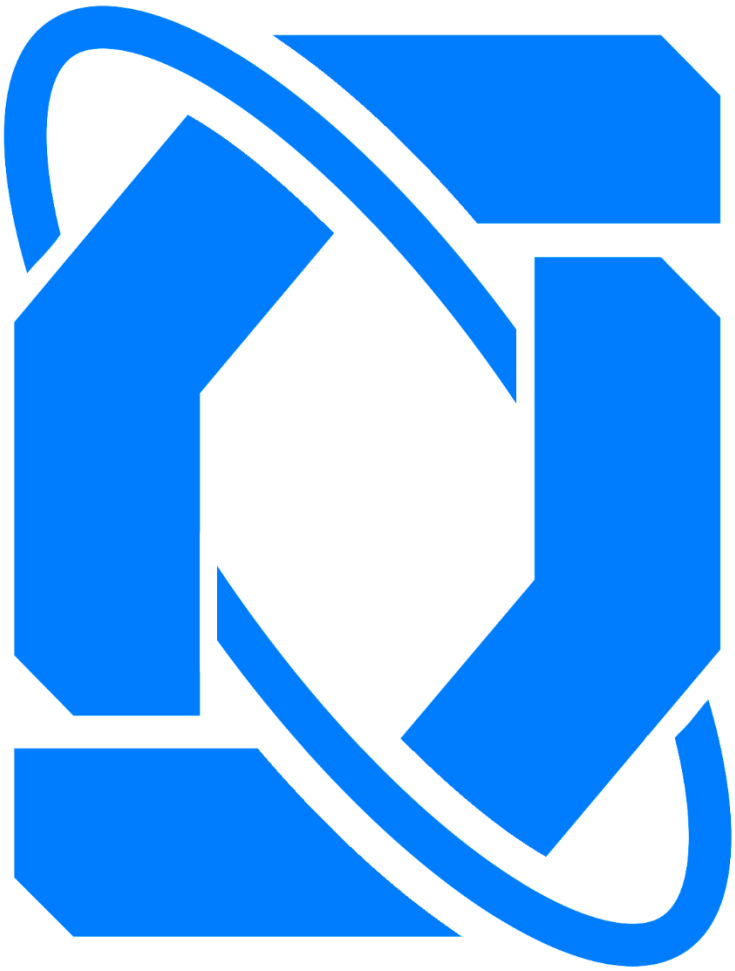


Orbital Station Excelsior Smart Contract Security Audit

Contents

Smart Contract & EVM Compiler Overview 3
EVM Compiler Version Bugs Info 4
Code Overview 5
Analysis: Basic Coding Bugs 15
Analysis: Smart Contract Functions Scheme 17
Analysis: Smart Contract Vulnerability Check 18
Summary 19
Disclaimer 20
About Orbital Station Excelsior (OSEX) 22
Contacts and links 23
Report 24

Smart Contract Security Audit Start Point

Smart Contract & EVM Compiler Overview

Blockchain: Binance Smart Chain (BSC)

Address: 0x52ABb09Ceb590f04c7d70bE9d6aA420177E94744

Explorer Link: https://bscscan.com/address/0x52abb09ceb590f04c7d70be9d6aa420177e94744

Token Name: BNBDog

Verified: Yes

Status: Deployed

Symbol: (BNBDOG)

Contract Name: BNBDOGToken

Decimals: 18

Total Supply: 1940895167788587613.280391380278395040

Deployment Date: Apr-02-2023 07:07:44 PM +UTC

Compiler Version: v0.8.9+commit.e5eed63a

License Type: MIT license

Optimization: 200 runs

Compiler Settings: default evmVersion, MIT license

Solidity Compiler Bugs for this version:

- AbiReencodingHeadOverflowWithStaticArrayCleanup (medium-severity)
- DirtyBytesArrayToStorage (low-severity)
- DataLocationChangeInInternalOverride (very low-severity)
- NestedCalldataArrayAbiReencodingSizeValidation (very low-severity)

Code language: Solidity

Creator: 0x8e3d11a41d20fff2d5b577ad7327112f2406c289 at txn 0xfa51b595ebac4d783c4d1721ce05b1db86f17c2d4bde4e12f114e34daebc6ca4

