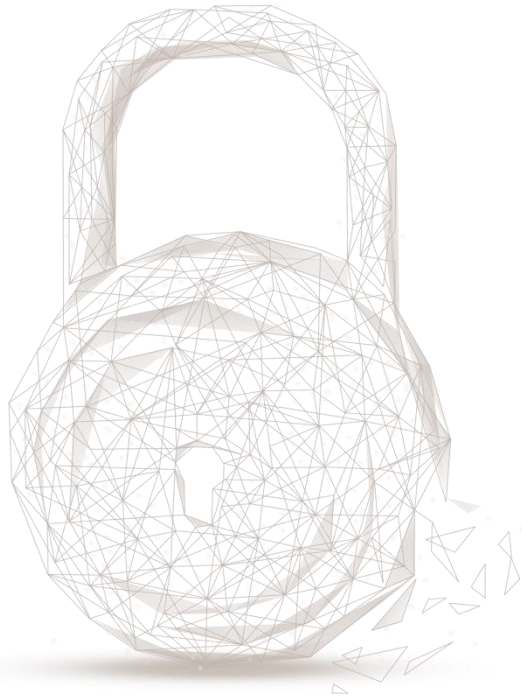




Smart contract security audit report



Audit Number: 202105111807

Smart Contract Name:

Bounce Token (Auction)

Smart Contract Address:

0xA9B1Eb5908CfC3cdf91F9B8B3a74108598009096

Smart Contract Address Link:

<https://etherscan.io/address/0xa9b1eb5908cfc3cdf91f9b8b3a74108598009096#code>

Start Date: 2021.05.11

Completion Date: 2021.05.11

Overall Result: Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	ERC20 Token Standards	Pass
		Compiler Version Security	Pass
		Visibility Specifiers	Pass
		Gas Consumption	Pass
		SafeMath Features	Pass
		Fallback Usage	Pass
		tx.origin Usage	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		Overriding Variables	Pass
2	Function Call Audit	Authorization of Function Call	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		selfdestruct Function Security	Pass

3	Business Security	Access Control of Owner	Pass
		Business Logics	Pass
		Business Implementations	Pass
4	Integer Overflow/Underflow	-	Pass
5	Reentrancy	-	Pass
6	Exceptional Reachable State	-	Pass
7	Transaction-Ordering Dependence	-	Pass
8	Block Properties Dependence	-	Pass
9	Pseudo-random Number Generator (PRNG)	-	Pass
10	DoS (Denial of Service)	-	Pass
11	Token Vesting Implementation	-	N/A
12	Fake Deposit	-	Pass
13	event security	-	Pass

Note: Audit results and suggestions in code comments

Disclaimer: This report is made in response to the project code. No description, expression or wording in this report shall be construed as an endorsement, affirmation or confirmation of the project. This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract Auction, including Coding Standards, Security, and Business Logic. **The Auction contract passes all audit items. The overall result is Pass. The smart contract is able to function properly.** Please find below the basic information of the smart contract:

1、 Basic Token Information

Token name	Bounce Token
Token symbol	Auction
decimals	18
totalSupply	Initially 0, mintable, the maximum limit is 10 million
Token type	ERC20

Table 1 – Basic Token Information

2、 Other function

The contract implements the *swap* function, and the user can use this function to exchange the specified token for the Bounce Token.

```

793     function swap(uint amountBot) external {
794         uint amountAuction = amountBot.mul(100);
795         ERC20(_botToken).transferFrom(msg.sender, DeadAddress, amountBot);
796         _mint(msg.sender, amountAuction);
797         emit Swapped(msg.sender, amountBot, amountAuction);
798     }
  
```

Figure 1 Source code of function *swap*

Audited Source Code with Comments:

```

/**
 *Submitted for verification at Etherscan.io on 2021-01-29
 */

// SPDX-License-Identifier: MIT

// File: @openzeppelin/contracts/GSN/Context.sol

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/*
 * @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 {
    // Beosin (Chengdu LianAn) // Internal function '_msgSender' for getting the caller address.
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }
    // Beosin (Chengdu LianAn) // Internal function '_msgData' for getting the transaction data.
    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

pragma solidity ^0.6.0;

/**
 * @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`.
     */
    // Beosin (Chengdu LianAn) // Declare the interface 'balanceOf'.
    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.
*/

// Beosin (Chengdu LianAn) // Declare the interface 'transfer'.
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.
 */

// Beosin (Chengdu LianAn) // Declare the interface 'allowance'.
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.
 */

// Beosin (Chengdu LianAn) // Declare the interface 'approve'.
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.
 */

// Beosin (Chengdu LianAn) // Declare the interface 'transferFrom'.
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.
*/
// Beosin (Chengdu LianAn) // Declare the event 'Transfer'.
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.
*/
// Beosin (Chengdu LianAn) // Declare the event 'Approval'.
event Approval(address indexed owner, address indexed spender, uint256 value);
}

// File: @openzeppelin/contracts/math/SafeMath.sol

pragma solidity ^0.6.0;

/**
* @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.
*/
// Beosin (Chengdu LianAn) // The SafeMath library declares these functions for safe mathematical
operations.
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.
 */
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.

*/

