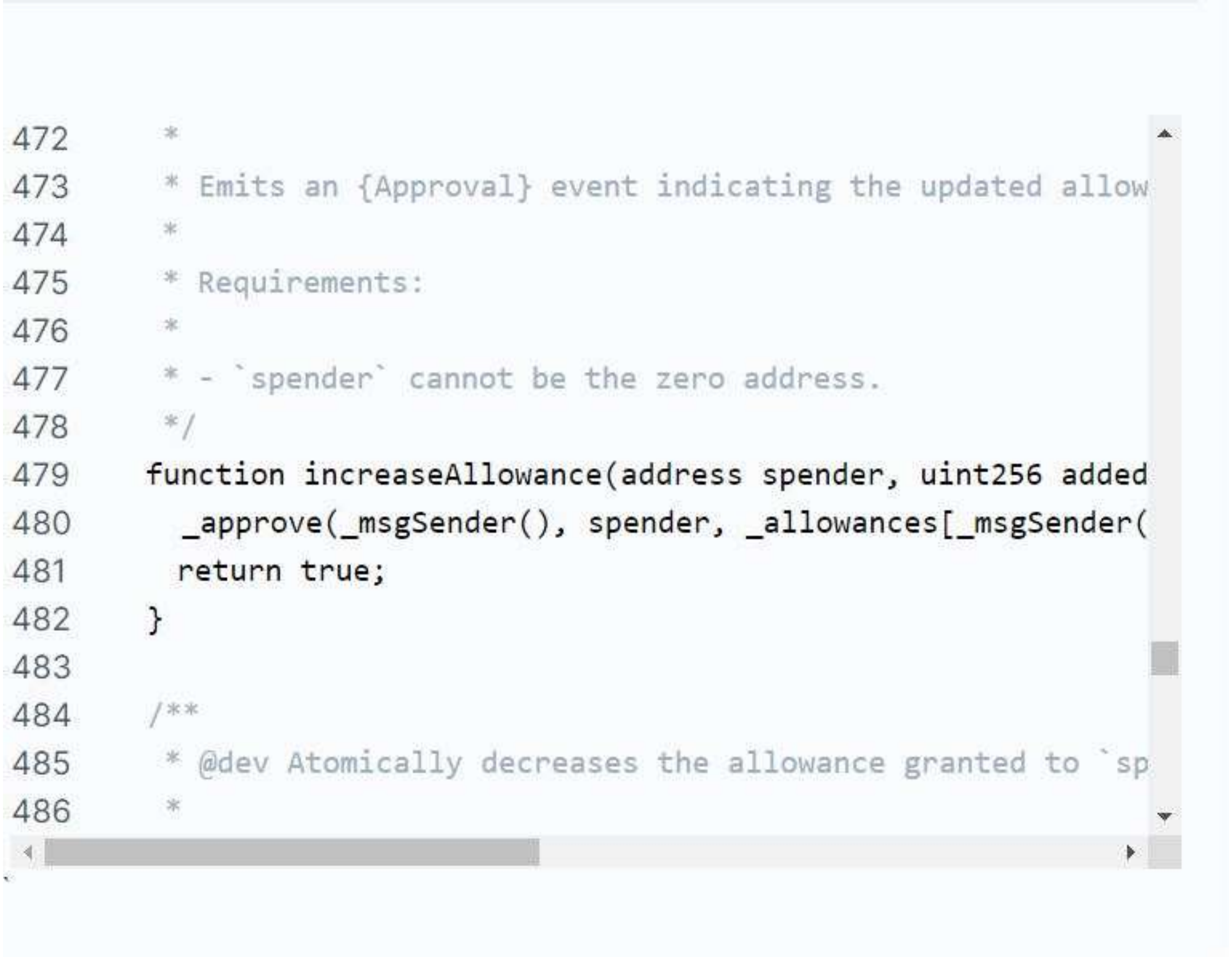
It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Type 4:

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract `PWLC` is importing an access control library `@openzeppelin/contracts/access/Ownable.sol` but the function `increaseAllowance` is missing the modifier `onlyOwner`.



```
 *
 * Emits an {Approval} event indicating the updated allow
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function increaseAllowance(address spender, uint256 added
  _approve(_msgSender(), spender, _allowances[_msgSender(
  return true;
}

/**
 * @dev Atomically decreases the allowance granted to `sp
 *
```

Remediation # 1:

