VATIN

Smart contract security audit report

Audit Number: 202112031330

Smart Contract Name: Haino(HE)

Smart Contract Address: 0xe5Da6365574512BD4B71537c7F5Fe95684AAa19D

Smart Contract Address Link:
https://bscscan.com/address/0xe5Da6365574512BD4B71537c7F5Fe95684AAa19D#code

Start Date: 20211201

Completion Date: 20211203

Overall Result: Pass

Audit Team: Vatin Audit(Singapore) Technology Co. Ltd

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	BEP20 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 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. Vatin Audit 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, Vatin Audit 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 Vatin Audit 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, Vatin Audit assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Vatin Audit is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Vatin Audit. Due to the technical limitations of any organization, this report conducted by Vatin Audit still has the possibility that the entire risk cannot be completely detected. Vatin disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Vatin.

Audit Results Explained:

Vatin audit has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract Haino, including Coding Standards, Security, and Business Logic. **Haino contract passed 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	Haino
Token symbol	HE
decimals	18
totalSupply	1,000,000,000,000
Token type	BEP20

Table 1 - Basic Token Information

2、Token Vesting Information

N/A

Audited Source Code with Comments:

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

import "../utils/Context.sol";
/**
 * @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.
 */
// VATIN Declare the contents of the ownable contract
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;
```

```
    }
}

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/**
 * @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.
 */
// VATIN Declare the mathematical operation library to prevent overflow of mathematical operation
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;
    }
}


// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
// VATIN Declare the contents of the ierc20 interface
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);
}


// SPDX-License-Identifier: MIT

pragma solidity >=0.6.2 <0.8.0;

/**
 * @dev Collection of functions related to the address type
 */
// VATIN Declare address type library
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-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");
        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 memory) {
        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) internal view
returns (bytes memory) {
        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) internal
returns (bytes memory) {
        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) private
```

```
pure returns (bytes memory) {
        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);
            }
        }
    }
}


// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/*
 * @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.
 */
// VATIN Claim context management contract
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.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}
```

```
// SPDX-License-Identifier: MIT

pragma solidity >= 0.6.0;

// VATIN Declare iemefactory contract interface
interface IEMEFactory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    function feeTo() external view returns (address);

    function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);

    function expectPairFor(address token0, address token1) external view returns (address);

    function allPairs(uint) external view returns (address pair);

    function allPairsLength() external view returns (uint);

    function createPair(address tokenA, address tokenB) external returns (address pair);

    function setFeeTo(address) external;

    function setFeeToSetter(address) external;
}


// SPDX-License-Identifier: MIT

pragma solidity >= 0.6.0;
// VATIN  Declare iemerouter contract interface
interface IEMERouter {
    function factory() external view returns (address);

    function WETH() external view returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
```

```
) external returns (uint amountA, uint amountB, uint liquidity);

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

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

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

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

function removeLiquidityETHWithPermit(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline,
```

```
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountToken, uint amountETH);

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

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

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

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

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

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

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

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

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

    function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memory
```

```
amounts);

    function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memory
amounts);

    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountETH);

