MythX

REPORT 60D34DE94C7BEC00185C8604

Created	Wed Jun 23 2021 15:06:17 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	60d316478bfa1246dff293e4

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
7f82b89f-b0fa-4b39-8223-19bd22141761	contracts/WhaleToken.sol	38

Analysis 7f82b89f-b0fa-4b39-8223-19bd22141761 MythX

Started	Wed Jun 23 2021 15:06:19 GMT+0000 (Coordinated Universal Time)
Finished	Wed Jun 23 2021 15:23:10 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Client Tool	Remythx
Main Source File	Contracts/WhaleToken.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	22	16

ISSUES

MEDIUM Function could be marked as external.

SWC-000

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

contracts/WhaleToken.sol

Locations

```
805 * thereby removing any functionality that is only available to the owner.
806 */
807 function renounceOwnership() public virtual onlyOwner {
808 emit OwnershipTransferred(_owner, address(0));
809 _owner = address(0);
810 }
811
812 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

contracts/WhaleToken.sol

Locations

```
814 * Can only be called by the current owner.
815 */
816 function transferOwnership(address newOwner) public virtual onlyOwner {
817 require(newOwner != address(0), "Ownable: new owner is the zero address");
818 emit OwnershipTransferred(_owner, newOwner);
819 _owner = newOwner;
820 }
821 }
```

MEDIUM

Function could be marked as external.

SWC-000

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

contracts/WhaleToken.sol

Locations

```
895 * @dev Returns the token decimals.
896 */
897 function decimals() public override view returns (uint8) {
898 return _decimals;
899 }
900
901 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

contracts/WhaleToken.sol

Locations

```
902 * @dev Returns the token symbol.
903 */
904 function symbol() public override view returns (string memory) {
905 return _symbol;
906 }
907
908 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

contracts/WhaleToken.sol

Locations

```
928 * - the caller must have a balance of at least `amount`.
929 */
930 function transfer(address recipient, uint256 amount) public override returns (bool) {
931 _transfer(_msgSender(), recipient, amount);
932 return true;
933 }
934
935 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

contracts/WhaleToken.sol

Locations

```
936 * @dev See {BEP20-allowance}.
937 */
938 function allowance(address owner, address spender) public override view returns (uint256) {
939 return _allowances[owner][spender];
940 }
941
942 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

contracts/WhaleToken.sol

Locations

```
947 * - `spender` cannot be the zero address.
948 */
949 function approve(address spender, uint256 amount) public override returns (bool) {
950 _approve(_msgSender(), spender, amount);
951 return true;
952 }
953
954 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

contracts/WhaleToken.sol

Locations

```
964 * `amount`.
965 */
966 function transferFrom(
967 address sender,
968 address recipient,
969 uint256 amount
970 ) public override returns (bool) {
971 _transfer(sender, recipient, amount);
972 _approve(
973 sender,
974 _msgSender(),
975 _allowances[sender][_msgSender()].sub(amount, "BEP20: transfer amount exceeds allowance")
976 );
977 return true;
978 }
979
980 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

contracts/WhaleToken.sol

Locations

```
990 * - `spender` cannot be the zero address.
991 */
992 function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
993 _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
994 return true;
995 }
996
997 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

contracts/WhaleToken.sol

Locations

```
1009 * `subtractedValue`.
1010 */
1011 function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
1012 _approve(
1013 _msgSender(),
1014 spender,
1015 _allowances[_msgSender()][spender].sub(subtractedValue, "BEP20: decreased allowance below zero")
1016 );
1017 return true;
1018 }
1019
1020 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

contracts/WhaleToken.sol

Locations

```
1026 * - `msg.sender` must be the token owner
1027 */
1028 function mint(uint256 amount) public onlyOwner returns (bool) {
1029 _mint(_msgSender(), amount);
1030 return true;
1031 }
1032
1033 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

contracts/WhaleToken.sol

Locations

```
1226
1227 /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
1228 function mint(address _to, uint256 _amount) public onlyOwner {
1229 _mint(_to, _amount);
1230 _moveDelegates(address(0), _delegates[_to], _amount);
1231 }
1232
1233 /// @dev overrides transfer function to meet tokenomics of WHALE
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "isExcludedFromAntiWhale" 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

contracts/WhaleToken.sol

Locations