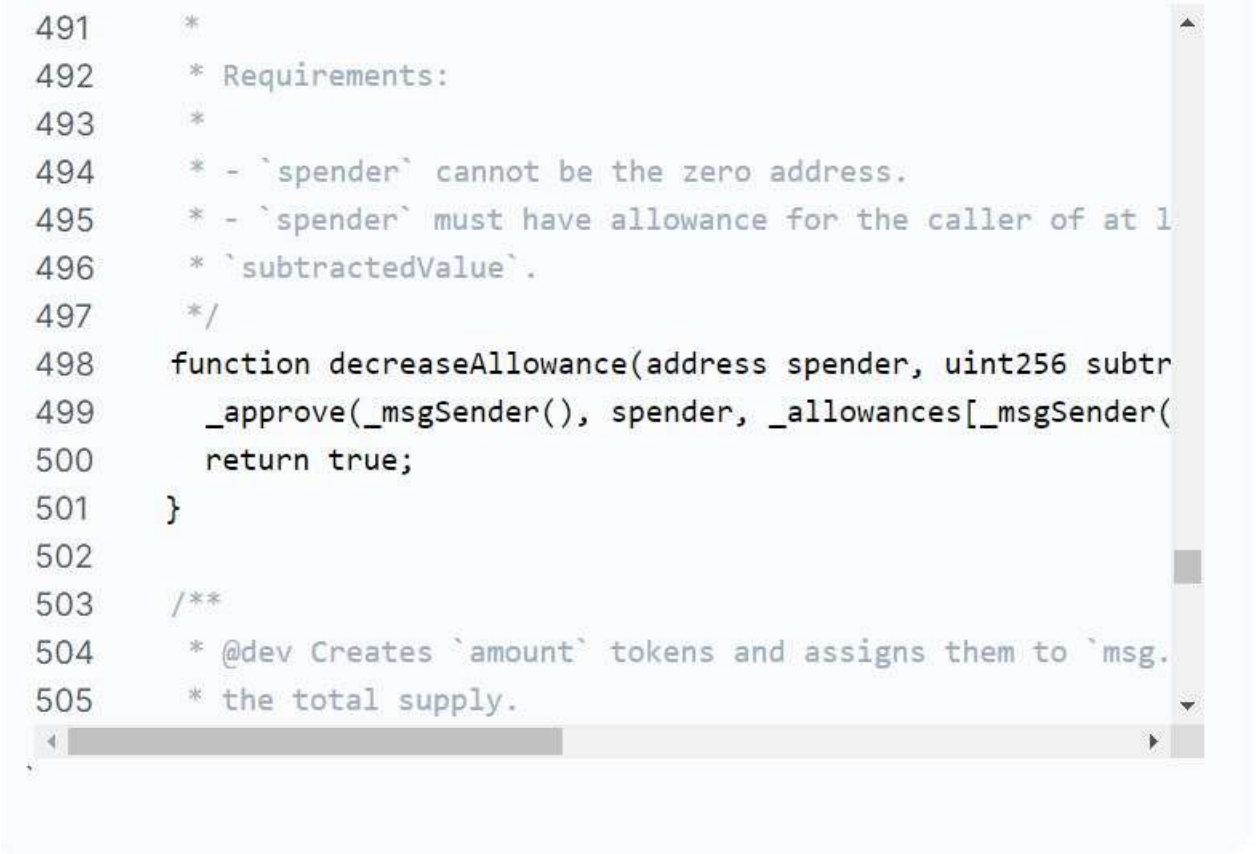
It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Type 5:

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract `PWLC` is importing an access control library `@openzeppelin/contracts/access/Ownable.sol` but the function `decreaseAllowance` is missing the modifier `onlyOwner`.



```
   *
   * Requirements:
   *
   * - `spender` cannot be the zero address.
   * - `spender` must have allowance for the caller of at l
   * `subtractedValue`.
   */
  function decreaseAllowance(address spender, uint256 subtr
    _approve(_msgSender(), spender, _allowances[_msgSender(
    return true;
  }

  /**
   * @dev Creates `amount` tokens and assigns them to `msg.
   * the total supply.
```

Remediation # 1:

