MythX

REPORT 60D5938B3A286200190F04B5

Created	Fri Jun 25 2021 08:27:55 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	60d58d6043f2c39d6f12de1d

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
51ffdc6e-73fc-48c4-b8fb-f574e6a02cfc	contracts/FoxToken.sol	33

Analysis `51ffdc6e-73fc-48c4-b8fb-f574e6a02cfc` MythX

Started	Fri Jun 25 2021 08:27:59 GMT+0000 (Coordinated Universal Time)
Finished	Fri Jun 25 2021 09:14:58 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Remythx
Main Source File	`Contracts/FoxToken.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	24	9

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/FoxToken.sol`

Locations

```
252 * thereby removing any functionality that is only available to the owner.
253 */
254 function renounceOwnership() public virtual onlyOwner {
255 emit OwnershipTransferred(_owner, address(0));
256 _owner = address(0);
257 }
258
259 /**
```

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/FoxToken.sol

Locations

```
261 * Can only be called by the current owner.
262 */
263 function transferOwnership(address newOwner) public virtual onlyOwner {
264 require(newOwner != address(0), "Ownable: new owner is the zero address");
265 emit OwnershipTransferred(_owner, newOwner);
266 _owner = newOwner;
267 }
268 }
```

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/FoxToken.sol

Locations

```
387 }
388
389 function symbol() public override view returns (string memory) {
390 return _symbol;
391 }
392
393 function decimals() public override view returns (uint8) {
```

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/FoxToken.sol

Locations

```
391 }
392
393 function decimals() public override view returns (uint8) {
394 return _decimals;
395 }
396
397 function totalSupply() public override view returns (uint256) {
```

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/FoxToken.sol

Locations

```
403 }
404
405 function transfer(address recipient, uint256 amount) public override returns (bool) {
406 _transfer(_msgSender(), recipient, amount);
407 return true;
408 }
409
410 function allowance(address owner, address spender) public override view returns (uint256) {
```

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/FoxToken.sol

Locations

```
408 }
409
410 function allowance(address owner, address spender) public override view returns (uint256) {
411 return _allowances[owner][spender];
412 }
413
414 function approve(address spender, uint256 amount) public override returns (bool) {
```

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/FoxToken.sol

Locations

```
412 }
413
414 function approve(address spender, uint256 amount) public override returns (bool) {
415 _approve(_msgSender(), spender, amount);
416 return true;
417 }
418
419 function transferFrom (address sender, address recipient, uint256 amount) public override returns (bool) {
```

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

contracts/FoxToken.sol

Locations

```
417 }
418
419 function transferFrom (address sender, address recipient, uint256 amount) public override returns (bool) {
420 _transfer(sender, recipient, amount);
421 _approve(
422 sender,
423 _msgSender(),
424 _allowances[sender][_msgSender()].sub(amount, 'BEP20: transfer amount exceeds allowance')
425 );
426 return true;
427 }
428
429 function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
```

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

contracts/FoxToken.sol

Locations

```
427 }
428
429 function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
430 _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
431 return true;
432 }
433
434 function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
```

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

contracts/FoxToken.sol

Locations

```
432 }
433
434 function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
435 _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, 'BEP20: decreased allowance below zero'));
436 return true;
437 }
438
439 function mint(uint256 amount) public onlyOwner returns (bool) {
```

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/FoxToken.sol

Locations

```
437 }
438
439 function mint(uint256 amount) public onlyOwner returns (bool) {
440 _mint(_msgSender(), amount);
441 return true;
442 }
443
444 function _transfer (address sender, address recipient, uint256 amount) internal virtual {
```

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/FoxToken.sol

Locations

```
567
568 /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
569 function mint(address _to, uint256 _amount) public onlyOwner {
570 _mint(_to, _amount);
571 _moveDelegates(address(0), _delegates[_to], _amount);
572 }
573
574 /// @dev overrides transfer function to meet tokenomics
```

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/FoxToken.sol

Locations

```
684 * @dev Returns the address is excluded from antiWhale or not.
685 */
686 function isExcludedFromAntiWhale(address _account) public view returns (bool) {
687 return _excludedFromAntiWhale[_account];
688 }
689
690 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "isExcludedFromTransferTax" 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/FoxToken.sol

Locations