```
1343 * @dev Returns the address is excluded from antiWhale or not.
1344 */
1345 function isExcludedFromAntiWhale(address _account) public view returns (bool) {
1346 return _excludedFromAntiWhale[_account];
1347 }
1348
1349 // To receive BNB from whaleRouter when swapping
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateTransferTaxRate" 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

contracts/WhaleToken.sol

Locations

```
1354 * Can only be called by the current operator.
1355 */
1356 function updateTransferTaxRate(uint16 _transferTaxRate) public onlyOperator {
1357 require(_transferTaxRate <= MAXIMUM_TRANSFER_TAX_RATE, "WHALE::updateTransferTaxRate: Transfer tax rate must not exceed the maximum rate.");
1358 emit TransferTaxRateUpdated(msg.sender, transferTaxRate, _transferTaxRate);
1359 transferTaxRate = _transferTaxRate;
1360 }
1361
1362 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateBurnRate" 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

contracts/WhaleToken.sol

Locations

```
1364 * Can only be called by the current operator.
1365 */
1366 function updateBurnRate(uint16 _burnRate) public onlyOperator {
1367 require(_burnRate <= 100, "WHALE::updateBurnRate: Burn rate must not exceed the maximum rate.");
1368 emit BurnRateUpdated(msg.sender, burnRate, _burnRate);
1369 burnRate = _burnRate;
1370 }
1371
1372 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateMaxTransferAmountRate" 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

contracts/WhaleToken.sol

Locations

```
1374 * Can only be called by the current operator.
1375 */
1376 function updateMaxTransferAmountRate(uint16 _maxTransferAmountRate) public onlyOperator {
1377 require(_maxTransferAmountRate <= 100000, "WHALE::updateMaxTransferAmountRate: Max transfer amount rate must not exceed the maximum rate.");
1378 emit MaxTransferAmountRateUpdated(msg.sender, maxTransferAmountRate, _maxTransferAmountRate);
1379 maxTransferAmountRate = _maxTransferAmountRate;
1380 }
1381
1382 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateMinAmountToLiquify" 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

contracts/WhaleToken.sol

Locations

```
1384 * Can only be called by the current operator.
1385 */
1386 function updateMinAmountToLiquify(uint256 _minAmount) public onlyOperator {
1387 emit MinAmountToLiquifyUpdated(msg.sender, minAmountToLiquify, _minAmount);
1388 minAmountToLiquify = _minAmount;
1389 }
1390
1391 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setExcludedFromAntiWhale" 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

contracts/WhaleToken.sol

Locations

```
1393 * Can only be called by the current operator.
1394 */
1395 function setExcludedFromAntiWhale(address _account, bool _excluded) public onlyOperator {
1396 _excludedFromAntiWhale[_account] = _excluded;
1397 }
1398
1399 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateSwapAndLiquifyEnabled" 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

contracts/WhaleToken.sol

Locations

```
1401 * Can only be called by the current operator.
1402 */
1403 function updateSwapAndLiquifyEnabled(bool _enabled) public onlyOperator {
1404 emit SwapAndLiquifyEnabledUpdated(msg.sender, _enabled);
1405 swapAndLiquifyEnabled = _enabled;
1406 }
1407
1408 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "UpdateSwapEnabled" 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

contracts/WhaleToken.sol

Locations

```
1409 * @dev Update the swapEnabled. Can only be called by the current Owner.
1410 */
1411 function UpdateSwapEnabled(bool _enabled) public onlyOwner {
1412 emit SwapEnabledUpdated(msg.sender, _enabled);
1413 swapEnabled = _enabled;
1414 }
1415
1416 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateWhaleRouter" 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

contracts/WhaleToken.sol

Locations

```
1418 * Can only be called by the current operator.
1419 */
1420 function updateWhaleRouter(address _router) public onlyOperator {
1421 whaleRouter = IUniswapV2Router02(_router);
1422 whalePair = IUniswapV2Factory(whaleRouter.factory()).getPair(address(this), whaleRouter.WETH());
1423 require(whalePair != address(0), "WHALE::updateWhaleRouter: Invalid pair address.");
1424 emit WhaleRouterUpdated(msg.sender, address(whaleRouter), whalePair);
1425 }
1426
1427 /**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "transferOperator" 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

contracts/WhaleToken.sol

Locations

```
1436 * Can only be called by the current operator.
1437 */
1438 function transferOperator(address newOperator) public onlyOperator {
1439 require(newOperator != address(0), "WHALE::transferOperator: new operator is the zero address");
1440 emit OperatorTransferred(_operator, newOperator);
1441 _operator = newOperator;
1442 }
1443
1444 // Copied and modified from YAM code:
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.0"". 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

contracts/WhaleToken.sol

Locations