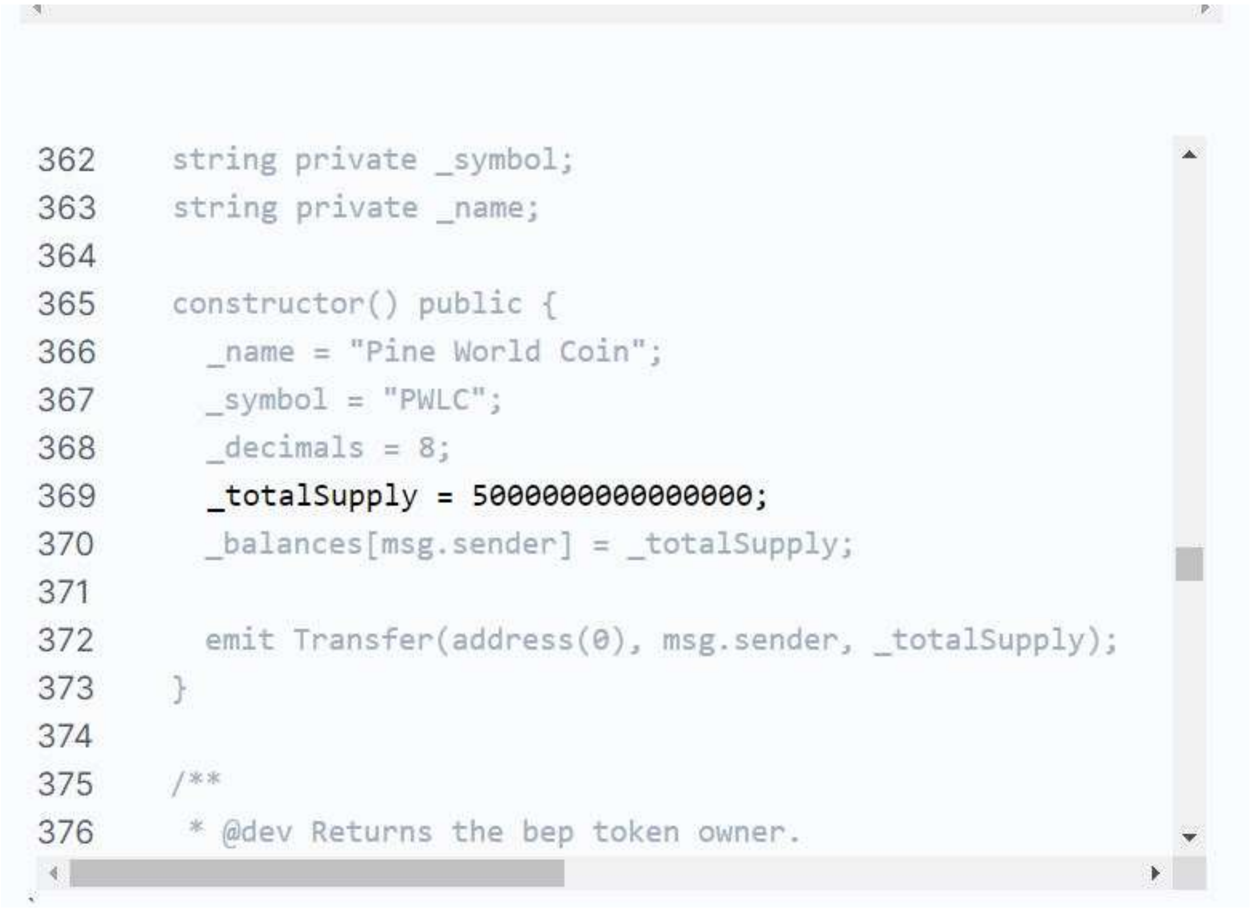
It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Low Severity Issues

Issue # 1:

LONG NUMBER LITERALS

Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different notation also supported by Solidity.
The value `5000000000000000` was detected on line `369`.



```
  string private _symbol;
  string private _name;

  constructor() public {
    _name = "Pine World Coin";
    _symbol = "PWLC";
    _decimals = 8;
    _totalSupply = 5000000000000000;
    _balances[msg.sender] = _totalSupply;

    emit Transfer(address(0), msg.sender, _totalSupply);
  }

  /**
   * @dev Returns the bep token owner.
```

