MythX

REPORT 606C7D53DE5409001944EB09

Created	Tue Apr 06 2021 15:25:07 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	takodefi@gmail.com

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
180faf39-6718-454c-bc66-55847effa456	TakoToken_Flat.sol	24

Analysis 180faf39-6718-454c-bc66-55847effa456 MythX

Started	Tue Apr 06 2021 15:25:15 GMT+0000 (Coordinated Universal Time)
Finished	Tue Apr 06 2021 16:11:14 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Remythx
Main Source File	TakoToken_Flat.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	13	11

ISSUES

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "renounceOwnership" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

TakoToken_Flat.sol

Locations

```
* thereby removing any functionality that is only available to the owner.
*/
function renounceOwnership() public virtual onlyOwner {
emit OwnershipTransferred(_owner, address(0));
_owner = address(0);
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "transferOwnership" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

TakoToken_Flat.sol

Locations

```
* Can only be called by the current owner.
*/
function transferOwnership(address newOwner) public virtual onlyOwner {
require(newOwner != address(0), "Ownable: new owner is the zero address");
emit OwnershipTransferred(_owner, newOwner);
_owner = newOwner;
}
}

```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "symbol" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

TakoToken_Flat.sol

Locations

```
* name.
*/
function symbol() public override view returns (string memory) {
return _symbol;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "decimals" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

TakoToken_Flat.sol

Locations

```
* @dev Returns the number of decimals used to get its user representation.
*/
function decimals() public override view returns (uint8) {
return _decimals;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "totalSupply" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`TakoToken_Flat.sol`

Locations

```
* @dev See {BEP20-totalSupply}.
*/
function totalSupply() public override view returns (uint256) {
return _totalSupply;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "transfer" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`TakoToken_Flat.sol`

Locations

```
* - the caller must have a balance of at least `amount`.
*/
function transfer(address recipient, uint256 amount) public override returns (bool) {
_transfer(_msgSender(), recipient, amount);
return true;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "allowance" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`TakoToken_Flat.sol`

Locations

```
* @dev See {BEP20-allowance}.
*/
function allowance(address owner, address spender) public override view returns (uint256) {
return _allowances[owner][spender];
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "approve" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

TakoToken_Flat.sol

Locations

```
* - `spender` cannot be the zero address.
*/
function approve(address spender, uint256 amount) public override returns (bool) {
_approve(_msgSender(), spender, amount);
return true;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "transferFrom" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

TakoToken_Flat.sol

Locations

```
* `amount`.
*/
function transferFrom (address sender, address recipient, uint256 amount) public override returns (bool) {
_transfer(sender, recipient, amount);
_approve(
sender,
_msgSender(),
_allowances[sender][_msgSender()].sub(amount, 'BEP20: transfer amount exceeds allowance')
);
return true;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "increaseAllowance" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

TakoToken_Flat.sol

Locations

```
* - `spender` cannot be the zero address.
*/
function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
_approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
return true;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "decreaseAllowance" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

TakoToken_Flat.sol

Locations

```
* `subtractedValue`.
*/
function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
_approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, 'BEP20: decreased allowance below zero'));
return true;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "mint" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

TakoToken_Flat.sol

Locations

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

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "mint" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

TakoToken_Flat.sol

Locations

```
contract TakoToken is BEP20('Tako Token', 'TAKO') {
/// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
function mint(address _to, uint256 _amount) public onlyOwner {
_mint(_to, _amount);
_moveDelegates(address(0), _delegates[_to], _amount);
}

// Copied and modified from YAM code:
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is "">=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`TakoToken_Flat.sol`

Locations

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/*
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is "">=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`TakoToken_Flat.sol`

Locations

```


pragma solidity >=0.6.0 <0.8.0;

/**
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is "">=0.6.4"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`TakoToken_Flat.sol`

Locations

```
// File: contracts\libs\IBEP20.sol

pragma solidity >=0.6.4;

interface IBEP20 {
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is "">=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`TakoToken_Flat.sol`

Locations

```


pragma solidity >=0.6.0 <0.8.0;

/**
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is "">=0.4.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`TakoToken_Flat.sol`

Locations

```


pragma solidity >=0.4.0;


```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`TakoToken_Flat.sol`

Locations

```
require(signatory != address(0), "TAKO::delegateBySig: invalid signature");
require(nonce == nonces[signatory]++, "TAKO::delegateBySig: invalid nonce");
require(now <= expiry, "TAKO::delegateBySig: signature expired");
return _delegate(signatory, delegatee);
}
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`TakoToken_Flat.sol`

Locations

```
returns (uint256)
{
require(blockNumber < block.number, "TAKO::getPriorVotes: not yet determined");

uint32 nCheckpoints = numCheckpoints[account];
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`TakoToken_Flat.sol`

Locations

```
internal
{
uint32 blockNumber = safe32(block.number, "TAKO::_writeCheckpoint: block number exceeds 32 bits");

if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
```

LOW

SWC-120

A control flow decision is made based on The block.number environment variable.

The block.number environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`TakoToken_Flat.sol`

Locations

```
returns (uint256)
{
require(blockNumber < block.number, "TAKO::getPriorVotes: not yet determined");

uint32 nCheckpoints = numCheckpoints[account];
```

LOW

SWC-128

Potentially unbounded data structure passed to builtin.

Gas consumption in function "delegateBySig" in contract "TakoToken" depends on the size of data structures that may grow unboundedly. Specifically the "1-st" argument to builtin "keccak256" may be able to grow unboundedly causing the builtin to consume more gas than the block gas limit, effectively causing a denial-of-service condition.Consider that an attacker might attempt to cause this condition on purpose.

Source file

`TakoToken_Flat.sol`

Locations

```
abi.encode(
DOMAIN_TYPEHASH,
keccak256(bytes(name())),
getChainId(),
address(this)
```

LOW

SWC-128

Loop over unbounded data structure.

Gas consumption in function "getPriorVotes" in contract "TakoToken" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

`TakoToken_Flat.sol`

Locations

```
uint32 lower = 0;
uint32 upper = nCheckpoints - 1;
while (upper > lower) {
uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
Checkpoint memory cp = checkpoints[account][center];
```