```
691 * @dev Returns the address is excluded from transferTax or not.
692 */
693 function isExcludedFromTransferTax(address _account) public view returns (bool) {
694 return _excludedFromTransferTax[_account];
695 }
696
697 // To receive BNB from swapRouter 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/FoxToken.sol

Locations

```
702 * Can only be called by the current operator.
703 */
704 function updateTransferTaxRate(uint16 _transferTaxRate) public onlyOperator {
705 require(_transferTaxRate <= MAXIMUM_TRANSFER_TAX_RATE, "ERROR::updateTransferTaxRate: Transfer tax rate must not exceed the maximum rate.");
706 emit TransferTaxRateUpdated(msg.sender, transferTaxRate, _transferTaxRate);
707 transferTaxRate = _transferTaxRate;
708 }
709
710 /**
```

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/FoxToken.sol

Locations

```
712 * Can only be called by the current operator.
713 */
714 function updateBurnRate(uint16 _burnRate) public onlyOperator {
715 require(_burnRate <= 100, "ERROR::updateBurnRate: Burn rate must not exceed the maximum rate.");
716 emit BurnRateUpdated(msg.sender, burnRate, _burnRate);
717 burnRate = _burnRate;
718 }
719
720 /**
```

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/FoxToken.sol

Locations

```
722 * Can only be called by the current operator.
723 */
724 function updateMaxTransferAmountRate(uint16 _maxTransferAmountRate) public onlyOperator {
725 require(_maxTransferAmountRate <= 10000, "ERROR::updateMaxTransferAmountRate: Max transfer amount rate must not exceed the maximum rate.");
726 emit MaxTransferAmountRateUpdated(msg.sender, maxTransferAmountRate, _maxTransferAmountRate);
727 maxTransferAmountRate = _maxTransferAmountRate;
728 }
729
730 /**
```

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/FoxToken.sol

Locations

```
732 * Can only be called by the current operator.
733 */
734 function updateMinAmountToLiquify(uint256 _minAmount) public onlyOperator {
735 emit MinAmountToLiquifyUpdated(msg.sender, minAmountToLiquify, _minAmount);
736 minAmountToLiquify = _minAmount;
737 }
738
739 /**
```

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/FoxToken.sol

Locations

```
741 * Can only be called by the current operator.
742 */
743 function setExcludedFromAntiWhale(address _account, bool _excluded) public onlyOperator {
744 _excludedFromAntiWhale[_account] = _excluded;
745 }
746
747 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setExcludedFromTransferTax" 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/FoxToken.sol

Locations

```
749 * Can only be called by the current operator.
750 */
751 function setExcludedFromTransferTax(address _account, bool _excluded) public onlyOperator {
752 _excludedFromTransferTax[_account] = _excluded;
753 }
754
755 /**
```

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/FoxToken.sol

Locations

```
757 * Can only be called by the current operator.
758 */
759 function updateSwapAndLiquifyEnabled(bool _enabled) public onlyOperator {
760 emit SwapAndLiquifyEnabledUpdated(msg.sender, _enabled);
761 swapAndLiquifyEnabled = _enabled;
762 }
763
764 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateSwapRouter" 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/FoxToken.sol

Locations

```
766 * Can only be called by the current operator.
767 */
768 function updateSwapRouter(address _router) public onlyOperator {
769 swapRouter = IUniswapV2Router02(_router);
770
771 // create pair if not exist
772 if(swapPair == address(0)){
773 swapPair = IUniswapV2Factory(swapRouter.factory()).createPair(address(this), swapRouter.WETH());
774 } else {
775 swapPair = IUniswapV2Factory(swapRouter.factory()).getPair(address(this), swapRouter.WETH());
776 }
777 require(swapPair != address(0), "ERROR::updateSwapRouter: Invalid pair address.");
778 emit SwapRouterUpdated(msg.sender, address(swapRouter), swapPair);
779 }
780
781 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "createSwapPair" 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/FoxToken.sol

Locations