Date of audit: 2023/11

EVM Compiler Version Bugs Info

Bug Name: AbiReencodingHeadOverflowWithStaticArrayCleanup
Description: When ABI-encoding a statically-sized calldata array, the compiler always pads the data area to a multiple of 32-bytes and ensures that the padding bytes are zeroed. In some cases, this cleanup used to be performed by always writing exactly 32 bytes, regardless of how many needed to be zeroed. This was done with the assumption that the data that would eventually occupy the area past the end of the array had not yet been written, because the encoder processes tuple components in the order they were given. While this assumption is mostly true, there is an important corner case: dynamically encoded tuple components are stored separately from the statically-sized ones in an area called the *tail* of the encoding and the tail immediately follows the *head*, which is where the statically-sized components are placed. The aforementioned cleanup, if performed for the last component of the head would cross into the tail and overwrite up to 32 bytes of the first component stored there with zeros. The only array type for which the cleanup could actually result in an overwrite were arrays with ``uint256`` or ``bytes32`` as the base element type and in this case the size of the corrupted area was always exactly 32 bytes. The problem affected tuples at any nesting level. This included also structs, which are encoded as tuples in the ABI. Note also that lists of parameters and return values of functions, events and errors are encoded as tuples.

Bug Name: DirtyBytesArrayToStorage
Description: Copying ``bytes`` arrays from memory or calldata to storage is done in chunks of 32 bytes even if the length is not a multiple of 32. Thereby, extra bytes past the end of the array may be copied from calldata or memory to storage. These dirty bytes may then become observable after a ``.push()`` without arguments to the bytes array in storage, i.e. such a push will not result in a zero value at the end of the array as expected. This bug only affects the legacy code generation pipeline, the new code generation pipeline via IR is not affected.

Bug Name: DataLocationChangeInInternalOverride
Description: When calling external functions, it is irrelevant if the data location of the parameters is ``calldata`` or ``memory``, the encoding of the data does not change. Because of that, changing the data location when overriding external functions is allowed. The compiler incorrectly also allowed a change in the data location for overriding public and internal functions. Since public functions can be called internally as well as externally, this causes invalid code to be generated when such an incorrectly overridden function is called internally through the base contract. The caller provides a memory pointer, but the called function interprets it as a calldata pointer or vice-versa.

Bug Name: NestedCalldataArrayAbiReencodingSizeValidation
Description: Calldata validation for nested dynamic types is deferred until the first access to the nested values. Such an access may for example be a copy to memory or an index or member access to the outer type. While in most such accesses calldata validation correctly checks that the data area of the nested array is completely contained in the passed calldata (i.e. in the range [0, calldatasize()]), this check may not be performed, when ABI encoding such nested types again directly from calldata. For instance, this can happen, if a value in calldata with a nested dynamic array is passed to an external call, used in ``abi.encode`` or emitted as event. In such cases, if the data area of the nested array extends beyond ``calldatasize()``, ABI encoding it did not revert, but continued reading values from beyond ``calldatasize()`` (i.e. zero values).