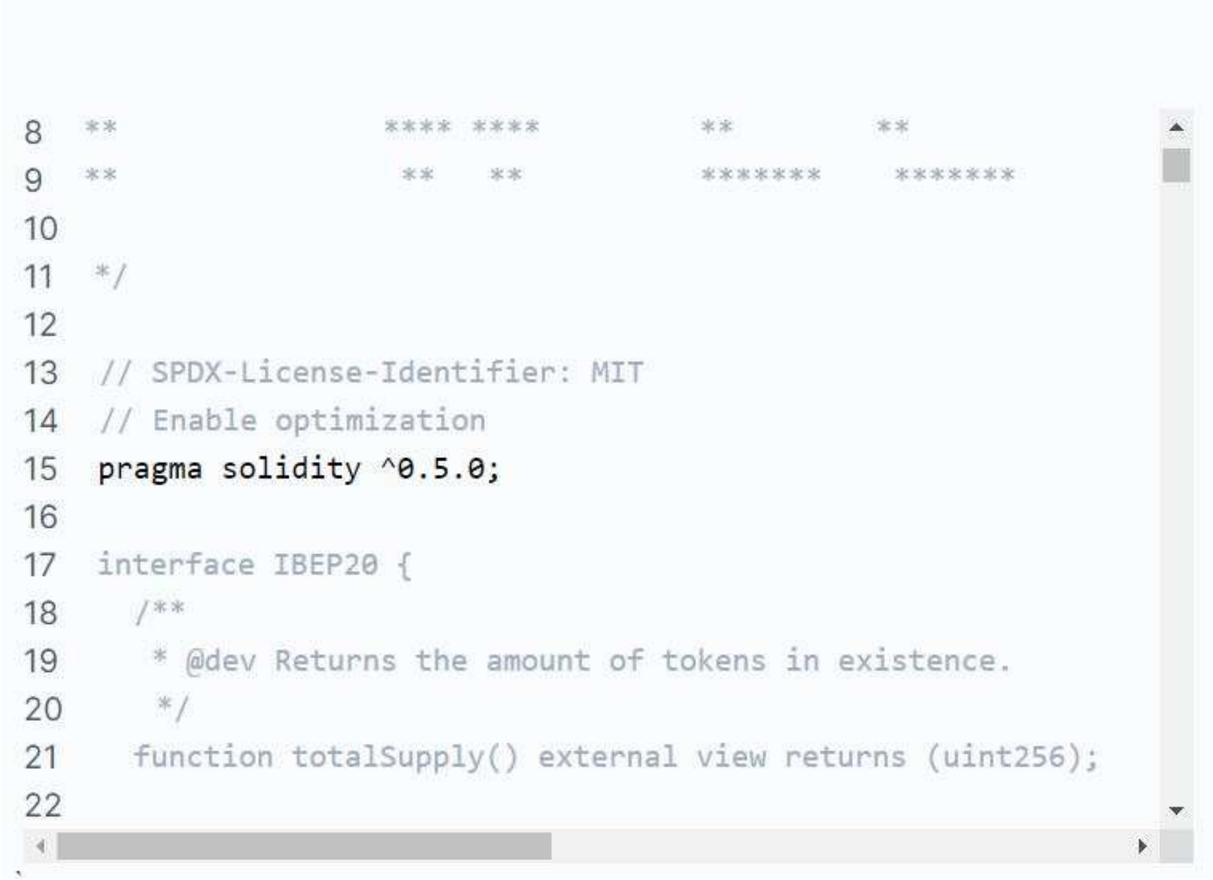
Remediation # 1:

Scientific notation in the form of `2e10` is also supported, where the mantissa can be fractional but the exponent has to be an integer. The literal `MeE` is equivalent to `M * 10**E`. Examples include `2e10`, `2e10`, `2e-10`, `2.5e1`, as suggested in official solidity documentation `https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals`

Issue # 2:

USE OF FLOATING PRAGMA

Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version.
The contract was found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described.
The following affected files were found to be using floating pragma:
`contract.sol - ^0.5.0`



```
**                    **** ****          **        **
**                     **   **           *******    *******

 */

// SPDX-License-Identifier: MIT
// Enable optimization
pragma solidity ^0.5.0;

interface IBEP20 {
  /**
   * @dev Returns the amount of tokens in existence.
   */
  function totalSupply() external view returns (uint256);