```
783 * Can only be called by the current operator.
784 */
785 function createSwapPair(address _router, address _pairAddress) public onlyOperator {
786 IUniswapV2Router02 router = IUniswapV2Router02(_router);
787 address pair = IUniswapV2Factory(router.factory()).getPair(address(this), _pairAddress);
788 require(pair == address(0), "ERROR::createSwapPair: Pair address already created.");
789
790 address newPair = IUniswapV2Factory(router.factory()).createPair(address(this), _pairAddress);
791 require(newPair != address(0), "ERROR::createSwapPair: Invalid pair address.");
792 emit SwapRouterUpdated(msg.sender, address(router), newPair);
793 }
794
795 /**
```

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/FoxToken.sol

Locations

```
804 * Can only be called by the current operator.
805 */
806 function transferOperator(address newOperator) public onlyOperator {
807 require(newOperator != address(0), "ERROR::transferOperator: new operator is the zero address");
808 emit OperatorTransferred(_operator, newOperator);
809 _operator = newOperator;
810 }
811 // Copied and modified from YAM code:
812 // https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernanceStorage.sol
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

contracts/FoxToken.sol

Locations

```
771 // create pair if not exist
772 if(swapPair == address(0)){
773 swapPair = IUniswapV2Factory(swapRouter.factory()).createPair(address(this), swapRouter.WETH());
774 } else {
775 swapPair = IUniswapV2Factory(swapRouter.factory()).getPair(address(this), swapRouter.WETH());
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

contracts/FoxToken.sol

Locations

```
776 }
777 require(swapPair != address(0), "ERROR::updateSwapRouter: Invalid pair address.");
778 emit SwapRouterUpdated(msg.sender, address(swapRouter), swapPair);
779 }
```

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/FoxToken.sol

Locations

```
771 // create pair if not exist
772 if(swapPair == address(0)){
773 swapPair = IUniswapV2Factory(swapRouter.factory()).createPair(address(this), swapRouter.WETH());
774 } else {
775 swapPair = IUniswapV2Factory(swapRouter.factory()).getPair(address(this), swapRouter.WETH());
```

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/FoxToken.sol

Locations

```
785 function createSwapPair(address _router, address _pairAddress) public onlyOperator {
786 IUniswapV2Router02 router = IUniswapV2Router02(_router);
787 address pair = IUniswapV2Factory(router.factory()).getPair(address(this), _pairAddress);
788 require(pair == address(0), "ERROR::createSwapPair: Pair address already created.");
```

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/FoxToken.sol

Locations

```
913 require(signatory != address(0), "Error::delegateBySig: invalid signature");
914 require(nonce == nonces[signatory]++, "Error::delegateBySig: invalid nonce");
915 require(now <= expiry, "Error::delegateBySig: signature expired");
916 return _delegate(signatory, delegatee);
917 }
```

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/FoxToken.sol

Locations

```
943 returns (uint256)
944 {
945 require(blockNumber < block.number, "Error::getPriorVotes: not yet determined");
946
947 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/FoxToken.sol

Locations

```
1016 internal
1017 {
1018 uint32 blockNumber = safe32(block.number, "Error::_writeCheckpoint: block number exceeds 32 bits");
1019
1020 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/FoxToken.sol

Locations

```
943 returns (uint256)
944 {
945 require(blockNumber < block.number, "Error::getPriorVotes: not yet determined");
946
947 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/FoxToken.sol

Locations

```
785 function createSwapPair(address _router, address _pairAddress) public onlyOperator {
786 IUniswapV2Router02 router = IUniswapV2Router02(_router);
787 address pair = IUniswapV2Factory(router.factory()).getPair(address(this), _pairAddress);
788 require(pair == address(0), "ERROR::createSwapPair: Pair address already created.");
```