Code Overview

```
/**
 *Submitted for verification at BscScan.com on 2023-04-02
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.9;

// CAUTION
// This version of SafeMath should only be used with Solidity 0.8 or later,
// because it relies on the compiler's built in overflow checks.

/**
 * @dev Wrappers over Solidity's arithmetic operations.
 *
 * NOTE: `SafeMath` is generally not needed starting with Solidity 0.8, since the compiler
 * now has built in overflow checking.
 */
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) {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
            return (true, c);
        }
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            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) {
        unchecked {
            // 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) {
        unchecked {
            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) {
        unchecked {
            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) {
        return a + b;
    }

    /**
     * @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 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) {
        return a * b;
    }

    /**
     * @dev Returns the integer division of two unsigned inte-
gers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator.
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns
(uint256) {
        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) {
        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 re-
quires 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 er-
rorMessage) internal pure returns (uint256) {
        unchecked {
            require(b <= a, errorMessage);
            return a - b;
        }
    }

    /**
     * @dev Returns the integer division of two unsigned inte-
gers, reverting 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 remain-
ing gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory er-
rorMessage) internal pure returns (uint256) {
        unchecked {
            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 re-
quires 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 er-
rorMessage) internal pure returns (uint256) {
        unchecked {
            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 fol-
lowing
     * 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
     *
     * Furthermore, `isContract` will also return true if the target contract within
     * the same transaction is already scheduled for destruction by `SELFDESTRUCT`,
     * which only has an effect at the end of a transaction.
     * ====
     *
     * [IMPORTANT]
     * ====
     * You shouldn't rely on `isContract` to protect against flash loan attacks!
     *
     * Preventing calls from contracts is highly discouraged. It breaks composability, breaks support for smart wallets
     * like Gnosis Safe, and does not provide security since it can be circumvented by calling from a contract
     * constructor.
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize/address.code.length, which returns 0
        // for contracts in construction, since the code is only stored at the end
        // of the constructor execution.

        return account.code.length > 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://consensys.net/diligence/blog/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");

        (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 functionCallWithValue(target, data, 0, "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");
        (bool success, bytes memory returndata) = target.call{value: value}(data);
        return verifyCallResultFromTarget(target, 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) {
        (bool success, bytes memory returndata) = target.staticcall(data);
        return verifyCallResultFromTarget(target, 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) {
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return verifyCallResultFromTarget(target, success, returndata, errorMessage);
    }

    /**
     * @dev Tool to verify that a low level call to smart-contract was successful, and revert (either by bubbling
     * the revert reason or using the provided one) in case of unsuccessful call or if target was not a contract.
     *
     * _Available since v4.8._
     */
    function verifyCallResultFromTarget(
        address target,
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) internal view returns (bytes memory) {
        if (success) {
            if (returndata.length == 0) {
                // only check isContract if the call was successful and the return data is empty
                // otherwise we already know that it was a contract
                require(isContract(target), "Address: call to non-contract");
            }
            return returndata;
        } else {
            _revert(returndata, errorMessage);
        }
    }

    /**
     * @dev Tool to verify that a low level call was successful, and revert if it wasn't, either by bubbling the
     * revert reason or using the provided one.
     *
     * _Available since v4.3._
     */
    function verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) internal pure returns (bytes memory) {
        if (success) {
            return returndata;
        } else {
            _revert(returndata, errorMessage);
        }
    }

    function _revert(bytes memory returndata, string memory errorMessage) private pure {
        // 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
            /// @solidity memory-safe-assembly
            assembly {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
```

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


/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

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

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


/**
 * @dev Interface for the optional metadata functions from the ERC20 standard.
 *
 * _Available since v4.1._
 */
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);

    /**
     * @dev Returns the symbol of the token.
     */
    function symbol() external view returns (string memory);

    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view returns (uint8);
}


/**
 * @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 meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
}


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

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor() {
        _transferOwnership(_msgSender());
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        _checkOwner();
        _;
    }

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

    /**
     * @dev Throws if the sender is not the owner.
     */
    function _checkOwner() internal view virtual {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby disabling any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        _transferOwnership(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");
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Internal function without access restriction.
     */
    function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}

/**
 * @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.openzeppelin.com/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * The default value of {decimals} is 18. To change this, you should override
 * this function so it returns a different value.
 *
 * We have followed general OpenZeppelin Contracts guidelines: functions revert
```

```
 * instead 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, IERC20Metadata {

    using SafeMath for uint256;
    using Address for address;

    mapping(address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * All two of these values are immutable: they can only be set once during
     * construction.
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

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

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

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5.05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the default value returned by this function, unless
     * it's overridden.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view virtual override returns (uint8) {
        return 18;
    }

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

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

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

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

    /**
     * @dev See {IERC20-approve}.
     *
     * NOTE: If `amount` is the maximum `uint256`, the allowance is not updated on
     * `transferFrom`. This is semantically equivalent to an infinite approval.
     *
     * Requirements:
     *
```