```

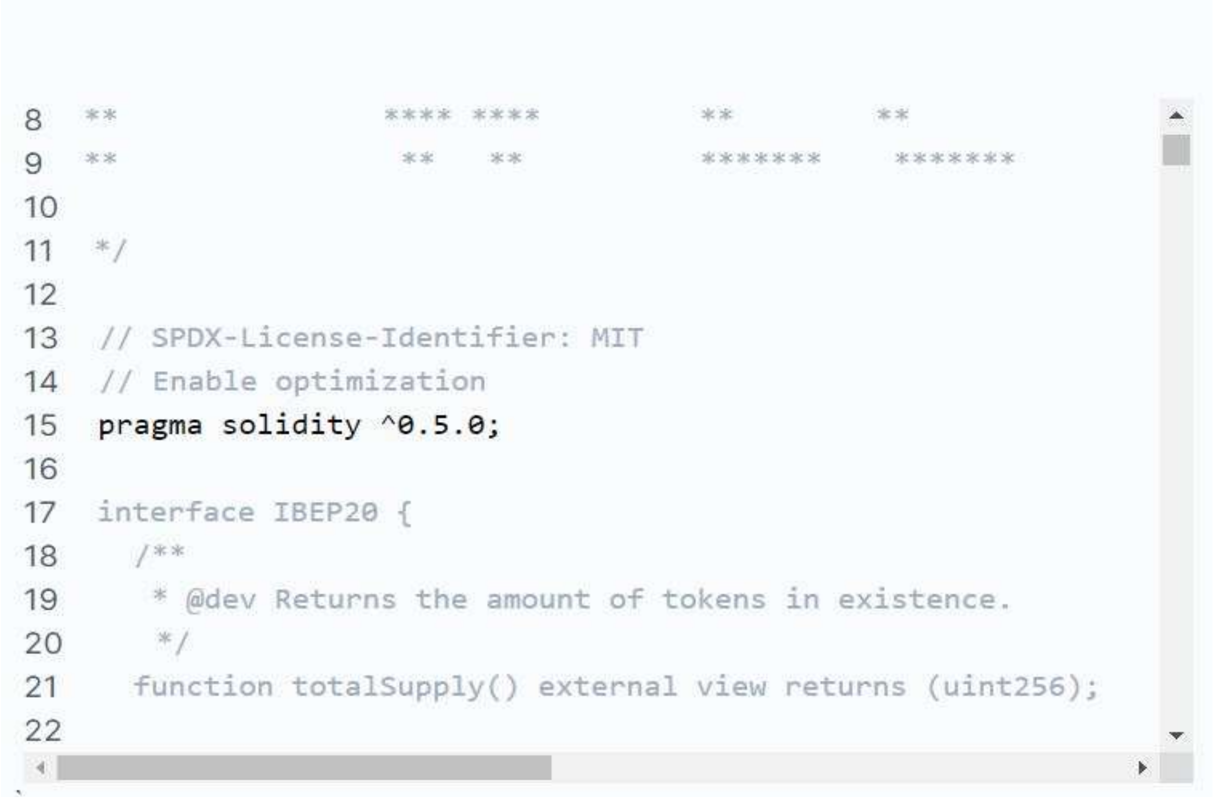
Remediation # 2:

It is recommended to use a fixed pragma version, as future compiler versions may handle certain language constructions in a way the developer did not foresee.
Using a floating pragma may introduce several vulnerabilities if compiled with an older version.
The developers should always use the exact Solidity compiler version when designing their contracts as it may break the changes in the future.
Instead of `^0.5.0` use `pragma solidity 0.8.7`, which is a stable and recommended version right now.

Issue # 3:

OUTDATED COMPILER VERSION

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.
The following outdated versions were detected:
`contract.sol` - `^0.5.0`



```
**                   **** ****           **          **
**                    **   **            *******     *******

 */

// SPDX-License-Identifier: MIT
// Enable optimization
pragma solidity ^0.5.0;

interface IBEP20 {
  /**
   * @dev Returns the amount of tokens in existence.
   */
  function totalSupply() external view returns (uint256);

```

Remediation # 3:

