Compound Farm

Smart Contract Audit

Compound Farm Audit Summary

Project name : Compound Farm Contract

Project address: https://cmp.farm

Code URL : https://github.com/primes-dev/cmp-contracts.git

Commit : 8e89020c7d61fef9761adb6d8d83bcc5f147d3f9

Project target : Compound Farm Contract Audit

Blockchain : Huobi ECO Chain（Heco）

Test result : PASSED

Audit Info

Audit NO : 0X202105130018

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

Compound Farm Audit

The Compound Farm team asked us to review and audit their Compound Farm contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
Compound Farm Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-05-13

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Compound Farm contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

(Statement: Armors Labs reports only on facts that have occurred or existed before this report is issued and assumes corresponding responsibilities. Armors Labs is not able to determine the security of its smart contracts and is not responsible for any subsequent or existing facts after this report is issued. The security audit analysis and other content of this report are only based on the documents and information provided by the information provider to Armors Labs at the time of issuance of this report (" information provided " for short). Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered,

deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused.)

Audited target file

file	md5
CMPToken.sol	cb68378270567c0ebc1ddc2d1510c40b
Farm.sol	7e1d503ac55b62d78b61dda594fd1f65
StratX_MDEX.sol	5600104dff55fddfbf41d9b2d430e188

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe

Vulnerability	status
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

CMPToken.sol

```
pragma solidity ^0.7.0;


// SPDX-License-Identifier: MIT
/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
}

//
/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }
```

```
    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

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

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

//
/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
 * types.
 *
 * Sets have the following properties:
 *
 * - Elements are added, removed, and checked for existence in constant time
 * (O(1)).
 * - Elements are enumerated in O(n). No guarantees are made on the ordering.
 *
 *
 * contract Example {
 *     // Add the library methods
 *     using EnumerableSet for EnumerableSet.AddressSet;
 *
 *     // Declare a set state variable
 *     EnumerableSet.AddressSet private mySet;
 * }
 *
 *
 * As of v3.3.0, sets of type `bytes32` (`Bytes32Set`), `address` (`AddressSet`)
 * and `uint256` (`UintSet`) are supported.
 */
library EnumerableSet {
    // To implement this library for multiple types with as little code
```

```
    // repetition as possible, we write it in terms of a generic Set type with
    // bytes32 values.
    // The Set implementation uses private functions, and user-facing
    // implementations (such as AddressSet) are just wrappers around the
    // underlying Set.
    // This means that we can only create new EnumerableSets for types that fit
    // in bytes32.

    struct Set {
        // Storage of set values
        bytes32[] _values;

        // Position of the value in the `values` array, plus 1 because index 0
        // means a value is not in the set.
        mapping (bytes32 => uint256) _indexes;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function _add(Set storage set, bytes32 value) private returns (bool) {
        if (!_contains(set, value)) {
            set._values.push(value);
            // The value is stored at length-1, but we add 1 to all indexes
            // and use 0 as a sentinel value
            set._indexes[value] = set._values.length;
            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function _remove(Set storage set, bytes32 value) private returns (bool) {
        // We read and store the value's index to prevent multiple reads from the same storage slot
        uint256 valueIndex = set._indexes[value];

        if (valueIndex != 0) { // Equivalent to contains(set, value)
            // To delete an element from the _values array in O(1), we swap the element to delete wit
            // the array, and then remove the last element (sometimes called as 'swap and pop').
            // This modifies the order of the array, as noted in {at}.

            uint256 toDeleteIndex = valueIndex - 1;
            uint256 lastIndex = set._values.length - 1;

            // When the value to delete is the last one, the swap operation is unnecessary. However,
            // so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' stateme

            bytes32 lastvalue = set._values[lastIndex];

            // Move the last value to the index where the value to delete is
            set._values[toDeleteIndex] = lastvalue;
            // Update the index for the moved value
            set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

            // Delete the slot where the moved value was stored
            set._values.pop();
```

```
            // Delete the index for the deleted slot
            delete set._indexes[value];

            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function _contains(Set storage set, bytes32 value) private view returns (bool) {
        return set._indexes[value] != 0;
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function _length(Set storage set) private view returns (uint256) {
        return set._values.length;
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function _at(Set storage set, uint256 index) private view returns (bytes32) {
        require(set._values.length > index, "EnumerableSet: index out of bounds");
        return set._values[index];
    }

    // Bytes32Set

    struct Bytes32Set {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _add(set._inner, value);
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _remove(set._inner, value);
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
```

```
     */
    function contains(Bytes32Set storage set, bytes32 value) internal view returns (bool) {
        return _contains(set._inner, value);
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(Bytes32Set storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(Bytes32Set storage set, uint256 index) internal view returns (bytes32) {
        return _at(set._inner, index);
    }

    // AddressSet

    struct AddressSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(AddressSet storage set, address value) internal returns (bool) {
        return _add(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(AddressSet storage set, address value) internal returns (bool) {
        return _remove(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(AddressSet storage set, address value) internal view returns (bool) {
        return _contains(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(AddressSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }
```

```
    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(AddressSet storage set, uint256 index) internal view returns (address) {
        return address(uint160(uint256(_at(set._inner, index))));
    }


    // UintSet

    struct UintSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(UintSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(UintSet storage set, uint256 value) internal view returns (bool) {
        return _contains(set._inner, bytes32(value));
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
```

```
        return uint256(_at(set._inner, index));
    }
}

//
/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
```

```
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
```

```
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
```

```
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}

//
/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);
```

```
    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

//
/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /**
```

```
     * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
     * a default value of 18.
     *
     * To select a different value for {decimals}, use {_setupDecimals}.
     *
     * All three of these values are immutable: they can only be set once during
     * construction.
     */
    constructor (string memory name_, string memory symbol_) public {
        _name = name_;
        _symbol = symbol_;
        _decimals = 18;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view virtual returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view virtual returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
     * called.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view virtual returns (uint8) {
        return _decimals;
    }

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view virtual override returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view virtual override returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
```

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

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override returns (uint256)
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * Requirements:
     *
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
```

```
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC2
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements:
```

```
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal virtual {
        _decimals = decimals_;
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks]
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
}

//
abstract contract DelegateERC20 is ERC20 {
    using SafeMath for uint256;
```

```
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,add

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

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


// support delegates mint
function _mint(address account, uint256 amount) internal override virtual {
    super._mint(account, amount);

    // add delegates to the minter
    _moveDelegates(address(0), _delegates[account], amount);
}


function _transfer(address sender, address recipient, uint256 amount) internal override virtual {
    super._transfer(sender, recipient, amount);
    _moveDelegates(_delegates[sender], _delegates[recipient], amount);
}


/**
* @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), "CMPToken::delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "CMPToken::delegateBySig: invalid nonce");
        require(block.timestamp <= expiry, "CMPToken::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 misin
     * @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, "CMPToken::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 balances (not scale
        _delegates[delegator] = delegatee;

        _moveDelegates(currentDelegate, delegatee, delegatorBalance);

        emit DelegateChanged(delegator, currentDelegate, delegatee);
    }

    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, "CMPToken::_writeCheckpoint: block number exceeds 3

        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;
    }

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

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

}


contract CMPToken is DelegateERC20, Ownable {
    using SafeMath for uint256;

    uint256 private constant preMineSupply = 10000000 * 1e18;
    uint256 private constant maxSupply = 200000000 * 1e18;     // the total supply