```
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * NOTE: Does not update the allowance if the current allowance
     * is the maximum `uint256`.
     *
     * Requirements:
     *
     * - `from` and `to` cannot be the zero address.
     * - `from` must have a balance of at least `amount`.
     * - the caller must have allowance for ``from``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address from, address to, uint256 amount) public virtual override returns (bool) {
        address spender = _msgSender();
        _spendAllowance(from, spender, amount);
        _transfer(from, to, amount);
        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) {
        address owner = _msgSender();
        _approve(owner, spender, allowance(owner, spender) + addedValue);
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        address owner = _msgSender();
        uint256 currentAllowance = allowance(owner, spender);
        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
        unchecked {
            _approve(owner, spender, currentAllowance - subtractedValue);
        }

        return true;
    }

    /**
     * @dev Moves `amount` of tokens from `from` to `to`.
     *
     * This 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:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `from` must have a balance of at least `amount`.
     */
    function _transfer(address from, address to, uint256 amount) internal virtual {
        require(from != address(0), "ERC20: transfer from the zero address");
        require(to != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(from, to, amount);

        uint256 fromBalance = _balances[from];
        require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
        unchecked {
            _balances[from] = fromBalance - amount;
            // Overflow not possible: the sum of all balances is capped by totalSupply, and the sum is preserved by
            // decrementing then incrementing.
            _balances[to] += amount;
        }

        emit Transfer(from, to, amount);

        _afterTokenTransfer(from, to, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements:
```

```
     *
     * - `account` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal
virtual {
        require(account != address(0), "ERC20: mint to the zero
address");

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

        _totalSupply += amount;
        unchecked {
            // Overflow not possible: balance + amount is at most
totalSupply + amount, which is checked above.
            _balances[account] += amount;
        }
        emit Transfer(address(0), account, amount);

        _afterTokenTransfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reduc-
ing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero ad-
dress.
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal
virtual {
        require(account != address(0), "ERC20: burn from the
zero address");

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

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn
amount exceeds balance");
        unchecked {
            _balances[account] = accountBalance - amount;
            // Overflow not possible: amount <= accountBalance
<= totalSupply.
            _totalSupply -= amount;
        }

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }

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

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

    /**
     * @dev Updates `owner` s allowance for `spender` based
on spent `amount`.
     *
     * Does not update the allowance amount in case of infinite
allowance.
     * Revert if not enough allowance is available.
     *
     * Might emit an {Approval} event.
     */
    function _spendAllowance(address owner, address
spender, uint256 amount) internal virtual {
        uint256 currentAllowance = allowance(owner, spender);
        if (currentAllowance != type(uint256).max) {
            require(currentAllowance >= amount, "ERC20: insuf-
ficient allowance");
            unchecked {
                _approve(owner, spender, currentAllowance -
amount);
            }
        }
    }

    /**
     * @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:extend-
ing-contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(address from, address to,
uint256 amount) internal virtual {}

    /**
     * @dev Hook that is called after any transfer of tokens.
This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of
``from``'s tokens
     * has been transferred to `to`.
     * - when `from` is zero, `amount` tokens have been
minted for `to`.
```

```
    * - when `to` is zero, `amount` of ``from``'s tokens have been 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 _afterTokenTransfer(address from, address to, uint256 amount) internal virtual {}
}


contract BNBDOGToken is ERC20, Ownable {

    constructor() ERC20("BNBDog", "BNBDOG") {
        _mint(msg.sender, 2000000000000000000 * 10 ** decimals());
    }

    uint256 public transferBurnRate = 3;

    mapping(address => bool) private _transferBurnExceptAddresses;

    /**
    * @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 override {
        if (transferBurnRate > 0 && _transferBurnExceptAddresses[sender] != true && _transferBurnExceptAddresses[recipient] != true && recipient != address(0)) {
            uint256 _burntAmount = amount * transferBurnRate / 100;
            // Burn transferBurnRate% from amount
            super._burn(sender, _burntAmount);
            // Recalibrate the transfer amount
            amount = amount - _burntAmount;
        }

        super._transfer(sender, recipient, amount);
    }

    /**
    * @dev Destroys `amount` tokens from the caller.
    *
    * See {ERC20-_burn}.
    */
    function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount);
    }