```
function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    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.
 */
```

```
function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b != 0, errorMessage);
    return a % b;
}
```

```
// File: @openzeppelin/contracts/utils/Address.sol
```

```
pragma solidity ^0.6.2;

/**
 * @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) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256("")`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @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-
interactions-pattern[checks-effects-interactions pattern].
*/

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.decode#abi-
encoding-and-decoding-functions[`abi.decode`].
 *
 * 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 returns
(bytes memory) {
    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 (bytes memory) {
    return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
}

/**
 * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[functionCallWithValue], but
 * 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 errorMessage) internal returns (bytes memory) {
    require(address(this).balance >= value, "Address: insufficient balance for call");
    return _functionCallWithValue(target, data, value, errorMessage);
}

function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory errorMessage) private returns (bytes memory) {
    require(isContract(target), "Address: call to non-contract");

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

// File: @openzeppelin/contracts/token/ERC20/ERC20.sol

```
pragma solidity ^0.6.0;
```

```
/**
```

```
 * @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; // Beosin (Chengdu LianAn) // Using the SafeMath library for
    mathematical operation. Avoid integer overflow.
```

```
using Address for address; // Beosin (Chengdu LianAn) // Using the Address library for address.
```

```
mapping (address => uint256) private _balances; // Beosin (Chengdu LianAn) // Mapping for storing token balance of corresponding address.
```

```
mapping (address => mapping (address => uint256)) private _allowances; // Beosin (Chengdu LianAn) // Mapping for storing the allowance between two addresses.
```

```
uint256 private _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable '_totalSupply' to store the token total supply.
```

```
string private _name; // Beosin (Chengdu LianAn) // Declare private variable '_name' to store the token name.
```

```
string private _symbol; // Beosin (Chengdu LianAn) // Declare private variable '_symbol' to store the token symbol.
```

```
uint8 private _decimals; // Beosin (Chengdu LianAn) // Declare private variable '_decimals' to store the token 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 returns (string memory) {  
    return _name; // Beosin (Chengdu LianAn) // Return the name of the token.  
}
```

```
/**
```

```
 * @dev Returns the symbol of the token, usually a shorter version of the  
 * name.
```

```
 */
```

```
function symbol() public view returns (string memory) {  
    return _symbol; // Beosin (Chengdu LianAn) // Return the symbol of the token.
```



```
}

/**
 * @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 returns (uint8) {
    return _decimals; // Beosin (Chengdu LianAn) // Return the decimals of the token.
}

/**
 * @dev See {IERC20-totalSupply}.
 */
function totalSupply() public view override returns (uint256) {
    return _totalSupply; // Beosin (Chengdu LianAn) // Return the total supply of the token.
}

/**
 * @dev See {IERC20-balanceOf}.
 */
function balanceOf(address account) public view override returns (uint256) {
    return _balances[account]; // Beosin (Chengdu LianAn) // Return the balance of the specified
address.
}

/**
 * @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); // Beosin (Chengdu LianAn) // Call the internal function
'_transfer' to transfer tokens.
    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.
 */
// Beosin (Chengdu LianAn) // 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.
// Beosin (Chengdu LianAn) // Using function 'increaseAllowance' and 'decreaseAllowance' to alter
allowance is recommended.
function approve(address spender, uint256 amount) public virtual override returns (bool) {
    _approve(_msgSender(), spender, amount); // Beosin (Chengdu LianAn) // Call the internal function
'_approve' to change the allowance.
    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 returns
(bool) {
    _transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function
'_transfer' to transfer tokens.
    _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
amount exceeds allowance")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to
decrease the allowance which 'sender' allowed to caller.
    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)); // Beosin
(Chengdu LianAn) // Call the internal function '_approve' to increase allowance.
    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, "ERC20:
decreased allowance below zero")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to
decrease allowance.
    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"); // Beosin (Chengdu LianAn) //
The non-zero address check for 'sender'.
    require(recipient != address(0), "ERC20: transfer to the zero address"); // Beosin (Chengdu LianAn) //
The non-zero address check for 'recipient'.

    _beforeTokenTransfer(sender, recipient, amount);

    _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance"); //
Beosin (Chengdu LianAn) // Update the 'sender' address token balance.
    _balances[recipient] = _balances[recipient].add(amount); // Beosin (Chengdu LianAn) // Update the
'recipient' address token balance.
    emit Transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
}

/** @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"); // Beosin (Chengdu LianAn) // The
non-zero address check for 'account'.

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

    _totalSupply = _totalSupply.add(amount); // Beosin (Chengdu LianAn) // Increase the total supply of
token.
    _balances[account] = _balances[account].add(amount); // Beosin (Chengdu LianAn) // Increase the
token balance of 'account'.
    emit Transfer(address(0), account, amount); // Beosin (Chengdu LianAn) // Trigger the event
'Transfer'.
}