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _minters;

    constructor() public ERC20("Compound Farming Token", "CMP"){
        _mint(msg.sender, preMineSupply);
    }

    // mint with max supply
    function mint(address _to, uint256 _amount) public onlyMinter returns (bool) {
        if (_amount.add(totalSupply()) > maxSupply) {
            return false;
        }
        _mint(_to, _amount);
        return true;
    }

    function addMinter(address _addMinter) public onlyOwner returns (bool) {
        require(_addMinter != address(0), "CMPToken: _addMinter is the zero address");
        return EnumerableSet.add(_minters, _addMinter);
    }

    function delMinter(address _delMinter) public onlyOwner returns (bool) {
        require(_delMinter != address(0), "CMPToken: _delMinter is the zero address");
        return EnumerableSet.remove(_minters, _delMinter);
    }
```

```
    function getMinterLength() public view returns (uint256) {
        return EnumerableSet.length(_minters);
    }

    function isMinter(address account) public view returns (bool) {
        return EnumerableSet.contains(_minters, account);
    }

    function getMinter(uint256 _index) public view onlyOwner returns (address){
        require(_index <= getMinterLength() - 1, "CMPToken: index out of bounds");
        return EnumerableSet.at(_minters, _index);
    }

    // modifier for mint function
    modifier onlyMinter() {
        require(isMinter(msg.sender), "caller is not the minter");
        _;
    }
}
```

Farm.sol

```
pragma solidity ^0.7.0;


// SPDX-License-Identifier: MIT
/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
```

```
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

//
/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
```

```
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
```

```
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
```

```
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}

//
/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
```

```
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
```

```
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data, string memory errorMessage) internal ret
        return functionCallWithValue(target, data, 0, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValu
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory er
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: value }(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memor
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data, string memory errorMessage) intern
        require(isContract(target), "Address: static call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.staticcall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
```

```
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) inte
        require(isContract(target), "Address: delegate call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _verifyCallResult(bool success, bytes memory returndata, string memory errorMessage) pri
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}

//
/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, valu
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
```

```
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
        // the target address contains contract code and also asserts for success in the low-level ca

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

//
/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
}

//
```

```
/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

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

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

//
/**
```

```
 * @dev Contract module that helps prevent reentrant calls to a function.
 *
 * Inheriting from `ReentrancyGuard` will make the {nonReentrant} modifier
 * available, which can be applied to functions to make sure there are no nested
 * (reentrant) calls to them.
 *
 * Note that because there is a single `nonReentrant` guard, functions marked as
 * `nonReentrant` may not call one another. This can be worked around by making
 * those functions `private`, and then adding `external` `nonReentrant` entry
 * points to them.
 *
 * TIP: If you would like to learn more about reentrancy and alternative ways
 * to protect against it, check out our blog post
 * https://blog.openzeppelin.com/reentrancy-after-istanbul/[Reentrancy After Istanbul].
 */
abstract contract ReentrancyGuard {
    // Booleans are more expensive than uint256 or any type that takes up a full
    // word because each write operation emits an extra SLOAD to first read the
    // slot's contents, replace the bits taken up by the boolean, and then write
    // back. This is the compiler's defense against contract upgrades and
    // pointer aliasing, and it cannot be disabled.

    // The values being non-zero value makes deployment a bit more expensive,
    // but in exchange the refund on every call to nonReentrant will be lower in
    // amount. Since refunds are capped to a percentage of the total
    // transaction's gas, it is best to keep them low in cases like this one, to
    // increase the likelihood of the full refund coming into effect.
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    constructor () internal {
        _status = _NOT_ENTERED;
    }

    /**
     * @dev Prevents a contract from calling itself, directly or indirectly.
     * Calling a `nonReentrant` function from another `nonReentrant`
     * function is not supported. It is possible to prevent this from happening
     * by making the `nonReentrant` function external, and make it call a
     * `private` function that does the actual work.
     */
    modifier nonReentrant() {
        // On the first call to nonReentrant, _notEntered will be true
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");

        // Any calls to nonReentrant after this point will fail
        _status = _ENTERED;

        _;

        // By storing the original value once again, a refund is triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _status = _NOT_ENTERED;
    }
}

//
/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
 * types.
 *
 * Sets have the following properties:
 *
```

```
 * - Elements are added, removed, and checked for existence in constant time
 * (O(1)).
 * - Elements are enumerated in O(n). No guarantees are made on the ordering.
 *
 *
 * contract Example {
 *     // Add the library methods
 *     using EnumerableSet for EnumerableSet.AddressSet;
 *
 *     // Declare a set state variable
 *     EnumerableSet.AddressSet private mySet;
 * }
 *
 *
 * As of v3.3.0, sets of type `bytes32` (`Bytes32Set`), `address` (`AddressSet`)
 * and `uint256` (`UintSet`) are supported.
 */
library EnumerableSet {
    // To implement this library for multiple types with as little code
    // repetition as possible, we write it in terms of a generic Set type with
    // bytes32 values.
    // The Set implementation uses private functions, and user-facing
    // implementations (such as AddressSet) are just wrappers around the
    // underlying Set.
    // This means that we can only create new EnumerableSets for types that fit
    // in bytes32.

    struct Set {
        // Storage of set values
        bytes32[] _values;

        // Position of the value in the `values` array, plus 1 because index 0
        // means a value is not in the set.
        mapping (bytes32 => uint256) _indexes;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function _add(Set storage set, bytes32 value) private returns (bool) {
        if (!_contains(set, value)) {
            set._values.push(value);
            // The value is stored at length-1, but we add 1 to all indexes
            // and use 0 as a sentinel value
            set._indexes[value] = set._values.length;
            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function _remove(Set storage set, bytes32 value) private returns (bool) {
        // We read and store the value's index to prevent multiple reads from the same storage slot
        uint256 valueIndex = set._indexes[value];

        if (valueIndex != 0) { // Equivalent to contains(set, value)
            // To delete an element from the _values array in O(1), we swap the element to delete wit
```

```
            // the array, and then remove the last element (sometimes called as 'swap and pop').
            // This modifies the order of the array, as noted in {at}.

            uint256 toDeleteIndex = valueIndex - 1;
            uint256 lastIndex = set._values.length - 1;

            // When the value to delete is the last one, the swap operation is unnecessary. However,
            // so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' stateme

            bytes32 lastvalue = set._values[lastIndex];

            // Move the last value to the index where the value to delete is
            set._values[toDeleteIndex] = lastvalue;
            // Update the index for the moved value
            set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

            // Delete the slot where the moved value was stored
            set._values.pop();

            // Delete the index for the deleted slot
            delete set._indexes[value];

            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function _contains(Set storage set, bytes32 value) private view returns (bool) {
        return set._indexes[value] != 0;
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function _length(Set storage set) private view returns (uint256) {
        return set._values.length;
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function _at(Set storage set, uint256 index) private view returns (bytes32) {
        require(set._values.length > index, "EnumerableSet: index out of bounds");
        return set._values[index];
    }

    // Bytes32Set

    struct Bytes32Set {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
```

```
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _add(set._inner, value);
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _remove(set._inner, value);
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(Bytes32Set storage set, bytes32 value) internal view returns (bool) {
        return _contains(set._inner, value);
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(Bytes32Set storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function at(Bytes32Set storage set, uint256 index) internal view returns (bytes32) {
        return _at(set._inner, index);
    }

    // AddressSet

    struct AddressSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(AddressSet storage set, address value) internal returns (bool) {
        return _add(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
```