    /**
    * @dev Destroys `amount` tokens from `account`, deducting from the caller's
    * allowance.
    *
    * See {ERC20-_burn} and {ERC20-allowance}.
    *
    * Requirements:
    *
    * - the caller must have allowance for ``accounts``'s tokens of at least
    * `amount`.
    */
    function burnFrom(address account, uint256 amount) public virtual {
        _spendAllowance(account, _msgSender(), amount);
        _burn(account, amount);
    }

    function addTransferBurnExceptAddress(address _transferBurnExceptAddress) public onlyOwner {
        require(_transferBurnExceptAddress != address(0), "cannot be the zero address");
        _transferBurnExceptAddresses[_transferBurnExceptAddress] = true;
    }

    function removeTransferBurnExceptAddress(address _transferBurnExceptAddress) public onlyOwner {
        require(_transferBurnExceptAddress != address(0), "cannot be the zero address");
        delete _transferBurnExceptAddresses[_transferBurnExceptAddress];
    }
}
```

Analysis: Basic Coding Bugs

1. Semantic Consistency Checks
- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: Not found
- Severity: Critical

2. Redundant Fallback Function
- Description: Whether the contract has a redundant fallback function.
- Result: Not found
- Severity: Critical

3. Constructor Mismatch
- Description: Whether the contract name and its constructor are not identical to each other.
- Result: Not found
- Severity: Critical

4. Ownership Takeover
- Description: Whether the set owner function is not protected.
- Result: Not found
- Severity: Critical

5. Overflows & Underflows
- Description: Whether the contract has general overflow o underflow vulnerabilities.
- Result: Not found
- Severity: Critical

6. Reentrancy
- Description: Reentrancy is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
- Result: Not found
- Severity: Critical

7. Blackhole
- Description: Whether the contract locks ETH indefinitely: merely in without out.
- Result: Not found
- Severity: High

8. Money-Giving Bug
- Description: Whether the contract returns funds to an arbitrary address.
- Result: Not found
- Severity: High

9. Unauthorized Self-Destruct
- Description: Whether the contract can be killed by any arbitrary address.
- Result: Not found
- Severity: Medium

10. Unchecked External Call
- Description: Whether the contract has any external call without checking the return value.
- Result: Not found
- Severity: Medium

11. Revert DoS
- Description: Whether the contract is vulnerable to DoS attack because of unexpected revert.
- Result: Not found
- Severity: Medium

12. Gasless Send
- Description: Whether the contract is vulnerable to gasless send.
- Result: Not found
- Severity: Medium

13. Send Instead Of Transfer
- Description: Whether the contract uses send instead of transfer.
- Result: Not found
- Severity: Medium

14. Use Of Untrusted Libraries
- Description: Whether the contract use any suspicious libraries.
- Result: Not found
- Severity: Medium

15. Costly Loop
- Description: Whether the contract has any costly loop which may lead to Out-Of-Gas exception.
- Result: Not found
- Severity: Medium

16. Use Of Predictable Variables
- Description: Whether the contract contains any randomness variable, but its value can be predicated.
- Result: Not found
- Severity: Medium

17. Deprecated Uses
- Description: Whether the contract use the deprecated Tx. Origin to perform the authorization.
- Result: Not found
- Severity: Medium

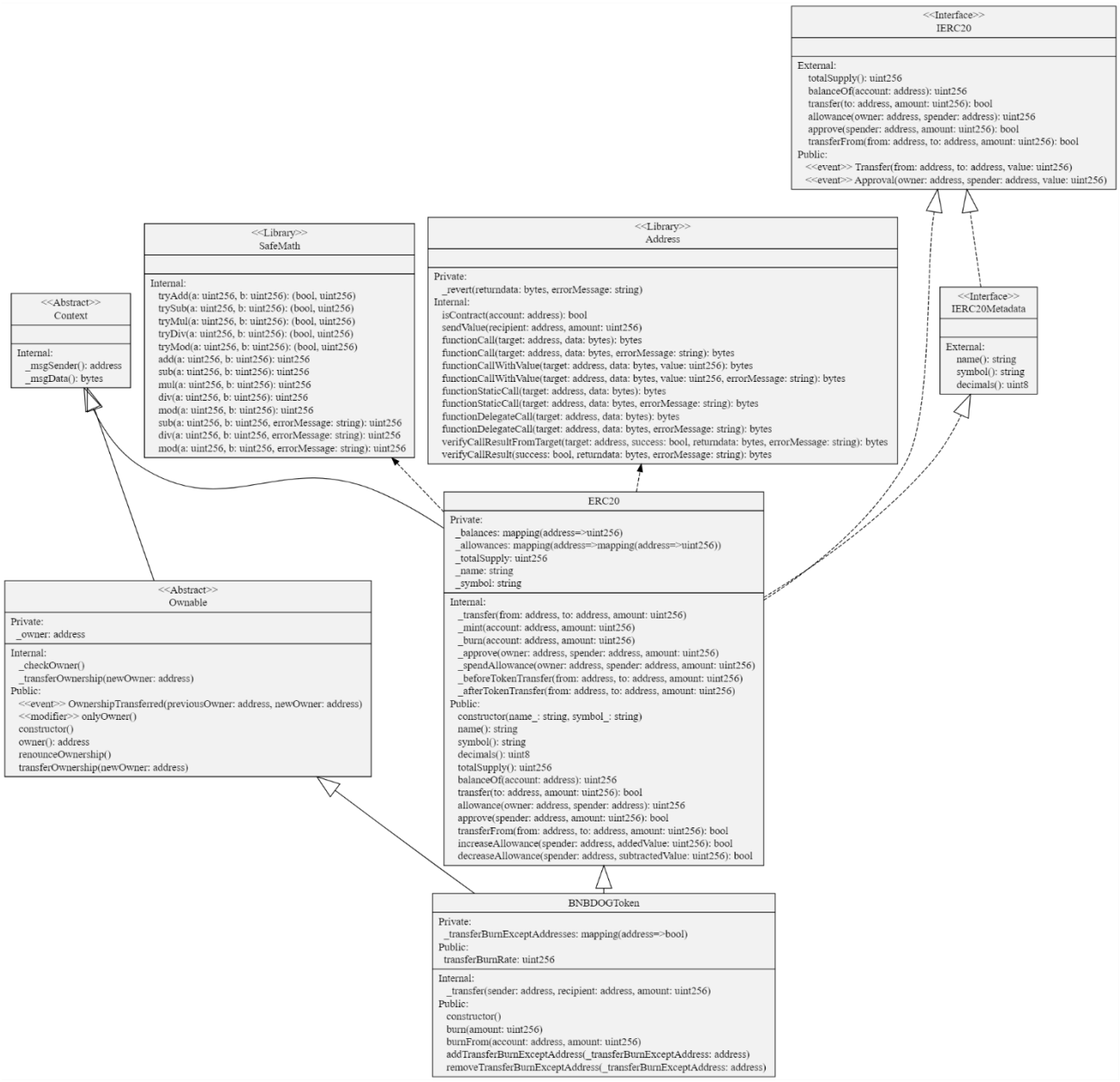
18. Transaction Ordering Dependence
- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Not found
- Severity: Medium

19. Make Type Inference Explicit
