Orbital Station Excelsior Smart Contract Security Audit

Contents

Smart Contract Security Audit Start Point

Smart Contract & EVM Compiler Overview

Blockchain: Binance Smart Chain (BSC)

Address: 0xa4aE157BF0EfaCE1f74060E5F295a2030e5fFc98

Explorer Link: https://bscscan.com/address/0xa4ae157bf0eface1f74060e5f295a2030e5ffc98#code

Token Name: BNBDog

Verified: Yes

Status: Deployed

Symbol: (BNBDOG)

Contract Name: BNBDOGToken

Decimals: 9

Total Supply: 2000000000000000000.000000000

Deployment Date: Oct-10-2025 04:53:56 PM UTC

Compiler Version: v0.8.30+commit.73712a01

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: 0x8E75B344F570A2b0d69DE6c189fbFa4f61967a59 at txn 0xd1f705b0645760b98cfe9efb6cf4b9d4748024c53430ce4405a12f166830b21d

Date of audit: 2025/10

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 2025-10-10
*/

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.30;


interface IERC20 {
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address to, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address from, address to, uint256 amount) external returns (bool);
}


interface IERC20Metadata is IERC20 {
    function name() external view returns (string memory);
    function symbol() external view returns (string memory);
    function decimals() external view returns (uint8);
}



abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

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



abstract contract Ownable is Context {
    address private _owner;
    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
    constructor(address owner_)
    {
        _transferOwnership(owner_);
    }


    modifier onlyOwner() {
        _checkOwner();
        _;
    }


    function owner() public view virtual returns (address) {
        return _owner;
    }


    function _checkOwner() internal view virtual {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
    }


    function renounceOwnership() public virtual onlyOwner {
        _transferOwnership(address(0));
    }

    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        _transferOwnership(newOwner);
    }


    function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
```

```
        emit OwnershipTransferred(oldOwner,
newOwner);
    }
}

contract ERC20 is Context, IERC20,
IERC20Metadata {

    mapping(address => uint256) private _bal-
ances;
    mapping(address => mapping(address =>
uint256)) private _allowances;
    uint256 private _totalSupply;
    string private _name;
    string private _symbol;

    constructor(string memory name_, string
memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

    function name() public view virtual over-
ride returns (string memory) {
        return _name;
    }

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

    function decimals() public view virtual
override returns (uint8) {
        return 9;
    }

    function totalSupply() public view vir-
tual override returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) pub-
lic view virtual override returns (uint256) {
        return _balances[account];
    }

    function transfer(address to, uint256
amount) public virtual override returns
(bool) {
        address owner = _msgSender();
        _transfer(owner, to, amount);
        return true;
    }

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

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

    function transferFrom(address from, ad-
dress to, uint256 amount) public virtual
override returns (bool) {
        address spender = _msgSender();
        _spendAllowance(from, spender,
amount);
        _transfer(from, to, amount);
        return true;
    }

    function increaseAllowance(address
spender, uint256 addedValue) public virtual
returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, allow-
ance(owner, spender) + addedValue);
        return true;
    }

    function decreaseAllowance(address
spender, uint256 subtractedValue) public vir-
tual returns (bool) {
```

```
        address owner = _msgSender();
        uint256 currentAllowance = allow-
ance(owner, spender);
        require(currentAllowance >= subtract-
edValue, "ERC20: decreased allowance below
zero");
        unchecked {
            _approve(owner, spender, cur-
rentAllowance - subtractedValue);
        }
        return true;
    }


    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 = _bal-
ances[from];
        require(fromBalance >= amount,
"ERC20: transfer amount exceeds balance");
        unchecked {
            _balances[from] = fromBalance -
amount;
            _balances[to] += amount;
        }
        emit Transfer(from, to, amount);
        _afterTokenTransfer(from, to,
amount);
    }


    function _mint(address account, uint256
amount) internal virtual {
        require(account != address(0),
"ERC20: mint to the zero address");
        _beforeTokenTransfer(address(0), ac-
count, 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), ac-
count, amount);
    }


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

        uint256 accountBalance = _bal-
ances[account];
        require(accountBalance >= amount,
"ERC20: burn amount exceeds balance");
        unchecked {
            _balances[account] = accountBal-
ance - amount;
            _totalSupply -= amount;
        }

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

        _afterTokenTransfer(account, ad-
dress(0), amount);
    }


    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);
    }


    function _spendAllowance(address owner,
address spender, uint256 amount) internal
virtual {
        uint256 currentAllowance = allow-
ance(owner, spender);
        if (currentAllowance !=
type(uint256).max) {
```