```
     */
    function remove(AddressSet storage set, address value) internal returns (bool) {
        return _remove(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(AddressSet storage set, address value) internal view returns (bool) {
        return _contains(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(AddressSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function at(AddressSet storage set, uint256 index) internal view returns (address) {
        return address(uint160(uint256(_at(set._inner, index))));
    }


    // UintSet

    struct UintSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(UintSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(UintSet storage set, uint256 value) internal view returns (bool) {
        return _contains(set._inner, bytes32(value));
    }
```

```
    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
        return uint256(_at(set._inner, index));
    }
}

//
/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /**
     * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
     * a default value of 18.
     *
     * To select a different value for {decimals}, use {_setupDecimals}.
```

```
     *
     * All three of these values are immutable: they can only be set once during
     * construction.
     */
    constructor (string memory name_, string memory symbol_) public {
        _name = name_;
        _symbol = symbol_;
        _decimals = 18;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view virtual returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view virtual returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
     * called.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view virtual returns (uint8) {
        return _decimals;
    }

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view virtual override returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view virtual override returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
```

```
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override returns (uint256)
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * Requirements:
     *
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
```

```
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC2
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
```

```
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal virtual {
        _decimals = decimals_;
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks]
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
}

//
abstract contract DelegateERC20 is ERC20 {
    using SafeMath for uint256;

    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,add

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

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


    // support delegates mint
    function _mint(address account, uint256 amount) internal override virtual {
        super._mint(account, amount);

        // add delegates to the minter
        _moveDelegates(address(0), _delegates[account], amount);
    }


    function _transfer(address sender, address recipient, uint256 amount) internal override virtual {
        super._transfer(sender, recipient, amount);
        _moveDelegates(_delegates[sender], _delegates[recipient], amount);
    }


    /**
    * @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), "CMPToken::delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "CMPToken::delegateBySig: invalid nonce");
        require(block.timestamp <= expiry, "CMPToken::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 misin
     * @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, "CMPToken::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 balances (not scale
        _delegates[delegator] = delegatee;

        _moveDelegates(currentDelegate, delegatee, delegatorBalance);

        emit DelegateChanged(delegator, currentDelegate, delegatee);
    }

    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, "CMPToken::_writeCheckpoint: block number exceeds 3

        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;
    }

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

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

}

//
contract CMPToken is DelegateERC20, Ownable {
    using SafeMath for uint256;

    uint256 private constant preMineSupply = 10000000 * 1e18;
    uint256 private constant maxSupply = 200000000 * 1e18;     // the total supply

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _minters;

    constructor() public ERC20("Compound Farming Token", "CMP"){
        _mint(msg.sender, preMineSupply);
    }

    // mint with max supply
    function mint(address _to, uint256 _amount) public onlyMinter returns (bool) {
        if (_amount.add(totalSupply()) > maxSupply) {
            return false;
        }
        _mint(_to, _amount);
        return true;
    }

    function addMinter(address _addMinter) public onlyOwner returns (bool) {
        require(_addMinter != address(0), "CMPToken: _addMinter is the zero address");
        return EnumerableSet.add(_minters, _addMinter);
    }

    function delMinter(address _delMinter) public onlyOwner returns (bool) {
        require(_delMinter != address(0), "CMPToken: _delMinter is the zero address");
        return EnumerableSet.remove(_minters, _delMinter);
    }

    function getMinterLength() public view returns (uint256) {
        return EnumerableSet.length(_minters);
    }
```

```
    function isMinter(address account) public view returns (bool) {
        return EnumerableSet.contains(_minters, account);
    }

    function getMinter(uint256 _index) public view onlyOwner returns (address){
        require(_index <= getMinterLength() - 1, "CMPToken: index out of bounds");
        return EnumerableSet.at(_minters, _index);
    }

    // modifier for mint function
    modifier onlyMinter() {
        require(isMinter(msg.sender), "caller is not the minter");
        _;
    }
}

//
// For interacting with our own strategy
interface IStrategy {
    // Total want tokens managed by stratfegy
    function wantLockedTotal() external view returns (uint256);

    // Sum of all shares of users to wantLockedTotal
    function sharesTotal() external view returns (uint256);

    // Main want token compounding function
    function earn() external;

    // Transfer want tokens autoFarm -> strategy
    function deposit(address _userAddress, uint256 _wantAmt)
        external
        returns (uint256);

    // Transfer want tokens strategy -> autoFarm
    function withdraw(address _userAddress, uint256 _wantAmt)
        external
        returns (uint256);

    function inCaseTokensGetStuck(
        address _token,
        uint256 _amount,
        address _to
    ) external;
}

contract Farm is Ownable, ReentrancyGuard {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    // Info of each user.
    struct UserInfo {
        uint256 shares; // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt. See explanation below.

        // We do some fancy math here. Basically, any point in time, the amount of CMP
        // entitled to a user but is pending to be distributed is:
        //
        //   amount = user.shares / sharesTotal * wantLockedTotal
        //   pending reward = (amount * pool.accCmpPerShare) - user.rewardDebt
        //
        // Whenever a user deposits or withdraws want tokens to a pool. Here's what happens:
        //   1. The pool's `accCmpPerShare` (and `lastRewardBlock`) gets updated.
        //   2. User receives the pending reward sent to his/her address.
        //   3. User's `amount` gets updated.
        //   4. User's `rewardDebt` gets updated.
    }
```

```
    struct PoolInfo {
        IERC20 want; // Address of the want token.
        uint256 allocPoint; // How many allocation points assigned to this pool. CMP to distribute pe
        uint256 lastRewardBlock; // Last block number that CMP distribution occurs.
        uint256 accCmpPerShare; // Accumulated CMP per share, times 1e12. See below.
        address strat; // Strategy address that will auto compound want tokens
    }

    IERC20 public cmp;

    uint256 public ownerCMPReward = 187500; // 15/80=18.75%

    uint256 public CMPMaxSupply = 190000000 * 1e18;
    // uint256 public cmpPerBlock = 338 * 1e16; // CMP tokens created per block
    uint256 public cmpPerBlock = 3780 * 1e16;
    uint256 public startBlock = uint256(-1); // default value, will be overwritten by deploy

    PoolInfo[] public poolInfo; // Info of each pool.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo; // Info of each user that stake
    uint256 public totalAllocPoint = 0; // Total allocation points. Must be the sum of all allocation
    uint256 public halvingPeriod = 0;

    event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
    event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
    event EmergencyWithdraw(
        address indexed user,
        uint256 indexed pid,
        uint256 amount
    );

    constructor(
        CMPToken _cmp,
        uint256 _startBlock
    ) public {
        cmp = _cmp;
        startBlock = _startBlock;
    }

    function setHalvingPeriod(uint256 _block) public onlyOwner {
        halvingPeriod = _block;
    }

    // Set the number of mdx produced by each block
    function setMdxPerBlock(uint256 _cmpPerBlock) public onlyOwner {
        massUpdatePools();
        cmpPerBlock = _cmpPerBlock;
    }

    function poolLength() external view returns (uint256) {
        return poolInfo.length;
    }

    // Add a new lp to the pool. Can only be called by the owner.
    // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do. (Only if
    function add(
        uint256 _allocPoint,
        IERC20 _want,
        bool _withUpdate,
        address _strat
    ) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock =
            block.number > startBlock ? block.number : startBlock;
```

```
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(
            PoolInfo({
                want: _want,
                allocPoint: _allocPoint,
                lastRewardBlock: lastRewardBlock,
                accCmpPerShare: 0,
                strat: _strat
            })
        );
    }