- Description: Do not use keyword var to specify the type, i.e., it asks the compiler to deduce the type, which is not safe especially in a loop.
- Result: Not found
- Severity: Low

20. Make Visibility Level Explicit
- Description: Assign explicit visibility specifiers for functions and state variables.
- Result: Not found
- Severity: Low

21. Avoid Use of Variadic Byte Array
- Description: Use fixed-size byte array is better than that of byte, a the latter is a waste of space.
- Result: Not found
- Severity: Low

Analysis: Smart Contract Functions Scheme



Analysis: Smart Contract Vulnerability Check

Fallback function security	Passed
Design logic	Passed
Economy model	Passed
Oracle calls	Passed
Timestamp dependence	Passed
Compiler warnings.	Passed
Methods execution permissions	Passed
Arithmetic accuracy	Passed
Scoping and declarations	Passed
Economy model	Passed
Safe Zeppelin module	Passed
Cross-function race conditions	Passed
Race conditions and reentrancy.	Passed
Cross-function	Passed
Integer overflow and underflow	Passed
Malicious Event log	Passed
Uninitialized storage pointers	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Private user data leaks	Passed
Impact of the exchange rate on the logic	Passed
Front running	Passed
Possible delays in data delivery	Passed

Summary

In this audit, we thoroughly analyzed the BNB DOGE INU design and implementation. The smart-contract presents a unique offering in current BSC ecosystem. We are truly impressed by the design and implementation, especially the dedication to maximized gas optimization. The current code base is well organized and those identified issues are promptly confirmed and fixed.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

Disclaimer

The following disclaimer outlines the terms and conditions that govern the use of our blockchain smart contract security audit services. By engaging our services, you accept and agree to be bound by these terms and conditions.

1. The purpose of this security audit is to assess the security measures implemented in the smart contract on the blockchain network. Our services are provided on an "as is" basis. We do not guarantee that our services will be error-free or uninterrupted, or that any defects will be corrected.

2. The security audit focuses solely on the smart contract code and its implementation within the blockchain network. It does not cover any external systems, applications, or third-party integrations that may interact with the contract. We do not accept any liability for any damages or losses arising from the use of our services, including but not limited to loss of profits, data, or business interruption.

3. Our services do not constitute legal, financial, or investment advice. We do not provide any warranties or representations as to the accuracy, completeness, or reliability of the information provided in our reports.

4. Orbital Station Excelsior strives to maintain independence and objectivity throughout the security audit process. However, it is important to acknowledge that no audit can be completely free from biases or conflicts of interest. Our services are limited to the scope of work agreed upon with the client. We are not responsible for any issues that fall outside of this scope, including but not limited to issues arising from third-party software or hardware.

5. Our services do not guarantee the security of the smart contract or the blockchain network. We provide recommendations based on our expertise, but it is ultimately the responsibility of the client to ensure the security of their smart contract and blockchain network. The security audit report may include recommendations and suggestions for improving the smart contract's security. It is essential to understand that implementing these recommendations does not guarantee absolute security, as new vulnerabilities may emerge over time.