```
7  // File: @uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol
8
9  pragma solidity ^0.5.0;
10
11 interface IUniswapV2Factory {
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.0"". 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

contracts/WhaleToken.sol

Locations

```
27 // File: @uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol
28
29 pragma solidity ^0.5.0;
30
31 interface IUniswapV2Pair {
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.2"". 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/WhaleToken.sol

Locations

```
82  // File: @uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol
83
84  pragma solidity >=0.6.2;
85
86  interface IUniswapV2Router01 {
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.2"". 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/WhaleToken.sol

Locations

```
180 // File: @uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol
181
182 pragma solidity >=0.6.2;
183
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.2<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

contracts/WhaleToken.sol

Locations

```
226 // File: @openzeppelin/contracts/utils/Address.sol
227
228 pragma solidity >=0.6.2 <0.8.0;
229
230 /**
```

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

contracts/WhaleToken.sol

Locations

```
416 // File: @openzeppelin/contracts/math/SafeMath.sol
417
418 pragma solidity >=0.6.0 <0.8.0;
419
420 /**
```

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

contracts/WhaleToken.sol

Locations

```
631 // File: contracts/libs/IBEP20.sol
632
633 pragma solidity >=0.4.0;
634
635 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

contracts/WhaleToken.sol

Locations

```
730 // File: @openzeppelin/contracts/utils/Context.sol
731
732 pragma solidity >=0.6.0 <0.8.0;
733
734 /*
```

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

contracts/WhaleToken.sol

Locations

```
755 // File: @openzeppelin/contracts/access/Ownable.sol
756
757 pragma solidity >=0.6.0 <0.8.0;
758
759 /**
```

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

contracts/WhaleToken.sol

Locations

```
823 // File: contracts/libs/BEP20.sol
824
825 pragma solidity >=0.4.0;
826
827 /**
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

contracts/WhaleToken.sol

Locations

```
1420 function updateWhaleRouter(address _router) public onlyOperator {
1421 whaleRouter = IUniswapV2Router02(_router);
1422 whalePair = IUniswapV2Factory(whaleRouter.factory()).getPair(address(this), whaleRouter.WETH());
1423 require(whalePair != address(0), "WHALE::updateWhaleRouter: Invalid pair address.");
1424 emit WhaleRouterUpdated(msg.sender, address(whaleRouter), whalePair);
```

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

contracts/WhaleToken.sol

Locations

```
1546 require(signatory != address(0), "WHALE::delegateBySig: invalid signature");
1547 require(nonce == nonces[signatory]++, "WHALE::delegateBySig: invalid nonce");
1548 require(now <= expiry, "WHALE::delegateBySig: signature expired");
1549 return _delegate(signatory, delegatee);
1550 }
```

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

contracts/WhaleToken.sol

Locations

```
1576 returns (uint256)
1577 {
1578 require(blockNumber < block.number, "WHALE::getPriorVotes: not yet determined");
1579
1580 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

contracts/WhaleToken.sol

Locations

```
1649 internal
1650 {
1651 uint32 blockNumber = safe32(block.number, "WHALE::_writeCheckpoint: block number exceeds 32 bits");
1652
1653 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

contracts/WhaleToken.sol

Locations

```
1576  returns (uint256)
1577  {
1578  require(blockNumber < block.number, "WHALE::getPriorVotes: not yet determined");
1579
1580  uint32 nCheckpoints = numCheckpoints[account];
```

LOW

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

SWC-123

Source file

contracts/WhaleToken.sol

Locations

```
1420 function updateWhaleRouter(address _router) public onlyOperator {
1421 whaleRouter = IUniswapV2Router02(_router);
1422 whalePair = IUniswapV2Factory(whaleRouter.factory()).getPair(address(this), whaleRouter.WETH());
1423 require(whalePair != address(0), "WHALE::updateWhaleRouter: Invalid pair address.");
1424 emit WhaleRouterUpdated(msg.sender, address(whaleRouter), whalePair);
```

Source file

contracts/WhaleToken.sol

Locations