```
            require(currentAllowance >=
amount, "ERC20: insufficient allowance");
            unchecked {
                _approve(owner, spender, cur-
rentAllowance - amount);
            }
        }
    }


    function _beforeTokenTransfer(address
from, address to, uint256 amount) internal
virtual {}

    function _afterTokenTransfer(address
from, address to, uint256 amount) internal
virtual {}
}


contract BNBDOGToken is ERC20, Ownable
{

    constructor(address owner_)
ERC20("BNBDog", "BNBDOG") Ownable(owner_)
    {
        _mint(msg.sender,
2000_000_000_000_000_000 * 10 ** decimals());
    }


    function _transfer(address sender, ad-
dress recipient, uint256 amount) internal
virtual override {
        super._transfer(sender, recipient,
amount);
    }

    function burn(uint256 amount) public vir-
tual {
        _burn(_msgSender(), amount);
    }
}
```

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

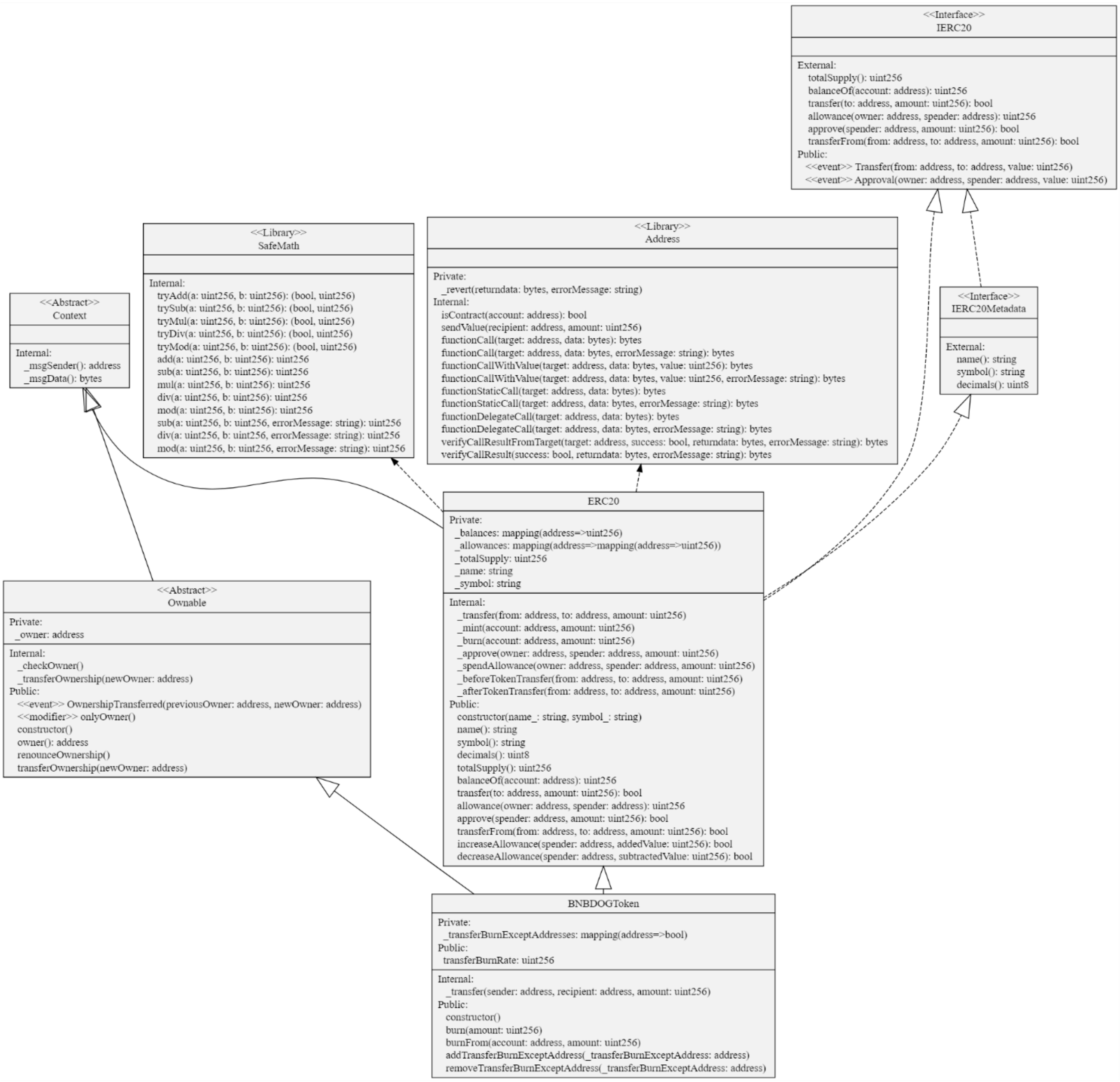
18. Transaction Ordering Dependence
- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Not found
- Severity: Medium

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 feed-backs 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 0xa4ae157bf0eface1f74060e5f295a2030e5ffc98)

OSEX CEO, Valdis Veiss