6. Security threats and vulnerabilities evolve over time. Therefore, it is crucial to continuously monitor and update the smart contract's security measures even after the completion of the audit. Regular security assessments are recommended to ensure ongoing protection.

7. We reserve the right to refuse service to anyone at any time for any reason.

First of all, it is important to note that no security audit can guarantee complete protection against all potential vulnerabilities. While we aim to identify and report any security flaws and vulnerabilities in the smart contract as thoroughly as possible, it is inevitable that some risks may remain undisclosed or may develop over time. Therefore, any use or reliance on the security audit report for the smart contract is done at your own risk.

Furthermore, the security audit report is based on the information provided to us at the time of the audit. Any subsequent changes, modifications or updates made to the smart contract without our knowledge may affect the security of the contract and therefore invalidate our report.

In conclusion, our blockchain smart contract security audit services are provided with the understanding that we are not liable for any damages or losses arising from their use. We recommend that clients seek legal and financial advice before engaging in any blockchain-related activities.

If you have any questions or concerns regarding this disclaimer or the security audit process, please contact auditkyc@osexnetwork.com for further clarification.

Smart Contract Security Audit End Point

About Orbital Station Excelsior (OSEX)

The Orbital Station Excelsior (OSEX) is an international community aimed to bring real benefits to the real world with the help of blockchain technologies, providing all possible assistance and giving hope where it no longer exists.

Our slogan - make the world better together! We do not focus on one blockchain, we use them all, using bridges, Web 3 applications, decentralized exchanges, related technologies and a financial system. We are not an official accountable government financial organization; the main mechanism of our smart contracts is the accumulation of funds, with their subsequent use for charitable needs and purposes around the whole world, as well as expanding the influence of our community on social networks and other Internet platforms. The 21st century is not only about discoveries and achievements in the IT field, but it is also the century of widespread use of these innovations. Blockchain technologies allow us to redirect money flows from one place to another without any intermediaries in the form of banks. The future belongs to electronic money, and Blockchain is the future!

OSEX aims to integrate the advantages of multiple chains to create a high-performance compound ecology. The special redistribution to all holders mechanism of smart-contract from every transaction provides participants with maximum profit. The transaction fee for OSEX includes an active burn mechanism that ensures that the token deflates, which means the price of the token increases over time.

All used DEX strives to provide one-stop liquidity services for more high-quality assets and brings users a safe, reliable, diversified and cost-effective transaction experience.

OSEX-project will build a new business ecology with full application scenarios covered and various chains connected. It will continue to expand application scenarios and lower the usage threshold to provide global users with more convenient, high-performance, low-cost and non-differentiated crypto-asset financial services, thereby realizing fair pricing of assets, instant settlement of transactions and free flow of values.