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

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

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

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

```
// SPDX-License-Identifier: MIT


pragma solidity ^0.6.12;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/utils/Context.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "../interfaces/IEMEFactory.sol";
import "../interfaces/IEMERouter.sol";
// VATIN Implement token contract content
contract HEToken is Context, IERC20, Ownable {
    using SafeMath for uint256;
    using Address for address;

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

    mapping(address => bool) private _isExcludedFromFee;

    mapping(address => bool) private _isExcluded;
    address[] private _excluded;

    uint256 private constant MAX = ~uint256(0);
    uint256 private _tTotal = 1 * 10 ** 12 * 10 ** 18;
    uint256 private _rTotal = (MAX - (MAX % _tTotal));
    uint256 private _tFeeTotal;

    string private _name = "Haino";
    string private _symbol = "HE";
    uint8 private _decimals = 18;
    // VATIN Proportion of interest on currency
    uint256 public _taxFee = 5;
    uint256 private _previousTaxFee = _taxFee;
    // VATIN Proportion of automatic market making fee
    uint256 public _liquidityFee = 5;
    uint256 private _previousLiquidityFee = _liquidityFee;
    // VATIN Proportion of destruction fee
    uint256 public _burnFee = 0;
    uint256 private _previousBurnFee = _burnFee;

    IEMERouter public immutable uniswapV2Router;
    address public immutable uniswapV2Pair;
    address public constant hole = 0x000000000000000000000000000000000000dEaD;
```

```
    bool inSwapAndLiquify;
    bool public swapAndLiquifyEnabled = true;

    uint256 public _maxTxAmount = 1 * 10 ** 12 * 10 ** 18;
    uint256 private numTokensSellToAddToLiquidity = 1 * 10 ** 9 * 10 ** 18;

    event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
    event SwapAndLiquifyEnabledUpdated(bool enabled);
    event SwapAndLiquify(
        uint256 tokensSwapped,
        uint256 ethReceived,
        uint256 tokensIntoLiqudity
    );

    modifier lockTheSwap {
        inSwapAndLiquify = true;
        _;
        inSwapAndLiquify = false;
    }
    // VATIN Constructor to determine the specific parameters of token contract
    constructor () public {
        _rOwned[_msgSender()] = _rTotal;

        IEMERouter _uniswapV2Router =
IEMERouter(0x10ED43C718714eb63d5aA57B78B54704E256024E);
        // Create a uniswap pair for this new token
        uniswapV2Pair = IEMEFactory(_uniswapV2Router.factory())
        .createPair(address(this), _uniswapV2Router.WETH());

        // set the rest of the contract variables
        uniswapV2Router = _uniswapV2Router;

        //exclude owner and this contract from fee
        _isExcludedFromFee[owner()] = true;
        _isExcludedFromFee[address(this)] = true;
        _isExcludedFromFee[hole] = true;

        emit Transfer(address(0), _msgSender(), _tTotal);
    }

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }
```

```
    function decimals() public view returns (uint8) {
        return _decimals;
    }

    function totalSupply() public view override returns (uint256) {
        return _tTotal;
    }

    function balanceOf(address account) public view override returns (uint256) {
        if (_isExcluded[account]) return _tOwned[account];
        return tokenFromReflection(_rOwned[account]);
    }

    function transfer(address recipient, uint256 amount) public override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    function allowance(address owner, address spender) public view override returns (uint256) {
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
amount exceeds allowance"));
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20:
decreased allowance below zero"));
        return true;
    }

    function isExcludedFromReward(address account) public view returns (bool) {
        return _isExcluded[account];
    }
```

```
    function totalFees() public view returns (uint256) {
        return _tFeeTotal;
    }
    // VATIN Destroy sender tokens and contribute to other token owners
    function deliver(uint256 tAmount) public {
        address sender = _msgSender();
        require(!_isExcluded[sender], "Excluded addresses cannot call this function");
        (uint256 rAmount,,,,,,) = _getValues(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rTotal = _rTotal.sub(rAmount);
        _tFeeTotal = _tFeeTotal.add(tAmount);
    }
    function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view returns (uint256) {
        require(tAmount <= _tTotal, "Amount must be less than supply");
        if (!deductTransferFee) {
            (uint256 rAmount,,,,,,) = _getValues(tAmount);
            return rAmount;
        } else {
            (,uint256 rTransferAmount,,,,,) = _getValues(tAmount);
            return rTransferAmount;
        }
    }

    // VATIN Tamount mapping ramount relationship
    function tokenFromReflection(uint256 rAmount) public view returns (uint256) {
        require(rAmount <= _rTotal, "Amount must be less than total reflections");
        uint256 currentRate = _getRate();
        return rAmount.div(currentRate);
    }
    // VATIN Cancel the account to receive interest, and only the owner calls
    function excludeFromReward(address account) public onlyOwner() {
        // require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can not exclude
Uniswap router.');
        require(!_isExcluded[account], "Account is already excluded");
        if (_rOwned[account] > 0) {
            _tOwned[account] = tokenFromReflection(_rOwned[account]);
        }
        _isExcluded[account] = true;
        _excluded.push(account);
    }

    // VATIN Set account to receive interest. Only the owner calls
    function includeInReward(address account) external onlyOwner() {
        require(_isExcluded[account], "Account is already excluded");
        for (uint256 i = 0; i < _excluded.length; i++) {
            if (_excluded[i] == account) {
                _excluded[i] = _excluded[_excluded.length - 1];
```

```
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}

function _transferBothExcluded(address sender, address recipient, uint256 tAmount) private {
    (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee,
uint256 tLiquidity,uint256 tBurn) = _getValues(tAmount);
    _tOwned[sender] = _tOwned[sender].sub(tAmount);
    _rOwned[sender] = _rOwned[sender].sub(rAmount);
    _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
    _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
    _takeLiquidity(tLiquidity);
    _takeBurn(tBurn);
    _reflectFee(rFee, tFee);
    emit Transfer(sender, recipient, tTransferAmount);
}
// VATIN Cancel the account fee free white list, which is only called by the owner
function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}

// VATIN Set the account fee free white list, which can only be called by the owner
function includeInFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = false;
}

// VATIN Set the interest handling fee proportion, which can only be called by the owner
function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
    _taxFee = taxFee;
}

// VATIN Set the handling charge proportion of automatic market making, which can only be called
by the owner
function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {
    _liquidityFee = liquidityFee;
}

// VATIN Set the destruction fee proportion, which can only be called by the owner
function setBurnFeePercent(uint256 burnFee) external onlyOwner() {
    _burnFee = burnFee;
}


// VATIN Set the maximum transfer quantity, which can only be called by the owner
```

```
    function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
        _maxTxAmount = _tTotal.mul(maxTxPercent).div(
            10 ** 2
        );
    }

    // VATIN Whether to enable automatic market making, only called by the owner
    function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
        swapAndLiquifyEnabled = _enabled;
        emit SwapAndLiquifyEnabledUpdated(_enabled);
    }

    //to recieve ETH from uniswapV2Router when swaping
    //receive() external payable {}

    function _reflectFee(uint256 rFee, uint256 tFee) private {
        _rTotal = _rTotal.sub(rFee);
        _tFeeTotal = _tFeeTotal.add(tFee);
    }

    function _getValues(uint256 tAmount) private view returns (uint256, uint256, uint256, uint256, uint256,
uint256, uint256) {
        (uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity,uint256 tBurn) = _getTValues(tAmount);
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = _getRValues(tAmount, tFee, tLiquidity,
tBurn, _getRate());
        return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee, tLiquidity, tBurn);
    }

    function _getTValues(uint256 tAmount) private view returns (uint256, uint256, uint256, uint256) {
        uint256 tFee = calculateTaxFee(tAmount);
        uint256 tLiquidity = calculateLiquidityFee(tAmount);
        uint256 tBurn = calculateBurnFee(tAmount);
        uint256 tTransferAmount = tAmount.sub(tFee).sub(tLiquidity).sub(tBurn);
        return (tTransferAmount, tFee, tLiquidity, tBurn);
    }

    function _getRValues(uint256 tAmount, uint256 tFee, uint256 tLiquidity, uint256 tBurn, uint256
currentRate) private pure returns (uint256, uint256, uint256) {
        uint256 rAmount = tAmount.mul(currentRate);
        uint256 rFee = tFee.mul(currentRate);
        uint256 rLiquidity = tLiquidity.mul(currentRate);
        uint256 rBurn = tBurn.mul(currentRate);
        uint256 rTransferAmount = rAmount.sub(rFee).sub(rLiquidity).sub(rBurn);
        return (rAmount, rTransferAmount, rFee);
    }

    function _getRate() private view returns (uint256) {
        (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
```

```
    return rSupply.div(tSupply);
}

function _getCurrentSupply() private view returns (uint256, uint256) {
    uint256 rSupply = _rTotal;
    uint256 tSupply = _tTotal;
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
        rSupply = rSupply.sub(_rOwned[_excluded[i]]);
        tSupply = tSupply.sub(_tOwned[_excluded[i]]);
    }
    if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
    return (rSupply, tSupply);
}

function _takeLiquidity(uint256 tLiquidity) private {
    uint256 currentRate = _getRate();
    uint256 rLiquidity = tLiquidity.mul(currentRate);
    _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity);
    if (_isExcluded[address(this)])
        _tOwned[address(this)] = _tOwned[address(this)].add(tLiquidity);
}

function _takeBurn(uint256 tBurn) private {
    uint256 currentRate = _getRate();
    uint256 rBurn = tBurn.mul(currentRate);
    _rOwned[hole] = _rOwned[hole].add(rBurn);
    if (_isExcluded[hole])
        _tOwned[hole] = _tOwned[hole].add(tBurn);
}

function calculateTaxFee(uint256 _amount) private view returns (uint256) {
    return _amount.mul(_taxFee).div(
        10 ** 2
    );
}

function calculateLiquidityFee(uint256 _amount) private view returns (uint256) {
    return _amount.mul(_liquidityFee).div(
        10 ** 2
    );
}

function calculateBurnFee(uint256 _amount) private view returns (uint256) {
    return _amount.mul(_burnFee).div(
        10 ** 2
    );
}
```

```
function removeAllFee() private {
    if (_taxFee == 0 && _liquidityFee == 0 && _burnFee == 0) return;

    _previousTaxFee = _taxFee;
    _previousLiquidityFee = _liquidityFee;
    _previousBurnFee = _burnFee;
    _taxFee = 0;
    _liquidityFee = 0;
    _burnFee = 0;
}

function restoreAllFee() private {
    _taxFee = _previousTaxFee;
    _liquidityFee = _previousLiquidityFee;
    _burnFee = _previousBurnFee;
}

function isExcludedFromFee(address account) public view returns (bool) {
    return _isExcludedFromFee[account];
}

function _approve(address owner, address spender, uint256 amount) private {
    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);
}
// VATIN Token transfer function
function _transfer(
    address from,
    address to,
    uint256 amount
) private {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");
    require(amount > 0, "Transfer amount must be greater than zero");
    if (from != owner() && to != owner())
        require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");

    // is the token balance of this contract address over the min number of
    // tokens that we need to initiate a swap + liquidity lock?
    // also, don't get caught in a circular liquidity event.
    // also, don't swap & liquify if sender is uniswap pair.
    uint256 contractTokenBalance = balanceOf(address(this));

    if (contractTokenBalance >= _maxTxAmount)
```

```
    {
        contractTokenBalance = _maxTxAmount;
    }

    bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
    // VATIN Automatic market making function
    if (
        overMinTokenBalance &&
        !inSwapAndLiquify &&
        from != uniswapV2Pair &&
        swapAndLiquifyEnabled
    ) {
        contractTokenBalance = numTokensSellToAddToLiquidity;
        //add liquidity
        swapAndLiquify(contractTokenBalance);
    }

    //indicates if fee should be deducted from transfer
    bool takeFee = true;

    //if any account belongs to _isExcludedFromFee account then remove the fee
    if (_isExcludedFromFee[from] || _isExcludedFromFee[to]) {
        takeFee = false;
    }

    //transfer amount, it will take tax, burn, liquidity fee
    // VATIN Execute the corresponding transfer function according to takefee
    _tokenTransfer(from, to, amount, takeFee);
}
// VATIN Automatic exchange and market making function
function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
    // split the contract balance into halves
    uint256 half = contractTokenBalance.div(2);
    uint256 otherHalf = contractTokenBalance.sub(half);

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

    // swap tokens for ETH
    swapTokensForEth(half);
    // <- this breaks the ETH -> HATE swap when swap+liquify is triggered

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