    // Update the given pool's CPM allocation point. Can only be called by the owner.
    function set(
        uint256 _pid,
        uint256 _allocPoint,
        bool _withUpdate
    ) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
            _allocPoint
        );
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    function phase(uint256 blockNumber) public view returns (uint256) {
        if (halvingPeriod == 0) {
            return 0;
        }
        if (blockNumber > startBlock) {
            return (blockNumber.sub(startBlock).sub(1)).div(halvingPeriod);
        }
        return 0;
    }

    function reward(uint256 blockNumber) public view returns (uint256) {
        uint256 _phase = phase(blockNumber);
        return cmpPerBlock.div(2 ** _phase);
    }

    // Return reward multiplier over the given _from to _to block.
    function getMultiplier(uint256 _from, uint256 _to)
        public
        view
        returns (uint256)
    {
        if (cmp.totalSupply() >= CMPMaxSupply) {
            return 0;
        }
        return _to.sub(_from);
    }

    // View function to see pending CMP on frontend.
    function pendingCMP(uint256 _pid, address _user)
        external
        view
        returns (uint256)
    {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accCmpPerShare = pool.accCmpPerShare;
        uint256 sharesTotal = IStrategy(pool.strat).sharesTotal();
        if (block.number > pool.lastRewardBlock && sharesTotal != 0) {
            uint256 multiplier =
```

```
            getMultiplier(pool.lastRewardBlock, block.number);
        uint256 cmpReward =
            multiplier.mul(reward(block.number)).mul(pool.allocPoint).div(
                totalAllocPoint
            );
        accCmpPerShare = accCmpPerShare.add(
            cmpReward.mul(1e12).div(sharesTotal)
        );
    }
    return user.shares.mul(accCmpPerShare).div(1e12).sub(user.rewardDebt);
}

// View function to see staked Want tokens on frontend.
function stakedWantTokens(uint256 _pid, address _user)
    external
    view
    returns (uint256)
{
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_user];

    uint256 sharesTotal = IStrategy(pool.strat).sharesTotal();
    uint256 wantLockedTotal =
        IStrategy(poolInfo[_pid].strat).wantLockedTotal();
    if (sharesTotal == 0) {
        return 0;
    }
    return user.shares.mul(wantLockedTotal).div(sharesTotal);
}

// Update reward variables for all pools. Be careful of gas spending!
function massUpdatePools() public onlyOwner{
    uint256 length = poolInfo.length;
    for (uint256 pid = 0; pid < length; ++pid) {
        updatePool(pid);
    }
}

// Update reward variables of the given pool to be up-to-date.
function updatePool(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    if (block.number <= pool.lastRewardBlock) {
        return;
    }
    uint256 sharesTotal = IStrategy(pool.strat).sharesTotal();
    if (sharesTotal == 0) {
        pool.lastRewardBlock = block.number;
        return;
    }
    uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
    if (multiplier <= 0) {
        return;
    }
    uint256 cmpReward =
        multiplier.mul(reward(block.number)).mul(pool.allocPoint).div(
            totalAllocPoint
        );

    CMPToken(address(cmp)).mint(
        owner(),
        cmpReward.mul(ownerCMPReward).div(1000000)
    );
    CMPToken(address(cmp)).mint(address(this), cmpReward);

    pool.accCmpPerShare = pool.accCmpPerShare.add(
        cmpReward.mul(1e12).div(sharesTotal)
```

```
        );
        pool.lastRewardBlock = block.number;
    }

    // Want tokens moved from user -> CMPFarm (CMP allocation) -> Strat (compounding)
    function deposit(uint256 _pid, uint256 _wantAmt) public nonReentrant {
        updatePool(_pid);
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];

        if (user.shares > 0) {
            uint256 pending =
                user.shares.mul(pool.accCmpPerShare).div(1e12).sub(
                    user.rewardDebt
                );
            if (pending > 0) {
                safeCMPTransfer(msg.sender, pending);
            }
        }
        if (_wantAmt > 0) {
            pool.want.safeTransferFrom(
                address(msg.sender),
                address(this),
                _wantAmt
            );

            pool.want.safeIncreaseAllowance(pool.strat, _wantAmt);
            uint256 sharesAdded =
                IStrategy(poolInfo[_pid].strat).deposit(msg.sender, _wantAmt);
            user.shares = user.shares.add(sharesAdded);
        }
        user.rewardDebt = user.shares.mul(pool.accCmpPerShare).div(1e12);
        emit Deposit(msg.sender, _pid, _wantAmt);
    }

    // Withdraw LP tokens from MasterChef.
    function withdraw(uint256 _pid, uint256 _wantAmt) public nonReentrant {
        updatePool(_pid);

        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];

        uint256 wantLockedTotal =
            IStrategy(poolInfo[_pid].strat).wantLockedTotal();
        uint256 sharesTotal = IStrategy(poolInfo[_pid].strat).sharesTotal();

        require(user.shares > 0, "user.shares is 0");
        require(sharesTotal > 0, "sharesTotal is 0");

        // Withdraw pending CMP
        uint256 pending =
            user.shares.mul(pool.accCmpPerShare).div(1e12).sub(
                user.rewardDebt
            );
        if (pending > 0) {
            safeCMPTransfer(msg.sender, pending);
        }

        // Withdraw want tokens
        uint256 amount = user.shares.mul(wantLockedTotal).div(sharesTotal);
        if (_wantAmt > amount) {
            _wantAmt = amount;
        }
        if (_wantAmt > 0) {
            uint256 sharesRemoved =
                IStrategy(poolInfo[_pid].strat).withdraw(msg.sender, _wantAmt);
```

```
            if (sharesRemoved > user.shares) {
                user.shares = 0;
            } else {
                user.shares = user.shares.sub(sharesRemoved);
            }

            uint256 wantBal = IERC20(pool.want).balanceOf(address(this));
            if (wantBal < _wantAmt) {
                _wantAmt = wantBal;
            }
            pool.want.safeTransfer(address(msg.sender), _wantAmt);
        }
        user.rewardDebt = user.shares.mul(pool.accCmpPerShare).div(1e12);
        emit Withdraw(msg.sender, _pid, _wantAmt);
    }

    // Withdraw without caring about rewards. EMERGENCY ONLY.
    function emergencyWithdraw(uint256 _pid) public nonReentrant {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];

        uint256 wantLockedTotal =
            IStrategy(poolInfo[_pid].strat).wantLockedTotal();
        uint256 sharesTotal = IStrategy(poolInfo[_pid].strat).sharesTotal();
        uint256 amount = user.shares.mul(wantLockedTotal).div(sharesTotal);

        IStrategy(poolInfo[_pid].strat).withdraw(msg.sender, amount);

        pool.want.safeTransfer(address(msg.sender), amount);
        emit EmergencyWithdraw(msg.sender, _pid, amount);
        user.shares = 0;
        user.rewardDebt = 0;
    }

    // Safe CMP transfer function, just in case if rounding error causes pool to not have enough
    function safeCMPTransfer(address _to, uint256 _CMPAmt) internal {
        uint256 CMPBal = cmp.balanceOf(address(this));
        if (_CMPAmt > CMPBal) {
            cmp.transfer(_to, CMPBal);
        } else {
            cmp.transfer(_to, _CMPAmt);
        }
    }

    function inCaseTokensGetStuck(address _token, uint256 _amount)
        public
        onlyOwner
    {
        IERC20(_token).safeTransfer(msg.sender, _amount);
    }
}
```

StratX_MDEX.sol