This project allows all holders to obtain benefits, and is also set to interact with the community. The Orbital Station Excelsior will allow participants to choose the direction of the project. Our beginning includes many opportunities for a crypto project to meet the real world. For example, 1% of each transaction will go to a special charity wallet. Cash income received from this wallet will be sent to charitable projects around the world (rescue funds, volunteer organizations, international movements, environmental protection, etc.). We guarantee to provide reports and proofs of our activities.

The Smart contract security audit and KYC department is an independent department in the general structure of the Orbital Station Excelsior project. Department employees report directly to the general director for their work.

The project was founded on November 07, 2021.

Contacts and links

Orbital Station Excelsior (OSEX)

Website: https://osexnetwork.com/

E-mail:
osex.owner@gmail.com (for advertising and commercial questions)
auditkyc@osexnetwork.com (for smart contract security audit or team KYC requests)

Official Twitter profile: https://twitter.com/OSEXNetwork

Telegram chat group: https://t.me/osex_chat
Telegram announcements channel: https://t.me/osex_announcements
Telegram RU chat group: https://t.me/osex_chat_ru
Telegram NG chat group: https://t.me/osex_chat_ng

Telegram contact (for private dialog): https://t.me/osex_support

We are on blockchain:
https://blockscan.com/address/0x42614e5acf9c084a8afdff402ecd89d19f675c00

Smart contract address (BSC): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00
Smart contract address (Avalanche): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00
Smart contract address (Cronos): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00
Smart contract address (Polygon): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00
Smart contract address (Huobi ECO Chain): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00
Smart contract address (Moonbeam): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00
Smart contract address (Fantom): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00

Contract code on block explorers:
https://bscscan.com/address/0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00#code
https://snowtrace.io/address/0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00#code
https://cronoscan.com/address/0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00#code
https://polygonscan.com/address/0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00#code
https://moonbeam.moonscan.io/address/0x42614e5acf9c084a8afdff402ecd89d19f675c00#code
https://ftmscan.com/address/0x42614e5acf9c084a8afdff402ecd89d19f675c00#code
https://www.hecoinfo.com/en-us/token/0x42614e5acf9c084a8afdff402ecd89d19f675c00?tab=Transfers

Discord: https://discord.gg/4uPVcyen8Z
GitHub: https://github.com/OrbitalStationExcelsior
Instagram: https://www.instagram.com/osexnetwork/
Youtube: https://www.youtube.com/channel/UCm5QiJSu9rySS15arUXZbpw
GitBook: https://osex.gitbook.io/docs/
Medium: https://medium.com/@OrbitalStationExcelsior
Reddit: https://www.reddit.com/r/OSEX/
Techrate free audit for BSC:
https://drive.google.com/file/d/17lO8y0qELM8yhm7iXKXGPyc0NrjXeJOU
Dev Linkedin: https://www.linkedin.com/in/valdisveiss
Whitepaper: https://osexnetwork.com/OSEX_Whitepaper.pdf

Smart contract security audit

Report

I hereby confirm the full implementation of the order from representatives of the «BNB DOG INU» team regarding the security audit of this smart contract: (BNBDOGToken 0x52ABb09Ceb590f04c7d70bE9d6aA420177E94744)

OSEX CEO, Valdis Veiss