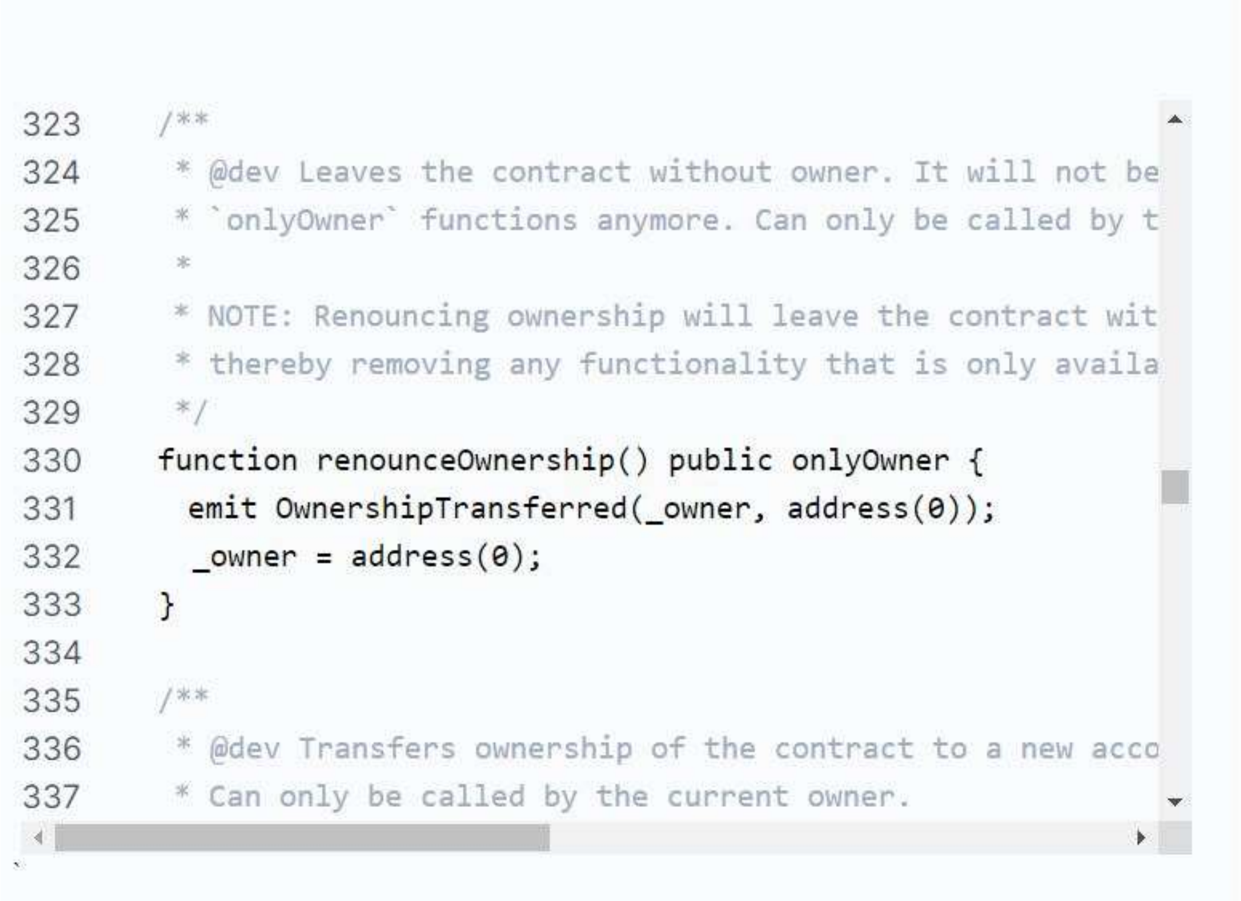
It is recommended to use a recent version of the Solidity compiler that should not be the most recent version, and it should not be an outdated version as well. Using very old versions of Solidity prevents the benefits of bug fixes and newer security checks. Consider using the solidity version `0.8.7`, which patches most solidity vulnerabilities.

Informative

1)

PRESENCE OF OVERPOWERED ROLE

The overpowered owner (i.e., the person who has too much power) is a project design where the contract is tightly coupled to their owner (or owners); only they can manually invoke critical functions.
Due to the fact that this function is only accessible from a single address, the system is heavily dependent on the address of the owner. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g., if the private key of this address is compromised, then an attacker can take control of the contract.



```
/**
 * @dev Leaves the contract without owner. It will not be
 * `onlyOwner` functions anymore. Can only be called by t
 *
 * NOTE: Renouncing ownership will leave the contract wit
 * thereby removing any functionality that is only availa
 */
function renounceOwnership() public onlyOwner {
  emit OwnershipTransferred(_owner, address(0));
  _owner = address(0);
}

/**
 * @dev Transfers ownership of the contract to a new acco
 * Can only be called by the current owner.
```