Source file

contracts/FoxToken.sol

Locations

```
482
483 // FOX Token with Governance.
484 contract FOXToken is BEP20 {
485 // Transfer tax rate in basis points. (default 10%), set to 0 for Presale
486 uint16 public transferTaxRate = 0; //1000
487 // Burn rate % of transfer tax. (default 50% x 10% = 5% of total amount).
488 uint16 public burnRate = 50;
489 // Max transfer tax rate: 10%.
490 uint16 public constant MAXIMUM_TRANSFER_TAX_RATE = 1000;
491 // Burn address
492 address public constant BURN_ADDRESS = 0x000000000000000000000000000000000000dEaD;
493
494 // Max transfer amount rate in basis points. (default is 0.5% of total supply), set to 100 for Presale
495 uint16 public maxTransferAmountRate = 10000; //50
496 // Addresses that excluded from antiWhale
497 mapping(address => bool) private _excludedFromAntiWhale;
498 // Addresses that excluded from transferTax
499 mapping(address => bool) private _excludedFromTransferTax;
500 // Automatic swap and liquify enabled
501 bool public swapAndLiquifyEnabled = false;
502 // Min amount to liquify. (default 8888)
503 uint256 public minAmountToLiquify = 8888 ether;
504 // The swap router, modifiable. Will be changed to other's router when our own AMM release
505 IUniswapV2Router02 public swapRouter;
506 // The trading pair
507 address public swapPair;
508 // In swap and liquify
509 bool private _inSwapAndLiquify;
510
511 // The operator can only update the transfer tax rate
512 address private _operator;
513
514 // Events
515 event OperatorTransferred(address indexed previousOperator, address indexed newOperator);
516 event TransferTaxRateUpdated(address indexed operator, uint256 previousRate, uint256 newRate);
517 event BurnRateUpdated(address indexed operator, uint256 previousRate, uint256 newRate);
518 event MaxTransferAmountRateUpdated(address indexed operator, uint256 previousRate, uint256 newRate);
519 event SwapAndLiquifyEnabledUpdated(address indexed operator, bool enabled);
520 event MinAmountToLiquifyUpdated(address indexed operator, uint256 previousAmount, uint256 newAmount);
521 event SwapRouterUpdated(address indexed operator, address indexed router, address indexed pair);
522 event SwapAndLiquify(uint256 tokensSwapped, uint256 ethReceived, uint256 tokensIntoLiqudity);
523
524 modifier onlyOperator() {
525 require(_operator == msg.sender, "operator: caller is not the operator");
526 _;
527 }
```

```
528
529 modifier antiWhale(address sender, address recipient, uint256 amount) {
530 if (maxTransferAmount() > 0) {
531 if (
532 _excludedFromAntiWhale[sender] == false
533 && _excludedFromAntiWhale[recipient] == false
534 ) {
535 require(amount <= maxTransferAmount(), "AntiWhale: Transfer amount exceeds the maxTransferAmount");
536 }
537 }
538 _;
539 }
540
541 modifier lockTheSwap {
542 _inSwapAndLiquify = true;
543 _;
544 _inSwapAndLiquify = false;
545 }
546
547 modifier transferTaxFree {
548 uint16 _transferTaxRate = transferTaxRate;
549 transferTaxRate = 0;
550 _;
551 transferTaxRate = _transferTaxRate;
552 }
553
554 /**
555 * @notice Constructs the of the contract.
556 */
557 constructor() public BEP20("Fox Token", "FOX") {
558 _operator = _msgSender();
559 emit OperatorTransferred(address(0), _operator);
560
561 _excludedFromAntiWhale[msg.sender] = true;
562 _excludedFromAntiWhale[address(0)] = true;
563 _excludedFromAntiWhale[address(this)] = true;
564 _excludedFromAntiWhale[BURN_ADDRESS] = true;
565 _excludedFromTransferTax[msg.sender] = true;
566 }
567
568 /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
569 function mint(address _to, uint256 _amount) public onlyOwner {
570 _mint(_to, _amount);
571 _moveDelegates(address(0), _delegates[_to], _amount);
572 }
573
574 /// @dev overrides transfer function to meet tokenomics
575 function _transfer(address sender, address recipient, uint256 amount) internal virtual override antiWhale(sender, recipient, amount) {
576 // swap and liquify
577 if (
578 swapAndLiquifyEnabled == true
579 && _inSwapAndLiquify == false
580 && address(swapRouter) != address(0)
581 && swapPair != address(0)
582 && sender != swapPair
583 && sender != owner()
584 ) {
585 swapAndLiquify();
586 }
587