```
pragma solidity ^0.7.0;


// SPDX-License-Identifier: MIT
/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
```

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

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
```

```
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

//
/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
```

```
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
```

```
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
```

```
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}

//
/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
```

```
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data, string memory errorMessage) internal ret
        return functionCallWithValue(target, data, 0, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValu
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory er
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");
```

```
        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: value }(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memor
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data, string memory errorMessage) intern
        require(isContract(target), "Address: static call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.staticcall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) inte
        require(isContract(target), "Address: delegate call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _verifyCallResult(bool success, bytes memory returndata, string memory errorMessage) pri
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
```

```
            } else {
                revert(errorMessage);
            }
        }
    }
}

//
/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, valu
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
```

```
        // the target address contains contract code and also asserts for success in the low-level ca

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

//
/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
}

//
/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual returns (address) {
        return _owner;
    }
```

```
    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

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

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

//
/**
 * @dev Contract module that helps prevent reentrant calls to a function.
 *
 * Inheriting from `ReentrancyGuard` will make the {nonReentrant} modifier
 * available, which can be applied to functions to make sure there are no nested
 * (reentrant) calls to them.
 *
 * Note that because there is a single `nonReentrant` guard, functions marked as
 * `nonReentrant` may not call one another. This can be worked around by making
 * those functions `private`, and then adding `external` `nonReentrant` entry
 * points to them.
 *
 * TIP: If you would like to learn more about reentrancy and alternative ways
 * to protect against it, check out our blog post
 * https://blog.openzeppelin.com/reentrancy-after-istanbul/[Reentrancy After Istanbul].
 */
abstract contract ReentrancyGuard {
    // Booleans are more expensive than uint256 or any type that takes up a full
    // word because each write operation emits an extra SLOAD to first read the
    // slot's contents, replace the bits taken up by the boolean, and then write
    // back. This is the compiler's defense against contract upgrades and
    // pointer aliasing, and it cannot be disabled.

    // The values being non-zero value makes deployment a bit more expensive,
    // but in exchange the refund on every call to nonReentrant will be lower in
    // amount. Since refunds are capped to a percentage of the total
    // transaction's gas, it is best to keep them low in cases like this one, to
    // increase the likelihood of the full refund coming into effect.
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    constructor () internal {
```

```
        _status = _NOT_ENTERED;
    }

    /**
     * @dev Prevents a contract from calling itself, directly or indirectly.
     * Calling a `nonReentrant` function from another `nonReentrant`
     * function is not supported. It is possible to prevent this from happening
     * by making the `nonReentrant` function external, and make it call a
     * `private` function that does the actual work.
     */
    modifier nonReentrant() {
        // On the first call to nonReentrant, _notEntered will be true
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");

        // Any calls to nonReentrant after this point will fail
        _status = _ENTERED;

        _;

        // By storing the original value once again, a refund is triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _status = _NOT_ENTERED;
    }
}

//
/**
 * @dev Contract module which allows children to implement an emergency stop
 * mechanism that can be triggered by an authorized account.
 *
 * This module is used through inheritance. It will make available the
 * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
 * the functions of your contract. Note that they will not be pausable by
 * simply including this module, only once the modifiers are put in place.
 */
abstract contract Pausable is Context {
    /**
     * @dev Emitted when the pause is triggered by `account`.
     */
    event Paused(address account);

    /**
     * @dev Emitted when the pause is lifted by `account`.
     */
    event Unpaused(address account);

    bool private _paused;

    /**
     * @dev Initializes the contract in unpaused state.
     */
    constructor () internal {
        _paused = false;
    }

    /**
     * @dev Returns true if the contract is paused, and false otherwise.
     */
    function paused() public view virtual returns (bool) {
        return _paused;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
     *
     * Requirements:
```

```
     *
     * - The contract must not be paused.
     */
    modifier whenNotPaused() {
        require(!paused(), "Pausable: paused");
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    modifier whenPaused() {
        require(paused(), "Pausable: not paused");
        _;
    }

    /**
     * @dev Triggers stopped state.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    function _pause() internal virtual whenNotPaused {
        _paused = true;
        emit Paused(_msgSender());
    }

    /**
     * @dev Returns to normal state.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    function _unpause() internal virtual whenPaused {
        _paused = false;
        emit Unpaused(_msgSender());
    }
}

//
/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
 * types.
 *
 * Sets have the following properties:
 *
 * - Elements are added, removed, and checked for existence in constant time
 * (O(1)).
 * - Elements are enumerated in O(n). No guarantees are made on the ordering.
 *
 *
 * contract Example {
 *     // Add the library methods
 *     using EnumerableSet for EnumerableSet.AddressSet;
 *
 *     // Declare a set state variable
 *     EnumerableSet.AddressSet private mySet;
 * }
 *
```

```
 *
 * As of v3.3.0, sets of type `bytes32` (`Bytes32Set`), `address` (`AddressSet`)
 * and `uint256` (`UintSet`) are supported.
 */
library EnumerableSet {
    // To implement this library for multiple types with as little code
    // repetition as possible, we write it in terms of a generic Set type with
    // bytes32 values.
    // The Set implementation uses private functions, and user-facing
    // implementations (such as AddressSet) are just wrappers around the
    // underlying Set.
    // This means that we can only create new EnumerableSets for types that fit
    // in bytes32.

    struct Set {
        // Storage of set values
        bytes32[] _values;

        // Position of the value in the `values` array, plus 1 because index 0
        // means a value is not in the set.
        mapping (bytes32 => uint256) _indexes;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function _add(Set storage set, bytes32 value) private returns (bool) {
        if (!_contains(set, value)) {
            set._values.push(value);
            // The value is stored at length-1, but we add 1 to all indexes
            // and use 0 as a sentinel value
            set._indexes[value] = set._values.length;
            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function _remove(Set storage set, bytes32 value) private returns (bool) {
        // We read and store the value's index to prevent multiple reads from the same storage slot
        uint256 valueIndex = set._indexes[value];

        if (valueIndex != 0) { // Equivalent to contains(set, value)
            // To delete an element from the _values array in O(1), we swap the element to delete wit
