# MythX

## Analysis c69f4e65-bd66-4be0-a969-217aec2339cb

MythX

| | |
|---|---|
| Started | Sun Apr 10 2022 17:40:29 GMT+0000 (Coordinated Universal Time) |
| Finished | Sun Apr 10 2022 17:40:35 GMT+0000 (Coordinated Universal Time) |
| Mode | Deep |
| Client Tool | Remythx |
| Main Source File | Contracts/Defla.Sol |

## DETECTED VULNERABILITIES

| HIGH | MEDIUM | LOW |
|---|---|---|
| 0 | 0 | 9 |

## ISSUES

### UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/defla.sol

Locations

```

function add(uint256 a, uint256 b) internal pure returns (uint256) {
uint256 c = a + b;
require(c >= a, "SafeMath: addition overflow");
return c;
```

### UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/defla.sol

Locations

```
function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
require(b <= a, errorMessage);
uint256 c = a - b;

return c;
```

UNKNOWN

SWC-101

### Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```
}

uint256 c = a * b;
require(c / a == b, "SafeMath: multiplication overflow");
```

UNKNOWN

SWC-101

### Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```

uint256 c = a * b;
require(c / a == b, "SafeMath: multiplication overflow");

return c;
```

UNKNOWN

SWC-101

### Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```
function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
require(b > 0, errorMessage);
uint256 c = a / b;
return c;
}
```

UNKNOWN

SWC-101

Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```

function mul(int256 a, int256 b) internal pure returns (int256) {
int256 c = a * b;

require(c != MIN_INT256 || (a & MIN_INT256) != (b & MIN_INT256));
```

UNKNOWN

SWC-101

Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```

require(c != MIN_INT256 || (a & MIN_INT256) != (b & MIN_INT256));
require((b == 0) || (c / b == a));
return c;
}
```

UNKNOWN

SWC-101

Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```
require(b != -1 || a != MIN_INT256);

return a / b;
}
```

UNKNOWN

SWC-101

## Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/defla.sol

Locations

```

function sub(int256 a, int256 b) internal pure returns (int256) {
int256 c = a - b;
require((b >= 0 && c <= a) || (b < 0 && c > a));
return c;
```

UNKNOWN

SWC-101

## Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/defla.sol

Locations

```

function add(int256 a, int256 b) internal pure returns (int256) {
int256 c = a + b;
require((b >= 0 && c >= a) || (b < 0 && c < a));
return c;
```

UNKNOWN

SWC-101

## Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/defla.sol

Locations

```
uint256 total;
Batch[] memory _rLock = _rLocks[_address];
for(uint256 i=0 ; i< _rLock.length; i++){
uint256 _timestamp = lockTimestamps[_rLock[i]._lockContract];
if(block.timestamp <= _timestamp){
```

UNKNOWN

SWC-101

## Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```
uint256 count = batches.length;
Batch[] memory returnBatches = new Batch[](count);
for(uint256 i = 0; i < batches.length; i++){
returnBatches[i]._lockContract = batches[i]._lockContract;
returnBatches[i]._value = batches[i]._value.div(rate);
```

UNKNOWN

SWC-101

## Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```
InterfaceLP public pairContract;

uint256 private constant INITIAL_SUPPLY = 10000000*10**_decimals;
uint256 public rate;
uint256 public _totalSupply;
```

UNKNOWN

SWC-101

## Arithmetic operation "**" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```
InterfaceLP public pairContract;

uint256 private constant INITIAL_SUPPLY = 10000000*10**_decimals;
uint256 public rate;
uint256 public _totalSupply;
```

UNKNOWN

SWC-101

## Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```
uint256 private constant MAX_UINT256 = ~uint256(0);
uint256 private constant MAX_SUPPLY = ~uint128(0);
uint256 private constant rSupply = MAX_UINT256 - (MAX_UINT256 % INITIAL_SUPPLY);

uint BURN_TAX = 2; // 2% burn
```

UNKNOWN

SWC-101

## Arithmetic operation "%" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```
uint256 private constant MAX_UINT256 = ~uint256(0);
uint256 private constant MAX_SUPPLY = ~uint128(0);
uint256 private constant rSupply = MAX_UINT256 - (MAX_UINT256 % INITIAL_SUPPLY);

uint BURN_TAX = 2; // 2% burn
```

UNKNOWN

SWC-101

## Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```
_transferFrom(msg.sender, recipient, amount);
} else {
uint burnAmount = amount.mul(BURN_TAX) / 100;
_transferFrom(msg.sender, recipient, amount.sub(burnAmount));
_transferFrom(msg.sender, DEAD, burnAmount);
```

UNKNOWN

SWC-101

## Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/defla.sol`

Locations

```
_transferFrom(sender, recipient, amount);
} else {
uint burnAmount = amount.mul(BURN_TAX) / 100;
_transferFrom(msg.sender, recipient, amount.sub(burnAmount));
_transferFrom(msg.sender, DEAD, burnAmount);
```