```

```
/**
 * @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"); // Beosin (Chengdu LianAn) //
    The non-zero address check for 'account'.

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

    _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance"); //
    Beosin (Chengdu LianAn) // Decrease the token balance of 'account'.
    _totalSupply = _totalSupply.sub(amount); // Beosin (Chengdu LianAn) // Decrease the total supply of
    token.
    emit Transfer(account, address(0), amount); // Beosin (Chengdu LianAn) // Trigger the event
    'Transfer'.
}

/**
 * @dev Sets `amount` as the allowance of `spender` over the `owner`'s tokens.
 *
 * This is 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"); // Beosin (Chengdu LianAn) //
    The non-zero address check for 'owner'.
    require(spender != address(0), "ERC20: approve to the zero address"); // Beosin (Chengdu LianAn) //
    The non-zero address check for 'spender'.

    _allowances[owner][spender] = amount; // Beosin (Chengdu LianAn) // Set the allowance which
```

'owner' allowed to 'spender' as 'amount'.

```
emit Approval(owner, spender, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.  
}
```

```
/**
```

```
 * @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 {  
    _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 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 { }
```

```
// File: @openzeppelin/contracts/token/ERC20/ERC20Capped.sol
```

```
pragma solidity ^0.6.0;
```

```
/**
```

```
 * @dev Extension of {ERC20} that adds a cap to the supply of tokens.
```

```
 */
```

```
abstract contract ERC20Capped is ERC20 {
```

```
    uint256 private _cap; // Beosin (Chengdu LianAn) // Declare private variable '_cap' to store the cap of  
    token.
```

```
/**
 * @dev Sets the value of the `cap`. This value is immutable, it can only be
 * set once during construction.
 */
constructor (uint256 cap) public {
    require(cap > 0, "ERC20Capped: cap is 0");
    _cap = cap;
}

/**
 * @dev Returns the cap on the token's total supply.
 */
function cap() public view returns (uint256) {
    return _cap; // Beosin (Chengdu LianAn) // Return the cap of the token.
}

/**
 * @dev See {ERC20-_beforeTokenTransfer}.
 *
 * Requirements:
 *
 * - minted tokens must not cause the total supply to go over the cap.
 */
function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual override {
    super._beforeTokenTransfer(from, to, amount);

    if (from == address(0)) { // When minting tokens
        require(totalSupply().add(amount) <= _cap, "ERC20Capped: cap exceeded"); // Beosin (Chengdu
LianAn) // Check whether the minting limit is reached.
    }
}

// File: contracts/BounceAuctionToken.sol

pragma solidity ^0.6.0;

contract BounceAuctionToken is ERC20Capped {
    using SafeMath for uint; // Beosin (Chengdu LianAn) // Using the SafeMath library for mathematical
operation. Avoid integer overflow.

    address internal constant DeadAddress = 0x0000000000000000000000000000000000000000000000000000000000000000; // Beosin
(Chengdu LianAn) // Declare internal constant variable 'DeadAddress' to store the dead address.
```



```
address immutable private _botToken; // Beosin (Chengdu LianAn) // Declare private immutable variable '_botToken' to store the specific token address.
```

```
event Swapped(address indexed sender, uint amountBot, uint amountAuction); // Beosin (Chengdu LianAn) // Declare the event 'Swapped'.
```

```
constructor (address botToken) ERC20Capped(100000000e18) ERC20("Bounce Token", "Auction") public {  
    _botToken = botToken;  
}
```

```
function swap(uint amountBot) external {  
    uint amountAuction = amountBot.mul(100); // Beosin (Chengdu LianAn) // Calculate the number of tokens that can be exchanged.
```

```
    ERC20(_botToken).transferFrom(msg.sender, DeadAddress, amountBot); // Beosin (Chengdu LianAn) // Send tokens to the dead address.
```

```
    _mint(msg.sender, amountAuction); // Beosin (Chengdu LianAn) // The corresponding number of minted tokens to the caller.
```

```
    emit Swapped(msg.sender, amountBot, amountAuction); // Beosin (Chengdu LianAn) // Trigger the event 'Swapped'.  
}
```

```
}
```

```
// Beosin (Chengdu LianAn) // Recommend the main contract to inherit 'Pausable' module to grant owner the authority of pausing all transactions when serious issue occurred.
```



BEOSIN
Blockchain Security

Official Website

<https://lianantech.com>

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com