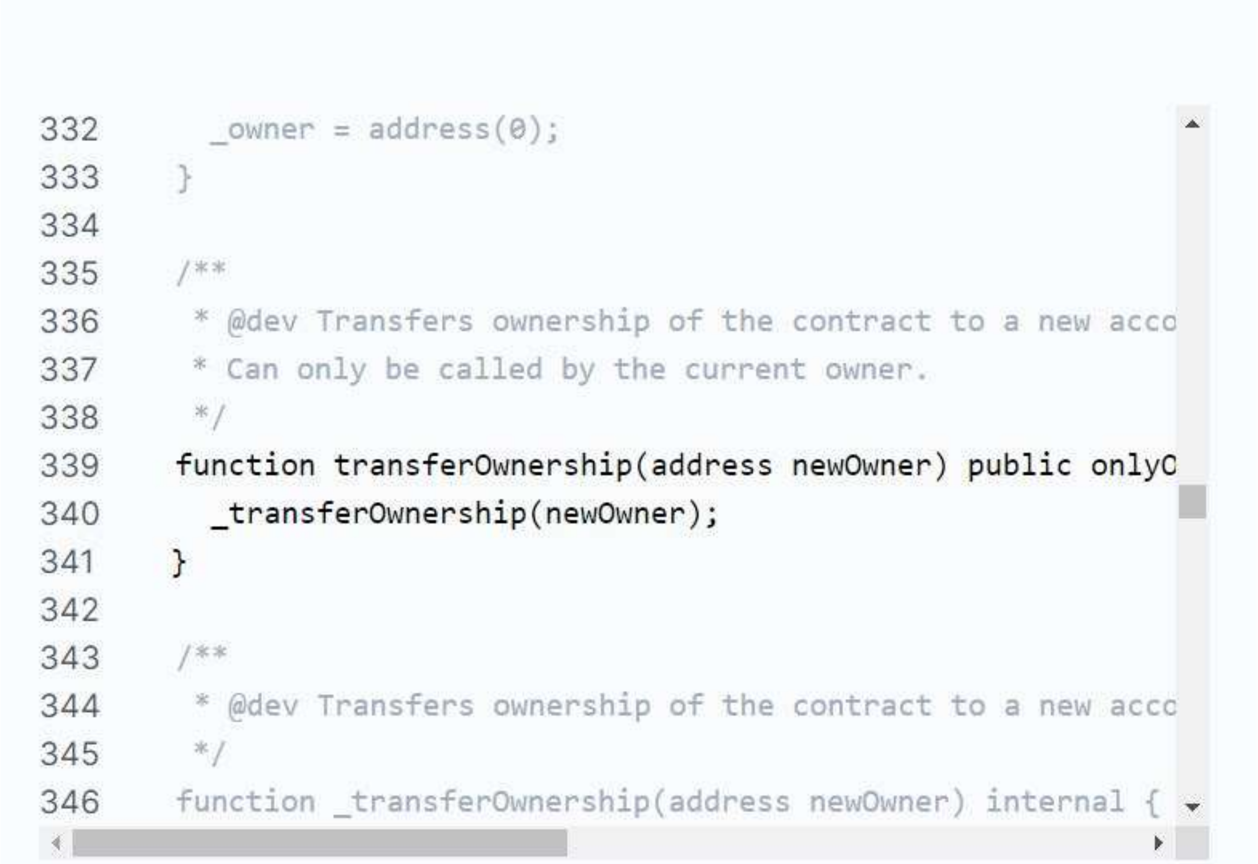
Remediation # 1:

We recommend designing contracts in a trust-less manner. For instance, this functionality can be implemented in the contract's constructor. Another option is to use a MultiSig wallet for this address. For systems that are provisioned for a single user, you can use [Ownable.sol].
For systems that require provisioning users in a group, you can use [@openzeppelin/Roles.sol] or [@hq20/Whitelist.sol].

Type 2

PRESENCE OF OVERPOWERED ROLE

The overpowered owner (i.e., the person who has too much power) is a project design where the contract is tightly coupled to their owner (or owners); only they can manually invoke critical functions.
Due to the fact that this function is only accessible from a single address, the system is heavily dependent on the address of the owner. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g., if the private key of this address is compromised, then an attacker can take control of the contract.



```
  _owner = address(0);
}

/**
 * @dev Transfers ownership of the contract to a new acco
 * Can only be called by the current owner.
 */
function transferOwnership(address newOwner) public onlyO
  _transferOwnership(newOwner);
}

/**
 * @dev Transfers ownership of the contract to a new acco
 */
function _transferOwnership(address newOwner) internal {
```