LOW

SWC-103

## A floating pragma is set.

The current pragma Solidity directive is ""^0.8.10"". 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

`contracts/defla.sol`

Locations

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

/*
```

LOW

SWC-108

## State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "WBNB" is internal. Other possible visibility settings are public and private.

Source file

`contracts/defla.sol`

Locations

```
using SafeMathInt for int256;

address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c; // Wrapped BNB
address DEAD = 0x000000000000000000000000000000000000dEaD; // DEAD
```

LOW

SWC-108

### State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "DEAD" is internal. Other possible visibility settings are public and private.

Source file

`contracts/defla.sol`

Locations

```

address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c; // Wrapped BNB
address DEAD = 0x000000000000000000000000000000000000dEaD; // DEAD

string constant _name = "Deflationary Finance";
```

LOW

SWC-108

### State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "_rBalance" is internal. Other possible visibility settings are public and private.

Source file

`contracts/defla.sol`

Locations

```
uint8 constant _decimals = 18;

mapping (address => uint256) _rBalance;
mapping (address => mapping (address => uint256)) _allowances;
```

LOW

SWC-108

### State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "_allowances" is internal. Other possible visibility settings are public and private.

Source file

`contracts/defla.sol`

Locations

```

mapping (address => uint256) _rBalance;
mapping (address => mapping (address => uint256)) _allowances;

mapping (address => bool) lockContracts;
```

LOW

SWC-108

## State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "lockContracts" is internal. Other possible visibility settings are public and private.

Source file

`contracts/defla.sol`

Locations

```
mapping (address => mapping (address => uint256)) _allowances;

mapping (address => bool) lockContracts;
mapping (address => uint256) lockTimestamps;
struct Batch{
```

LOW

SWC-108

## State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "lockTimestamps" is internal. Other possible visibility settings are public and private.

Source file

`contracts/defla.sol`

Locations

```

mapping (address => bool) lockContracts;
mapping (address => uint256) lockTimestamps;
struct Batch{
address _lockContract;
```

LOW

SWC-108

## State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "_rLocks" is internal. Other possible visibility settings are public and private.

Source file

`contracts/defla.sol`

Locations

```
uint256 _value;
}
mapping(address => Batch[]) _rLocks;
function setLockContract(address _contractAddress, uint256 _timestamp) external onlyOwner{
require(isContract(_contractAddress), "SafeBEP20: call to non-contract");
```

LOW

SWC-108

## State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "BURN_TAX" is internal. Other possible visibility settings are public and private.

Source file

`contracts/defla.sol`

Locations

```
uint256 private constant rSupply = MAX_UINT256 - (MAX_UINT256 % INITIAL_SUPPLY);

uint BURN_TAX = 2; // 2% burn
address public admin;
mapping(address => bool) public excludedFromTax;
```

UNKNOWN

SWC-110

## Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/defla.sol`

Locations

```
Batch[] memory _rLock = _rLocks[_address];
for(uint256 i=0 ; i< _rLock.length; i++){
uint256 _timestamp = lockTimestamps[_rLock[i]._lockContract];
if(block.timestamp <= _timestamp){
total = total.add(_rLock[i]._value);
```

UNKNOWN

SWC-110

## Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/defla.sol`

Locations

```
uint256 _timestamp = lockTimestamps[_rLock[i]._lockContract];
if(block.timestamp <= _timestamp){
total = total.add(_rLock[i]._value);
}
}
```

UNKNOWN

SWC-110

## Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/defla.sol

Locations

```
Batch[] memory returnBatches = new Batch[](count);
for(uint256 i = 0; i < batches.length; i++){
returnBatches[i]._lockContract = batches[i]._lockContract;
returnBatches[i]._value = batches[i]._value.div(rate);
}
```

UNKNOWN

SWC-110

## Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/defla.sol

Locations

```
Batch[] memory returnBatches = new Batch[](count);
for(uint256 i = 0; i < batches.length; i++){
returnBatches[i]._lockContract = batches[i]._lockContract;
returnBatches[i]._value = batches[i]._value.div(rate);
}
```

UNKNOWN

SWC-110

## Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/defla.sol

Locations

```
for(uint256 i = 0; i < batches.length; i++){
returnBatches[i]._lockContract = batches[i]._lockContract;
returnBatches[i]._value = batches[i]._value.div(rate);
}
return returnBatches;
```

UNKNOWN

SWC-110

## Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/defla.sol`

Locations

```
for(uint256 i = 0; i < batches.length; i++){
returnBatches[i]._lockContract = batches[i]._lockContract;
returnBatches[i]._value = batches[i]._value.div(rate);
}
return returnBatches;
```