```
1140
1141 // WhaleToken with Governance.
1142 contract WhaleToken is BEP20 {
1143 // Transfer tax rate in basis points. (default 8%)
1144 uint16 public transferTaxRate = 800;
1145 // Burn rate % of transfer tax. (default 50% x 8% = 4% of total amount).
1146 uint16 public burnRate = 50;
1147 // Max transfer tax rate: 10%.
1148 uint16 public constant MAXIMUM_TRANSFER_TAX_RATE = 1000;
1149 // Burn address
1150 address public constant BURN_ADDRESS = 0x000000000000000000000000000000000000dEaD;
1151
1152 // Max transfer amount rate in basis points. (default is 2.5% of total supply)
1153 uint16 public maxTransferAmountRate = 2500;
1154 // Addresses that excluded from antiWhale
1155 mapping(address => bool) private _excludedFromAntiWhale;
1156 // Automatic swap and liquify enabled
1157 bool public swapAndLiquifyEnabled = false;
1158 // Swap enabled when launch
1159 bool public swapEnabled = true;
1160 // Min amount to liquify. (default 1 WHALEs)
1161 uint256 public minAmountToLiquify = 1 ether;
1162 // The swap router, modifiable. Will be changed to Whale's router when our own AMM release
1163 IUniswapV2Router02 public whaleRouter;
1164 // The trading pair
1165 address public whalePair;
1166 // In swap and liquify
1167 bool private _inSwapAndLiquify;
1168
1169 // The operator can only update the transfer tax rate
1170 address private _operator;
1171
1172 // Events
1173 event OperatorTransferred(address indexed previousOperator, address indexed newOperator);
1174 event TransferTaxRateUpdated(address indexed operator, uint256 previousRate, uint256 newRate);
1175 event BurnRateUpdated(address indexed operator, uint256 previousRate, uint256 newRate);
1176 event MaxTransferAmountRateUpdated(address indexed operator, uint256 previousRate, uint256 newRate);
1177 event SwapAndLiquifyEnabledUpdated(address indexed operator, bool enabled);
1178 event SwapEnabledUpdated(address indexed owner, bool enabled);
1179 event MinAmountToLiquifyUpdated(address indexed operator, uint256 previousAmount, uint256 newAmount);
1180 event WhaleRouterUpdated(address indexed operator, address indexed router, address indexed pair);
1181 event SwapAndLiquify(uint256 tokensSwapped, uint256 ethReceived, uint256 tokensIntoLiqudity);
1182
1183 modifier onlyOperator() {
1184 require(_operator == msg.sender, "operator: caller is not the operator");
```

```
_;
}

modifier antiWhale(address sender, address recipient, uint256 amount) {
if (maxTransferAmount() > 0) {
if (
_excludedFromAntiWhale[sender] == false
&& _excludedFromAntiWhale[recipient] == false
) {
require(amount <= maxTransferAmount(), "WHALE::antiWhale: Transfer amount exceeds the maxTransferAmount");
require(swapEnabled == true, "WHALE::swap: Cannot transfer at the moment");
}
}
_;
}

modifier lockTheSwap {
_inSwapAndLiquify = true;
_;
_inSwapAndLiquify = false;
}

modifier transferTaxFree {
uint16 _transferTaxRate = transferTaxRate;
transferTaxRate = 0;
_;
transferTaxRate = _transferTaxRate;
}

/**
* @notice Constructs the WhaleToken contract.
*/
constructor() public BEP20("WhaleFarm Token", "WHALE") {
_operator = _msgSender();
emit OperatorTransferred(address(0), _operator);

_excludedFromAntiWhale[msg.sender] = true;
_excludedFromAntiWhale[address(0)] = true;
_excludedFromAntiWhale[address(this)] = true;
_excludedFromAntiWhale[BURN_ADDRESS] = true;
}