            // the array, and then remove the last element (sometimes called as 'swap and pop').
            // This modifies the order of the array, as noted in {at}.

            uint256 toDeleteIndex = valueIndex - 1;
            uint256 lastIndex = set._values.length - 1;

            // When the value to delete is the last one, the swap operation is unnecessary. However,
            // so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' stateme

            bytes32 lastvalue = set._values[lastIndex];

            // Move the last value to the index where the value to delete is
            set._values[toDeleteIndex] = lastvalue;
```

```
            // Update the index for the moved value
            set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

            // Delete the slot where the moved value was stored
            set._values.pop();

            // Delete the index for the deleted slot
            delete set._indexes[value];

            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function _contains(Set storage set, bytes32 value) private view returns (bool) {
        return set._indexes[value] != 0;
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function _length(Set storage set) private view returns (uint256) {
        return set._values.length;
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function _at(Set storage set, uint256 index) private view returns (bytes32) {
        require(set._values.length > index, "EnumerableSet: index out of bounds");
        return set._values[index];
    }

    // Bytes32Set

    struct Bytes32Set {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _add(set._inner, value);
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
```

```
    function remove(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _remove(set._inner, value);
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(Bytes32Set storage set, bytes32 value) internal view returns (bool) {
        return _contains(set._inner, value);
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(Bytes32Set storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function at(Bytes32Set storage set, uint256 index) internal view returns (bytes32) {
        return _at(set._inner, index);
    }

    // AddressSet

    struct AddressSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(AddressSet storage set, address value) internal returns (bool) {
        return _add(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(AddressSet storage set, address value) internal returns (bool) {
        return _remove(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(AddressSet storage set, address value) internal view returns (bool) {
        return _contains(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
```

```
     * @dev Returns the number of values in the set. O(1).
     */
    function length(AddressSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(AddressSet storage set, uint256 index) internal view returns (address) {
        return address(uint160(uint256(_at(set._inner, index))));
    }


    // UintSet

    struct UintSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(UintSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(UintSet storage set, uint256 value) internal view returns (bool) {
        return _contains(set._inner, bytes32(value));
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
```

```
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
        return uint256(_at(set._inner, index));
    }
}

//
interface IPancakeswapFarm {
    function poolLength() external view returns (uint256);

    function userInfo() external view returns (uint256);

    // Return reward multiplier over the given _from to _to block.
    function getMultiplier(uint256 _from, uint256 _to)
        external
        view
        returns (uint256);

    // View function to see pending CAKEs on frontend.
    function pendingCake(uint256 _pid, address _user)
        external
        view
        returns (uint256);

    // Deposit LP tokens to MasterChef for CAKE allocation.
    function deposit(uint256 _pid, uint256 _amount) external;

    // Withdraw LP tokens from MasterChef.
    function withdraw(uint256 _pid, uint256 _amount) external;

    // Stake CAKE tokens to MasterChef
    function enterStaking(uint256 _amount) external;

    // Withdraw CAKE tokens from STAKING.
    function leaveStaking(uint256 _amount) external;

    // Withdraw without caring about rewards. EMERGENCY ONLY.
    function emergencyWithdraw(uint256 _pid) external;
}

interface IPancakeRouter01 {
    function factory() external pure returns (address);

    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        external
        returns (
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        );
```

```
    function addLiquidityETH(
        address token,
        uint256 amountTokenDesired,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    )
        external
        payable
        returns (
            uint256 amountToken,
            uint256 amountETH,
            uint256 liquidity
        );

    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountA, uint256 amountB);

    function removeLiquidityETH(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountToken, uint256 amountETH);

    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountA, uint256 amountB);

    function removeLiquidityETHWithPermit(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountToken, uint256 amountETH);

    function swapExactTokensForTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
```

```
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);

    function swapTokensForExactTokens(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);

    function swapExactETHForTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable returns (uint256[] memory amounts);

    function swapTokensForExactETH(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);

    function swapExactTokensForETH(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);

    function swapETHForExactTokens(
        uint256 amountOut,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable returns (uint256[] memory amounts);

    function quote(
        uint256 amountA,
        uint256 reserveA,
        uint256 reserveB
    ) external pure returns (uint256 amountB);

    function getAmountOut(
        uint256 amountIn,
        uint256 reserveIn,
        uint256 reserveOut
    ) external pure returns (uint256 amountOut);

    function getAmountIn(
        uint256 amountOut,
        uint256 reserveIn,
        uint256 reserveOut
    ) external pure returns (uint256 amountIn);

    function getAmountsOut(uint256 amountIn, address[] calldata path)
        external
        view
        returns (uint256[] memory amounts);

    function getAmountsIn(uint256 amountOut, address[] calldata path)
```

```
        external
        view
        returns (uint256[] memory amounts);
}

interface IPancakeRouter02 is IPancakeRouter01 {
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountETH);

    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountETH);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;

    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable;

    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;
}

interface IMDEXSwapMining {
    function takerWithdraw() external;
}

interface IWBNB is IERC20 {
    function deposit() external payable;

    function withdraw(uint256 wad) external;
}

abstract contract StratX is Ownable, ReentrancyGuard, Pausable {
    // Maximises yields in pancakeswap

    using SafeMath for uint256;
```

```
    using SafeERC20 for IERC20;

    bool public isCAKEStaking; // only for staking CAKE using pancakeswap's native CAKE staking contr
    bool public isAutoComp; // this vault is purely for staking. eg. WBNB-AUTO staking vault.

    address public farmContractAddress; // address of farm, eg, PCS, Thugs etc.
    uint256 public pid; // pid of pool in farmContractAddress
    address public wantAddress;
    address public token0Address;
    address public token1Address;
    address public earnedAddress;
    address public uniRouterAddress; // uniswap, pancakeswap etc

    address public wbnbAddress;
    address public autoFarmAddress;
    address public CMPAddress;
    address public govAddress; // timelock contract
    bool public onlyGov = true;

    uint256 public lastEarnBlock = 0;
    uint256 public wantLockedTotal = 0;
    uint256 public sharesTotal = 0;

    uint256 public controllerFee = 160; // 70;
    uint256 public constant controllerFeeMax = 10000; // 100 = 1%
    uint256 public constant controllerFeeUL = 500;

    uint256 public buyBackRate = 0; // 250;