```
    // add liquidity to uniswap
    addLiquidity(otherHalf, newBalance);

    emit SwapAndLiquify(half, newBalance, otherHalf);
}
// VATIN Convert token to BNB function
function swapTokensForEth(uint256 tokenAmount) private {
    // generate the uniswap pair path of token -> weth
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();

    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // make the swap
    uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0, // accept any amount of ETH
        path,
        address(this),
        block.timestamp
    );
}
// VATIN Add liquidity function
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // add the liquidity
    uniswapV2Router.addLiquidityETH{value : ethAmount}(
address(this),
tokenAmount,
0, // slippage is unavoidable
0, // slippage is unavoidable
owner(),
block.timestamp
);
}

//this method is responsible for taking all fee, if takeFee is true
function _tokenTransfer(address sender, address recipient, uint256 amount, bool takeFee) private {
    if (!takeFee)
        removeAllFee();

    if (_isExcluded[sender] && !_isExcluded[recipient]) {
        _transferFromExcluded(sender, recipient, amount);
    } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
        _transferToExcluded(sender, recipient, amount);
```

```
        } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
            _transferStandard(sender, recipient, amount);
        } else if (_isExcluded[sender] && _isExcluded[recipient]) {
            _transferBothExcluded(sender, recipient, amount);
        } else {
            _transferStandard(sender, recipient, amount);
        }

        if (!takeFee)
            restoreAllFee();
    }

    function _transferStandard(address sender, address recipient, uint256 tAmount) private {
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee,
uint256 tLiquidity, uint256 tBurn) = _getValues(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _takeBurn(tBurn);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _transferToExcluded(address sender, address recipient, uint256 tAmount) private {
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee,
uint256 tLiquidity, uint256 tBurn) = _getValues(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _takeBurn(tBurn);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private {
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee,
uint256 tLiquidity, uint256 tBurn) = _getValues(tAmount);
        _tOwned[sender] = _tOwned[sender].sub(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _takeBurn(tBurn);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }

}
```

Appendix:safety risk rating criteria

Vulnerability rating	Vulnerability rating description
High risk	Loopholes that can directly cause the loss of token contracts or users' funds, such as value overflow loopholes that can cause the value of tokens to return to zero, false recharge loopholes that can cause the loss of tokens in the exchange, and reentry loopholes that can cause the loss of assets or tokens in the contract account; Vulnerabilities that can cause the loss of ownership of token contracts, such as access control defects of key functions, bypassing access control of key functions caused by call injection, etc; A vulnerability that can cause token contracts to fail to work properly.
Medium risk	High risk vulnerabilities that require a specific address to trigger, such as value overflow vulnerabilities that can be triggered by token contract owners; Access control defects of non key functions, logic design defects that can not cause direct capital loss, etc.
Low risk	Vulnerabilities that are difficult to trigger, vulnerabilities that do limited harm after triggering, such as value overflow vulnerabilities that require a large number of tokens to trigger, vulnerabilities that the attacker cannot make direct profits after triggering value overflow, transaction sequence dependency risk triggered by specifying high mining fee, etc.

VATIN

Official website

https://vatin.io

E-mail

support@vatin.io