/// @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);
}

/// @dev overrides transfer function to meet tokenomics of WHALE
function _transfer(address sender, address recipient, uint256 amount) internal virtual override antiWhale(sender, recipient, amount) {
// swap and liquify
if (
swapAndLiquifyEnabled == true
&& _inSwapAndLiquify == false
&& address(whaleRouter) != address(0)
&& whalePair != address(0)
&& sender != whalePair
&& sender != owner()
) {
swapAndLiquify();
}

if (recipient == BURN_ADDRESS || transferTaxRate == 0) {
```

```
super._transfer(sender, recipient, amount);
} else {
// default tax is 8% of every transfer
uint256 taxAmount = amount.mul(transferTaxRate).div(10000);
uint256 burnAmount = taxAmount.mul(burnRate).div(100);
uint256 liquidityAmount = taxAmount.sub(burnAmount);
require(taxAmount == burnAmount + liquidityAmount, "WHALE::transfer: Burn value invalid");

// default 92% of transfer sent to recipient
uint256 sendAmount = amount.sub(taxAmount);
require(amount == sendAmount + taxAmount, "WHALE::transfer: Tax value invalid");

super._transfer(sender, BURN_ADDRESS, burnAmount);
super._transfer(sender, address(this), liquidityAmount);
super._transfer(sender, recipient, sendAmount);
amount = sendAmount;
}
}

/// @dev Swap and liquify
function swapAndLiquify() private lockTheSwap transferTaxFree {
uint256 contractTokenBalance = balanceOf(address(this));
uint256 maxTransferAmount = maxTransferAmount();
contractTokenBalance = contractTokenBalance > maxTransferAmount ? maxTransferAmount : contractTokenBalance;

if (contractTokenBalance >= minAmountToLiquify) {
// only min amount to liquify
uint256 liquifyAmount = minAmountToLiquify;

// split the liquify amount into halves
uint256 half = liquifyAmount.div(2);
uint256 otherHalf = liquifyAmount.sub(half);

// capture the contract's current ETH balance.
// this is so that we can capture exactly the amount of ETH that the
// swap creates, and not make the liquidity event include any ETH that
// has been manually sent to the contract
uint256 initialBalance = address(this).balance;

// swap tokens for ETH
swapTokensForEth(half);

// how much ETH did we just swap into?
uint256 newBalance = address(this).balance.sub(initialBalance);

// add liquidity
addLiquidity(otherHalf, newBalance);

emit SwapAndLiquify(half, newBalance, otherHalf);
}
}

/// @dev Swap tokens for eth
function swapTokensForEth(uint256 tokenAmount) private {
// generate the whale pair path of token -> weth
address[] memory path = new address[](2);
path[0] = address(this);
path[1] = whaleRouter.WETH();

_approve(address(this), address(whaleRouter), tokenAmount);

// make the swap
whaleRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
```

```
tokenAmount,
0, // accept any amount of ETH
path,
address(this),
block.timestamp
);
}

/// @dev Add liquidity
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
// approve token transfer to cover all possible scenarios
_approve(address(this), address(whaleRouter), tokenAmount);

// add the liquidity
whaleRouter.addLiquidityETH{value: ethAmount}(
address(this),
tokenAmount,
0, // slippage is unavoidable
0, // slippage is unavoidable
operator(),
block.timestamp
);
}

/**
* @dev Returns the max transfer amount.
*/
function maxTransferAmount() public view returns (uint256) {
return totalSupply().mul(maxTransferAmountRate).div(100000);
}

/**
* @dev Returns the address is excluded from antiWhale or not.
*/
function isExcludedFromAntiWhale(address _account) public view returns (bool) {
return _excludedFromAntiWhale[_account];
}

// To receive BNB from whaleRouter when swapping
receive() external payable {}

/**
* @dev Update the transfer tax rate.
* Can only be called by the current operator.
*/
function updateTransferTaxRate(uint16 _transferTaxRate) public onlyOperator {
require(_transferTaxRate <= MAXIMUM_TRANSFER_TAX_RATE, "WHALE::updateTransferTaxRate: Transfer tax rate must not exceed the maximum rate.");
emit TransferTaxRateUpdated(msg.sender, transferTaxRate, _transferTaxRate);
transferTaxRate = _transferTaxRate;
}

/**
* @dev Update the burn rate.
* Can only be called by the current operator.
*/
function updateBurnRate(uint16 _burnRate) public onlyOperator {
require(_burnRate <= 100, "WHALE::updateBurnRate: Burn rate must not exceed the maximum rate.");
emit BurnRateUpdated(msg.sender, burnRate, _burnRate);
burnRate = _burnRate;
}