588 if (recipient == BURN_ADDRESS || transferTaxRate == 0 || _excludedFromTransferTax[sender] == true) {
589 super._transfer(sender, recipient, amount);
590 } else {
```

```
// default tax is 10% 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, "ERROR::transfer: Burn value invalid");

// default 90% of transfer sent to recipient
uint256 sendAmount = amount.sub(taxAmount);
require(amount == sendAmount + taxAmount, "ERROR::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 swap pair path of token -> weth
address[] memory path = new address[](2);
path[0] = address(this);
path[1] = swapRouter.WETH();

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

// make the swap
swapRouter.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(swapRouter), tokenAmount);

// add the liquidity
swapRouter.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(10000);
}

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

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

// To receive BNB from swapRouter 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, "ERROR::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, "ERROR::updateBurnRate: Burn rate must not exceed the maximum rate.");
emit BurnRateUpdated(msg.sender, burnRate, _burnRate);
```

```
burnRate = _burnRate;
}

/**
* @dev Update the max transfer amount rate.
* Can only be called by the current operator.
*/
function updateMaxTransferAmountRate(uint16 _maxTransferAmountRate) public onlyOperator {
require(_maxTransferAmountRate <= 10000, "ERROR::updateMaxTransferAmountRate: Max transfer amount rate must not exceed the maximum rate.");
emit MaxTransferAmountRateUpdated(msg.sender, maxTransferAmountRate, _maxTransferAmountRate);
maxTransferAmountRate = _maxTransferAmountRate;
}

/**
* @dev Update the min amount to liquify.
* Can only be called by the current operator.
*/
function updateMinAmountToLiquify(uint256 _minAmount) public onlyOperator {
emit MinAmountToLiquifyUpdated(msg.sender, minAmountToLiquify, _minAmount);
minAmountToLiquify = _minAmount;
}

/**
* @dev Exclude or include an address from antiWhale.
* Can only be called by the current operator.
*/
function setExcludedFromAntiWhale(address _account, bool _excluded) public onlyOperator {
_excludedFromAntiWhale[_account] = _excluded;
}

/**
* @dev Exclude or include an address from transferTax.
* Can only be called by the current operator.
*/
function setExcludedFromTransferTax(address _account, bool _excluded) public onlyOperator {
_excludedFromTransferTax[_account] = _excluded;
}

/**
* @dev Update the swapAndLiquifyEnabled.
* Can only be called by the current operator.
*/
function updateSwapAndLiquifyEnabled(bool _enabled) public onlyOperator {
emit SwapAndLiquifyEnabledUpdated(msg.sender, _enabled);
swapAndLiquifyEnabled = _enabled;
}

/**
* @dev Update the swap router.
* Can only be called by the current operator.
*/
function updateSwapRouter(address _router) public onlyOperator {
swapRouter = IUniswapV2Router02(_router);

// create pair if not exist
if(swapPair == address(0)){
swapPair = IUniswapV2Factory(swapRouter.factory()).createPair(address(this), swapRouter.WETH());
} else {
swapPair = IUniswapV2Factory(swapRouter.factory()).getPair(address(this), swapRouter.WETH());
}
require(swapPair != address(0), "ERROR::updateSwapRouter: Invalid pair address.");
emit SwapRouterUpdated(msg.sender, address(swapRouter), swapPair);
}
```

```
780
781 /**
782 * @dev Create LP pair.
783 * Can only be called by the current operator.
784 */
785 function createSwapPair(address _router, address _pairAddress) public onlyOperator {
786 IUniswapV2Router02 router = IUniswapV2Router02(_router);
787 address pair = IUniswapV2Factory(router.factory()).getPair(address(this), _pairAddress);
788 require(pair == address(0), "ERROR::createSwapPair: Pair address already created.");
789
790 address newPair = IUniswapV2Factory(router.factory()).createPair(address(this), _pairAddress);
791 require(newPair != address(0), "ERROR::createSwapPair: Invalid pair address.");
792 emit SwapRouterUpdated(msg.sender, address(router), newPair);
793 }
794
795 /**
796 * @dev Returns the address of the current operator.
797 */
798 function operator() public view returns (address) {