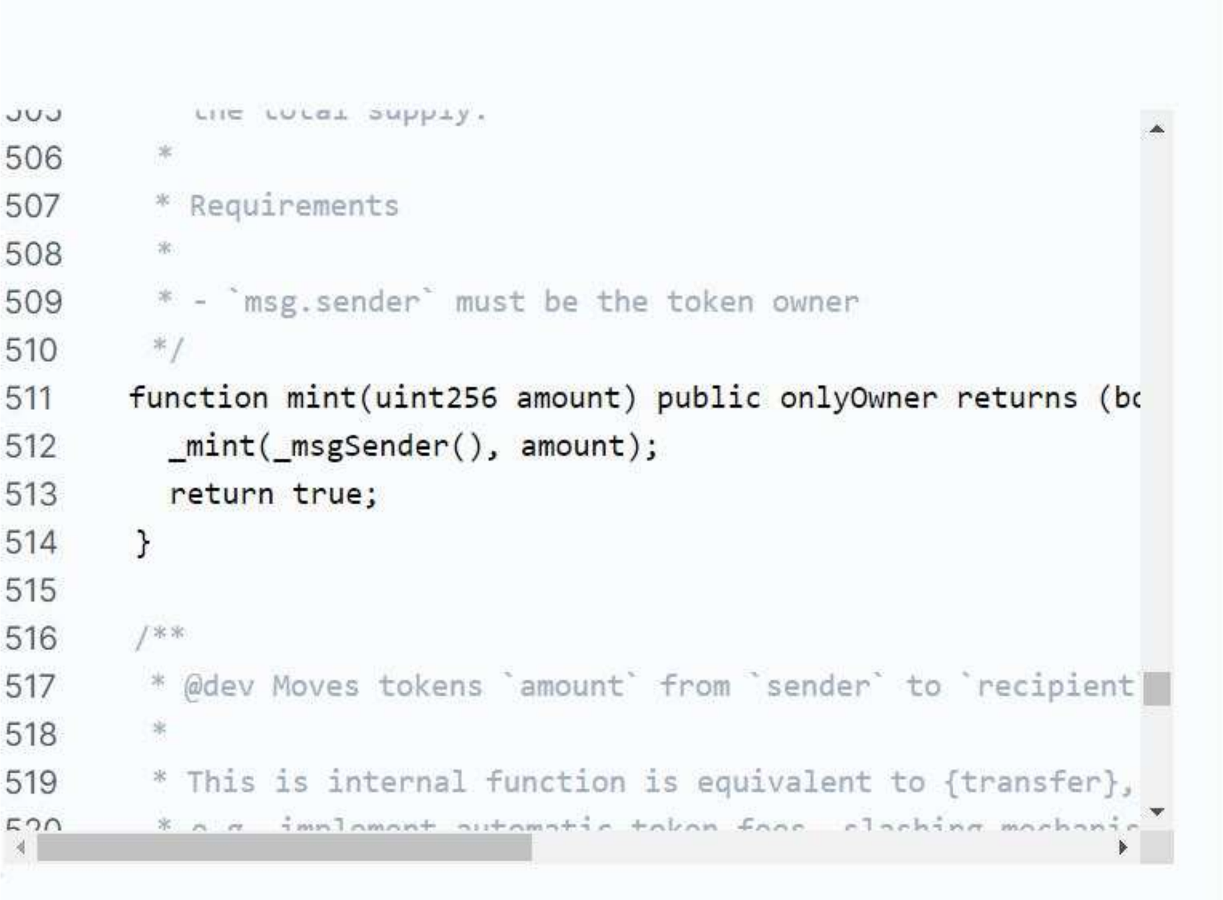
Remediation # 2:

We recommend designing contracts in a trust-less manner. For instance, this functionality can be implemented in the contract's constructor. Another option is to use a MultiSig wallet for this address. For systems that are provisioned for a single user, you can use [Ownable.sol].
For systems that require provisioning users in a group, you can use [@openzeppelin/Roles.sol] or [@hq20/Whitelist.sol].

Type 3

PRESENCE OF OVERPOWERED ROLE

The overpowered owner (i.e., the person who has too much power) is a project design where the contract is tightly coupled to their owner (or owners); only they can manually invoke critical functions.
Due to the fact that this function is only accessible from a single address, the system is heavily dependent on the address of the owner. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g., if the private key of this address is compromised, then an attacker can take control of the contract.



```
     * the total supply.
     *
     * Requirements
     *
     * - `msg.sender` must be the token owner
     */
    function mint(uint256 amount) public onlyOwner returns (bo
      _mint(_msgSender(), amount);
      return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`
     *
     * This is internal function is equivalent to {transfer},
     * e.g. implement automatic token fees, slashing mechanic
```

Remediation # 3:

We recommend designing contracts in a trust-less manner. For instance, this functionality can be implemented in the contract's constructor. Another option is to use a MultiSig wallet for this address. For systems that are provisioned for a single user, you can use [Ownable.sol].
For systems that require provisioning users in a group, you can use [@openzeppelin/Roles.sol] or [@hq20/Whitelist.sol].

Conclusion

Smart contracts contain High severity issues! Liquiditypair contract's security is not checked due to out of scope.

Liquidity locking details NOT provided by the team.

Itish note:

Please check the disclaimer above and note, the audit makes no statements or warranties on business model, investment attractiveness or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by Owner.