    uint256 public constant buyBackRateMax = 10000; // 100 = 1%
    uint256 public constant buyBackRateUL = 800;
    address public buyBackAddress = 0x3b5F91311d4c6165236565987B348327027ae007;
    address public rewardsAddress;

    uint256 public entranceFeeFactor = 9990; // < 0.1% entrance fee - goes to pool + prevents front-r
    uint256 public constant entranceFeeFactorMax = 10000;
    uint256 public constant entranceFeeFactorLL = 9950; // 0.5% is the max entrance fee settable. LL

    uint256 public withdrawFeeFactor = 10000; // 0.1% withdraw fee - goes to pool
    uint256 public constant withdrawFeeFactorMax = 10000;
    uint256 public constant withdrawFeeFactorLL = 9950; // 0.5% is the max withdraw fee settable. LL

    uint256 public slippageFactor = 950; // 5% default slippage tolerance
    uint256 public constant slippageFactorUL = 995;

    address[] public earnedToCMPPath;
    address[] public earnedToToken0Path;
    address[] public earnedToToken1Path;
    address[] public token0ToEarnedPath;
    address[] public token1ToEarnedPath;
    address[] public earnedToWantPath;

    event SetSettings(
        uint256 _entranceFeeFactor,
        uint256 _withdrawFeeFactor,
        uint256 _controllerFee,
        uint256 _buyBackRate,
        uint256 _slippageFactor
    );

    event SetGov(address _govAddress);
    event SetOnlyGov(bool _onlyGov);
    event SetUniRouterAddress(address _uniRouterAddress);
    event SetBuyBackAddress(address _buyBackAddress);
    event SetRewardsAddress(address _rewardsAddress);

    modifier onlyAllowGov() {
```

```
        require(msg.sender == govAddress, "!gov");
        _;
    }

    // Receives new deposits from user
    function deposit(address _userAddress, uint256 _wantAmt)
        public
        virtual
        onlyOwner
        nonReentrant
        whenNotPaused
        returns (uint256)
    {
        IERC20(wantAddress).safeTransferFrom(
            address(msg.sender),
            address(this),
            _wantAmt
        );

        uint256 sharesAdded = _wantAmt;
        if (wantLockedTotal > 0 && sharesTotal > 0) {
            sharesAdded = _wantAmt
                .mul(sharesTotal)
                .mul(entranceFeeFactor)
                .div(wantLockedTotal)
                .div(entranceFeeFactorMax);
        }
        sharesTotal = sharesTotal.add(sharesAdded);

        if (isAutoComp) {
            _farm();
        } else {
            wantLockedTotal = wantLockedTotal.add(_wantAmt);
        }

        return sharesAdded;
    }

    function farm() public virtual nonReentrant {
        _farm();
    }

    function _farm() internal virtual {
        require(isAutoComp, "!isAutoComp");
        uint256 wantAmt = IERC20(wantAddress).balanceOf(address(this));
        wantLockedTotal = wantLockedTotal.add(wantAmt);
        IERC20(wantAddress).safeIncreaseAllowance(farmContractAddress, wantAmt);

        if (isCAKEStaking) {
            IPancakeswapFarm(farmContractAddress).enterStaking(wantAmt); // Just for CAKE staking, we
        } else {
            IPancakeswapFarm(farmContractAddress).deposit(pid, wantAmt);
        }
    }

    function _unfarm(uint256 _wantAmt) internal virtual {
        if (isCAKEStaking) {
            IPancakeswapFarm(farmContractAddress).leaveStaking(_wantAmt); // Just for CAKE staking, w
        } else {
            IPancakeswapFarm(farmContractAddress).withdraw(pid, _wantAmt);
        }
    }

    function withdraw(address _userAddress, uint256 _wantAmt)
        public
        virtual
```

```
        onlyOwner
        nonReentrant
        returns (uint256)
    {
        require(_wantAmt > 0, "_wantAmt <= 0");

        uint256 sharesRemoved = _wantAmt.mul(sharesTotal).div(wantLockedTotal);
        if (sharesRemoved > sharesTotal) {
            sharesRemoved = sharesTotal;
        }
        sharesTotal = sharesTotal.sub(sharesRemoved);

        if (withdrawFeeFactor < withdrawFeeFactorMax) {
            _wantAmt = _wantAmt.mul(withdrawFeeFactor).div(
                withdrawFeeFactorMax
            );
        }

        if (isAutoComp) {
            _unfarm(_wantAmt);
        }

        uint256 wantAmt = IERC20(wantAddress).balanceOf(address(this));
        if (_wantAmt > wantAmt) {
            _wantAmt = wantAmt;
        }

        if (wantLockedTotal < _wantAmt) {
            _wantAmt = wantLockedTotal;
        }

        wantLockedTotal = wantLockedTotal.sub(_wantAmt);

        IERC20(wantAddress).safeTransfer(autoFarmAddress, _wantAmt);

        return sharesRemoved;
    }

    // 1. Harvest farm tokens
    // 2. Converts farm tokens into want tokens
    // 3. Deposits want tokens

    function earn() public virtual nonReentrant whenNotPaused {
        require(isAutoComp, "!isAutoComp");
        if (onlyGov) {
            require(msg.sender == govAddress, "!gov");
        }

        // Harvest farm tokens
        _unfarm(0);

        if (earnedAddress == wbnbAddress) {
            _wrapBNB();
        }

        // Converts farm tokens into want tokens
        uint256 earnedAmt = IERC20(earnedAddress).balanceOf(address(this));

        earnedAmt = distributeFees(earnedAmt);
        earnedAmt = buyBack(earnedAmt);

        if (isCAKEStaking) {
            lastEarnBlock = block.number;
            _farm();
            return;
        }
```

```
        IERC20(earnedAddress).safeApprove(uniRouterAddress, 0);
        IERC20(earnedAddress).safeIncreaseAllowance(
            uniRouterAddress,
            earnedAmt
        );

        // single coin
        if(token0Address == address(0x0) || token1Address == address(0x0)){
            // earned is the same as want, do nothing
            // earned is different from want, swap to want
            if(earnedAddress != wantAddress) {
                // otherwise need to conver the earned to want token
                _safeSwap(
                    uniRouterAddress,
                    earnedAmt,
                    slippageFactor,
                    earnedToWantPath,
                    address(this),
                    block.timestamp.add(600)
                );
            }

            lastEarnBlock = block.number;
            _farm();
            return;
        }

        // LP coin

        if (earnedAddress != token0Address) {
            // Swap half earned to token0
            _safeSwap(
                uniRouterAddress,
                earnedAmt.div(2),
                slippageFactor,
                earnedToToken0Path,
                address(this),
                block.timestamp.add(600)
            );
        }

        if (earnedAddress != token1Address) {
            // Swap half earned to token1
            _safeSwap(
                uniRouterAddress,
                earnedAmt.div(2),
                slippageFactor,
                earnedToToken1Path,
                address(this),
                block.timestamp.add(600)
            );
        }

        // Get want tokens, ie. add liquidity
        uint256 token0Amt = IERC20(token0Address).balanceOf(address(this));
        uint256 token1Amt = IERC20(token1Address).balanceOf(address(this));
        if (token0Amt > 0 && token1Amt > 0) {
            IERC20(token0Address).safeIncreaseAllowance(
                uniRouterAddress,
                token0Amt
            );
            IERC20(token1Address).safeIncreaseAllowance(
                uniRouterAddress,
                token1Amt
            );
```

```
            IPancakeRouter02(uniRouterAddress).addLiquidity(
                token0Address,
                token1Address,
                token0Amt,
                token1Amt,
                0,
                0,
                address(this),
                block.timestamp.add(600)
            );
        }

        lastEarnBlock = block.number;

        _farm();
    }

    function buyBack(uint256 _earnedAmt) internal virtual returns (uint256) {
        if (buyBackRate <= 0) {
            return _earnedAmt;
        }

        uint256 buyBackAmt = _earnedAmt.mul(buyBackRate).div(buyBackRateMax);

        if (earnedAddress == CMPAddress) {
            IERC20(earnedAddress).safeTransfer(buyBackAddress, buyBackAmt);
        } else {
            IERC20(earnedAddress).safeIncreaseAllowance(
                uniRouterAddress,
                buyBackAmt
            );

            _safeSwap(
                uniRouterAddress,
                buyBackAmt,
                slippageFactor,
                earnedToCMPPath,
                buyBackAddress,
                block.timestamp.add(600)
            );
        }

        return _earnedAmt.sub(buyBackAmt);
    }

    function distributeFees(uint256 _earnedAmt)
        internal
        virtual
        returns (uint256)
    {
        if (_earnedAmt > 0) {
            // Performance fee
            if (controllerFee > 0) {