/**
* @dev Update the max transfer amount rate.
```

```
1374  * Can only be called by the current operator.
1375  */
1376  function updateMaxTransferAmountRate(uint16 _maxTransferAmountRate) public onlyOperator {
1377  require(_maxTransferAmountRate <= 100000, "WHALE::updateMaxTransferAmountRate: Max transfer amount rate must not exceed the maximum rate.");
1378  emit MaxTransferAmountRateUpdated(msg.sender, maxTransferAmountRate, _maxTransferAmountRate);
1379  maxTransferAmountRate = _maxTransferAmountRate;
1380  }
1381
1382  /**
1383  * @dev Update the min amount to liquify.
1384  * Can only be called by the current operator.
1385  */
1386  function updateMinAmountToLiquify(uint256 _minAmount) public onlyOperator {
1387  emit MinAmountToLiquifyUpdated(msg.sender, minAmountToLiquify, _minAmount);
1388  minAmountToLiquify = _minAmount;
1389  }
1390
1391  /**
1392  * @dev Exclude or include an address from antiWhale.
1393  * Can only be called by the current operator.
1394  */
1395  function setExcludedFromAntiWhale(address _account, bool _excluded) public onlyOperator {
1396  _excludedFromAntiWhale[_account] = _excluded;
1397  }
1398
1399  /**
1400  * @dev Update the swapAndLiquifyEnabled.
1401  * Can only be called by the current operator.
1402  */
1403  function updateSwapAndLiquifyEnabled(bool _enabled) public onlyOperator {
1404  emit SwapAndLiquifyEnabledUpdated(msg.sender, _enabled);
1405  swapAndLiquifyEnabled = _enabled;
1406  }
1407
1408  /**
1409  * @dev Update the swapEnabled. Can only be called by the current Owner.
1410  */
1411  function UpdateSwapEnabled(bool _enabled) public onlyOwner {
1412  emit SwapEnabledUpdated(msg.sender, _enabled);
1413  swapEnabled = _enabled;
1414  }
1415
1416  /**
1417  * @dev Update the swap router.
1418  * Can only be called by the current operator.
1419  */
1420  function updateWhaleRouter(address _router) public onlyOperator {
1421  whaleRouter = IUniswapV2Router02(_router);
1422  whalePair = IUniswapV2Factory(whaleRouter.factory()).getPair(address(this), whaleRouter.WETH());
1423  require(whalePair != address(0), "WHALE::updateWhaleRouter: Invalid pair address.");
1424  emit WhaleRouterUpdated(msg.sender, address(whaleRouter), whalePair);
1425  }
1426
1427  /**
1428  * @dev Returns the address of the current operator.
1429  */
1430  function operator() public view returns (address) {
1431  return _operator;
1432  }
1433
1434  /**
1435  * @dev Transfers operator of the contract to a new account (`newOperator`).
1436  * Can only be called by the current operator.
```

```
*/
function transferOperator(address newOperator) public onlyOperator {
require(newOperator != address(0), "WHALE::transferOperator: new operator is the zero address");
emit OperatorTransferred(_operator, newOperator);
_operator = newOperator;
}

// Copied and modified from YAM code:
// https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernanceStorage.sol
// https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernance.sol
// Which is copied and modified from COMPOUND:
// https://github.com/compound-finance/compound-protocol/blob/master/contracts/Governance/Comp.sol

/// @dev A record of each accounts delegate
mapping (address => address) internal _delegates;

/// @notice A checkpoint for marking number of votes from a given block
struct Checkpoint {
uint32 fromBlock;
uint256 votes;
}

/// @notice A record of votes checkpoints for each account, by index
mapping (address => mapping (uint32 => Checkpoint)) public checkpoints;

/// @notice The number of checkpoints for each account
mapping (address => uint32) public numCheckpoints;

/// @notice The EIP-712 typehash for the contract's domain
bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");

/// @notice The EIP-712 typehash for the delegation struct used by the contract
bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");

/// @notice A record of states for signing / validating signatures
mapping (address => uint) public nonces;

/// @notice An event thats emitted when an account changes its delegate
event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

/// @notice An event thats emitted when a delegate account's vote balance changes
event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

/**
* @notice Delegate votes from `msg.sender` to `delegatee`
* @param delegator The address to get delegatee for
*/
function delegates(address delegator)
external
view
returns (address)
{
return _delegates[delegator];
}