799 return _operator;
800 }
801
802 /**
803 * @dev Transfers operator of the contract to a new account (`newOperator`).
804 * Can only be called by the current operator.
805 */
806 function transferOperator(address newOperator) public onlyOperator {
807 require(newOperator != address(0), "ERROR::transferOperator: new operator is the zero address");
808 emit OperatorTransferred(_operator, newOperator);
809 _operator = newOperator;
810 }
811 // Copied and modified from YAM code:
812 // https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernanceStorage.sol
813 // https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernance.sol
814 // Which is copied and modified from COMPOUND:
815 // https://github.com/compound-finance/compound-protocol/blob/master/contracts/Governance/Comp.sol
816
817 /// @dev A record of each accounts delegate
818 mapping (address => address) internal _delegates;
819
820 /// @notice A checkpoint for marking number of votes from a given block
821 struct Checkpoint {
822 uint32 fromBlock;
823 uint256 votes;
824 }
825
826 /// @notice A record of votes checkpoints for each account, by index
827 mapping (address => mapping (uint32 => Checkpoint)) public checkpoints;
828
829 /// @notice The number of checkpoints for each account
830 mapping (address => uint32) public numCheckpoints;
831
832 /// @notice The EIP-712 typehash for the contract's domain
833 bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");
834
835 /// @notice The EIP-712 typehash for the delegation struct used by the contract
836 bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");
837
838 /// @notice A record of states for signing / validating signatures
839 mapping (address => uint) public nonces;
840
841 /// @notice An event thats emitted when an account changes its delegate
842 event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);
```

```
843
844 /// @notice An event thats emitted when a delegate account's vote balance changes
845 event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);
846
847 /**
848 * @notice Delegate votes from `msg.sender` to `delegatee`
849 * @param delegator The address to get delegatee for
850 */
851 function delegates(address delegator)
852 external
853 view
854 returns (address)
855 {
856 return _delegates[delegator];
857 }
858
859 /**
860 * @notice Delegate votes from `msg.sender` to `delegatee`
861 * @param delegatee The address to delegate votes to
862 */
863 function delegate(address delegatee) external {
864 return _delegate(msg.sender, delegatee);
865 }
866
867 /**
868 * @notice Delegates votes from signatory to `delegatee`
869 * @param delegatee The address to delegate votes to
870 * @param nonce The contract state required to match the signature
871 * @param expiry The time at which to expire the signature
872 * @param v The recovery byte of the signature
873 * @param r Half of the ECDSA signature pair
874 * @param s Half of the ECDSA signature pair
875 */
876 function delegateBySig(
877 address delegatee,
878 uint nonce,
879 uint expiry,
880 uint8 v,
881 bytes32 r,
882 bytes32 s
883 )
884 external
885 {
886 bytes32 domainSeparator = keccak256(
887 abi.encode(
888 DOMAIN_TYPEHASH,
889 keccak256(bytes(name())),
890 getChainId(),
891 address(this)
892 )
893 );
894
895 bytes32 structHash = keccak256(
896 abi.encode(
897 DELEGATION_TYPEHASH,
898 delegatee,
899 nonce,
900 expiry
901 )
902 );
903
904 bytes32 digest = keccak256(
905 abi.encodePacked(
```

```
"\x19\x01",
domainSeparator,
structHash
)
));

address signatory = ecrecover(digest, v, r, s);
require(signatory != address(0), "Error::delegateBySig: invalid signature");
require(nonce == nonces[signatory]++, "Error::delegateBySig: invalid nonce");
require(now <= expiry, "Error::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, "Error::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 token (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, "Error::_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);
```

```
1032    return uint32(n);
1033    }
1034
1035    function getChainId() internal pure returns (uint) {
1036    uint256 chainId;
1037    assembly { chainId := chainid() }
1038    return chainId;
1039    }
1040    }
```