                uint256 fee =
                    _earnedAmt.mul(controllerFee).div(controllerFeeMax);
                IERC20(earnedAddress).safeTransfer(rewardsAddress, fee);
                _earnedAmt = _earnedAmt.sub(fee);
            }
        }

        return _earnedAmt;
    }

    function convertDustToEarned() public virtual whenNotPaused {
        require(isAutoComp, "!isAutoComp");
        require(!isCAKEStaking, "isCAKEStaking");
```

```
        // Converts dust tokens into earned tokens, which will be reinvested on the next earn().

        // Converts token0 dust (if any) to earned tokens
        uint256 token0Amt = IERC20(token0Address).balanceOf(address(this));
        if (token0Address != earnedAddress && token0Amt > 0) {
            IERC20(token0Address).safeIncreaseAllowance(
                uniRouterAddress,
                token0Amt
            );

            // Swap all dust tokens to earned tokens
            _safeSwap(
                uniRouterAddress,
                token0Amt,
                slippageFactor,
                token0ToEarnedPath,
                address(this),
                block.timestamp.add(600)
            );
        }

        // Converts token1 dust (if any) to earned tokens
        uint256 token1Amt = IERC20(token1Address).balanceOf(address(this));
        if (token1Address != earnedAddress && token1Amt > 0) {
            IERC20(token1Address).safeIncreaseAllowance(
                uniRouterAddress,
                token1Amt
            );

            // Swap all dust tokens to earned tokens
            _safeSwap(
                uniRouterAddress,
                token1Amt,
                slippageFactor,
                token1ToEarnedPath,
                address(this),
                block.timestamp.add(600)
            );
        }
    }

    function pause() public virtual onlyAllowGov {
        _pause();
    }

    function unpause() public virtual onlyAllowGov {
        _unpause();
    }

    function setSettings(
        uint256 _entranceFeeFactor,
        uint256 _withdrawFeeFactor,
        uint256 _controllerFee,
        uint256 _buyBackRate,
        uint256 _slippageFactor
    ) public virtual onlyAllowGov {
        require(
            _entranceFeeFactor >= entranceFeeFactorLL,
            "_entranceFeeFactor too low"
        );
        require(
            _entranceFeeFactor <= entranceFeeFactorMax,
            "_entranceFeeFactor too high"
        );
        entranceFeeFactor = _entranceFeeFactor;
```

```
        require(
            _withdrawFeeFactor >= withdrawFeeFactorLL,
            "_withdrawFeeFactor too low"
        );
        require(
            _withdrawFeeFactor <= withdrawFeeFactorMax,
            "_withdrawFeeFactor too high"
        );
        withdrawFeeFactor = _withdrawFeeFactor;

        require(_controllerFee <= controllerFeeUL, "_controllerFee too high");
        controllerFee = _controllerFee;

        require(_buyBackRate <= buyBackRateUL, "_buyBackRate too high");
        buyBackRate = _buyBackRate;

        require(
            _slippageFactor <= slippageFactorUL,
            "_slippageFactor too high"
        );
        slippageFactor = _slippageFactor;

        emit SetSettings(
            _entranceFeeFactor,
            _withdrawFeeFactor,
            _controllerFee,
            _buyBackRate,
            _slippageFactor
        );
    }

    function setGov(address _govAddress) public virtual onlyAllowGov {
        govAddress = _govAddress;
        emit SetGov(_govAddress);
    }

    function setOnlyGov(bool _onlyGov) public virtual onlyAllowGov {
        onlyGov = _onlyGov;
        emit SetOnlyGov(_onlyGov);
    }

    function setUniRouterAddress(address _uniRouterAddress)
        public
        virtual
        onlyAllowGov
    {
        uniRouterAddress = _uniRouterAddress;
        emit SetUniRouterAddress(_uniRouterAddress);
    }

    function setBuyBackAddress(address _buyBackAddress)
        public
        virtual
        onlyAllowGov
    {
        buyBackAddress = _buyBackAddress;
        emit SetBuyBackAddress(_buyBackAddress);
    }

    function setRewardsAddress(address _rewardsAddress)
        public
        virtual
        onlyAllowGov
    {
        rewardsAddress = _rewardsAddress;
```

```
        emit SetRewardsAddress(_rewardsAddress);
    }

    function inCaseTokensGetStuck(
        address _token,
        uint256 _amount,
        address _to
    ) public virtual onlyAllowGov {
        require(_token != earnedAddress, "!safe");
        require(_token != wantAddress, "!safe");
        IERC20(_token).safeTransfer(_to, _amount);
    }

    function _wrapBNB() internal virtual {
        // BNB -> WBNB
        uint256 bnbBal = address(this).balance;
        if (bnbBal > 0) {
            IWBNB(wbnbAddress).deposit{value: bnbBal}(); // BNB -> WBNB
        }
    }

    function wrapBNB() public virtual onlyAllowGov {
        _wrapBNB();
    }

    function _safeSwap(
        address _uniRouterAddress,
        uint256 _amountIn,
        uint256 _slippageFactor,
        address[] memory _path,
        address _to,
        uint256 _deadline
    ) internal virtual {
        uint256[] memory amounts =
            IPancakeRouter02(_uniRouterAddress).getAmountsOut(_amountIn, _path);
        uint256 amountOut = amounts[amounts.length.sub(1)];

        IPancakeRouter02(_uniRouterAddress)
            .swapExactTokensForTokensSupportingFeeOnTransferTokens(
            _amountIn,
            amountOut.mul(_slippageFactor).div(1000),
            _path,
            _to,
            _deadline
        );
    }
}

contract StratX_MDEX is StratX {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    address public MDXAddress;
    address public MDEXSwapMiningAddress;
    address[] public MDXToEarnedPath;

    constructor(
        address[] memory _addresses,
        uint256 _pid,
        bool _isCAKEStaking,
        bool _isAutoComp,
        address[] memory _earnedToCMPPath,
        address[] memory _earnedToToken0Path,
        address[] memory _earnedToToken1Path,
        address[] memory _token0ToEarnedPath,
        address[] memory _token1ToEarnedPath,
```

```
        address[] memory _earnedToWantPath,
        uint256 _controllerFee,
        uint256 _buyBackRate,
        uint256 _entranceFeeFactor,
        uint256 _withdrawFeeFactor,
        address[] memory _MDXToEarnedPath
    ) public {
        wbnbAddress = _addresses[0];
        govAddress = _addresses[1];
        autoFarmAddress = _addresses[2];
        CMPAddress = _addresses[3];

        wantAddress = _addresses[4];
        token0Address = _addresses[5];
        token1Address = _addresses[6];
        earnedAddress = _addresses[7];

        farmContractAddress = _addresses[8];
        pid = _pid;
        isCAKEStaking = _isCAKEStaking;
        isAutoComp = _isAutoComp;

        uniRouterAddress = _addresses[9];
        earnedToCMPPath = _earnedToCMPPath;
        earnedToToken0Path = _earnedToToken0Path;
        earnedToToken1Path = _earnedToToken1Path;
        token0ToEarnedPath = _token0ToEarnedPath;
        token1ToEarnedPath = _token1ToEarnedPath;
        earnedToWantPath = _earnedToWantPath;

        controllerFee = _controllerFee;
        rewardsAddress = _addresses[10];
        buyBackRate = _buyBackRate;
        buyBackAddress = _addresses[11];
        entranceFeeFactor = _entranceFeeFactor;
        withdrawFeeFactor = _withdrawFeeFactor;

        MDXAddress = _addresses[12];
        MDEXSwapMiningAddress = _addresses[13];
        MDXToEarnedPath = _MDXToEarnedPath;

        transferOwnership(autoFarmAddress);
    }

    // Claim trade mining rewards
    function claimTradeMiningReward() public {
        require(msg.sender == govAddress, "Not authorised");
        IMDEXSwapMining(MDEXSwapMiningAddress).takerWithdraw();
        _convertMDXToEarned();
    }

    function _convertMDXToEarned() internal {
        // Converts MDX (if any) to earned tokens
        uint256 MDXAmt = IERC20(MDXAddress).balanceOf(address(this));
        if (MDXAddress != earnedAddress && MDXAmt > 0) {
            IERC20(MDXAddress).safeIncreaseAllowance(uniRouterAddress, MDXAmt);
            // Swap all dust tokens to earned tokens
            _safeSwap(
                uniRouterAddress,
                MDXAmt,
                slippageFactor,
                MDXToEarnedPath,
                address(this),
                block.timestamp.add(600)
            );
        }
```

```
    }
}
```

Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already

deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Permission restrictions

- **Description:**
  Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other

users.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

armors.io

contact@armors.io