/**
* @notice Delegate votes from `msg.sender` to `delegatee`
* @param delegatee The address to delegate votes to
*/
function delegate(address delegatee) external {
return _delegate(msg.sender, delegatee);
}
```

```
/**
* @notice Delegates votes from signatory to `delegatee`
* @param delegatee The address to delegate votes to
* @param nonce The contract state required to match the signature
* @param expiry The time at which to expire the signature
* @param v The recovery byte of the signature
* @param r Half of the ECDSA signature pair
* @param s Half of the ECDSA signature pair
*/
function delegateBySig(
address delegatee,
uint nonce,
uint expiry,
uint8 v,
bytes32 r,
bytes32 s
)
external
{
bytes32 domainSeparator = keccak256(
abi.encode(
DOMAIN_TYPEHASH,
keccak256(bytes(name())),
getChainId(),
address(this)
)
);

bytes32 structHash = keccak256(
abi.encode(
DELEGATION_TYPEHASH,
delegatee,
nonce,
expiry
)
);

bytes32 digest = keccak256(
abi.encodePacked(
"\x19\x01",
domainSeparator,
structHash
)
);

address signatory = ecrecover(digest, v, r, s);
require(signatory != address(0), "WHALE::delegateBySig: invalid signature");
require(nonce == nonces[signatory]++, "WHALE::delegateBySig: invalid nonce");
require(now <= expiry, "WHALE::delegateBySig: signature expired");
return _delegate(signatory, delegatee);
}

/**
* @notice Gets the current votes balance for `account`
* @param account The address to get votes balance
* @return The number of current votes for `account`
*/
function getCurrentVotes(address account)
external
view
returns (uint256)
{
uint32 nCheckpoints = numCheckpoints[account];
```

```
return nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0;
}

/**
* @notice Determine the prior number of votes for an account as of a block number
* @dev Block number must be a finalized block or else this function will revert to prevent misinformation.
* @param account The address of the account to check
* @param blockNumber The block number to get the vote balance at
* @return The number of votes the account had as of the given block
*/
function getPriorVotes(address account, uint blockNumber)
external
view
returns (uint256)
{
require(blockNumber < block.number, "WHALE::getPriorVotes: not yet determined");

uint32 nCheckpoints = numCheckpoints[account];
if (nCheckpoints == 0) {
return 0;
}

// First check most recent balance
if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
return checkpoints[account][nCheckpoints - 1].votes;
}

// Next check implicit zero balance
if (checkpoints[account][0].fromBlock > blockNumber) {
return 0;
}

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];
if (cp.fromBlock == blockNumber) {
return cp.votes;
} else if (cp.fromBlock < blockNumber) {
lower = center;
} else {
upper = center - 1;
}
}
return checkpoints[account][lower].votes;
}

function _delegate(address delegator, address delegatee)
internal
{
address currentDelegate = _delegates[delegator];
uint256 delegatorBalance = balanceOf(delegator); // balance of underlying WHALEs (not scaled);
_delegates[delegator] = delegatee;

emit DelegateChanged(delegator, currentDelegate, delegatee);

_moveDelegates(currentDelegate, delegatee, delegatorBalance);
}

function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal {
if (srcRep != dstRep && amount > 0) {
if (srcRep != address(0)) {
```

```
// decrease old representative
uint32 srcRepNum = numCheckpoints[srcRep];
uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
uint256 srcRepNew = srcRepOld.sub(amount);
_writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
}

if (dstRep != address(0)) {
// increase new representative
uint32 dstRepNum = numCheckpoints[dstRep];
uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
uint256 dstRepNew = dstRepOld.add(amount);
_writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
}
}
}

function _writeCheckpoint(
address delegatee,
uint32 nCheckpoints,
uint256 oldVotes,
uint256 newVotes
)
internal
{
uint32 blockNumber = safe32(block.number, "WHALE::_writeCheckpoint: block number exceeds 32 bits");

if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
} else {
checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
numCheckpoints[delegatee] = nCheckpoints + 1;
}

emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
}

function safe32(uint n, string memory errorMessage) internal pure returns (uint32) {
require(n < 2**32, errorMessage);
return uint32(n);
}

function getChainId() internal pure returns (uint) {
uint256 chainId;
assembly { chainId := chainid() }
return chainId;
}
}

```