kaspersky

BlockBen Smart Contract Code Review and Security Analysis

Kaspersky

20.03.2020

kaspersky

Smart Contract Code Review and Security Analysis Report

Customer: BlockNetwork OÜ

Date: 20.03.2020

Contents

Introduction

Kaspersky Lab was contracted by BlockNetwork OÜ to conduct a Smart Contract Code Review and Security Analysis. The initial code review was conducted between 10.03.2020 – 20.03.2020.

BlockNetwork OÜ reacted swiftly and professionally to address issues found, fixed contract was deployed to Ethereum. Two of identified issues was operatively investigated and confirmed by Customer as a non-impactful.

We performed the second analysis on 19.03.2020 and reevaluated found issues. This report presents the findings of the second security assessment of Customer`s smart contract, that has no impact on business-process and cannot be exploited by malicious user.

It should be noted that this audit applies only for smart-contract. Results of the audit doesn't represent the overall security level of entire platform. We always recommend proceeding to several independent audits, continuous assessment during each update of the code-base and a public bug bounty program to ensure the security of the smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Executive Summary

The reviewed source codes are well crafted and follow common security practices and compliant with architecture, described in whitepaper.

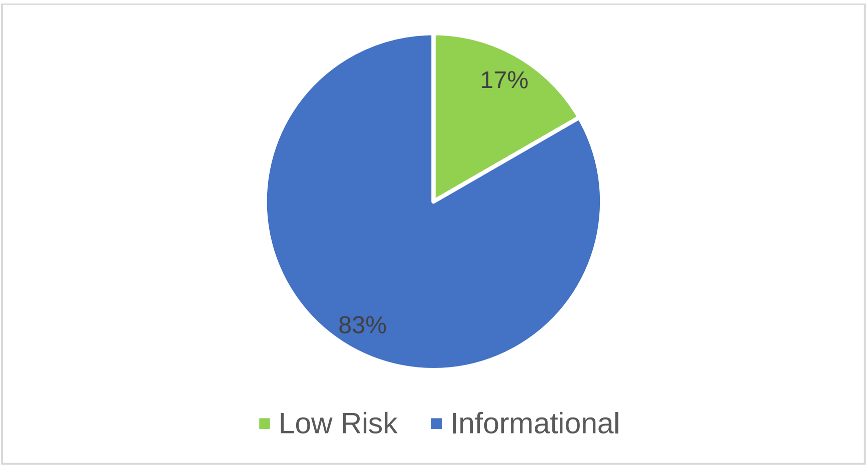
We confirmed during the code audit, that **smart-contract business-description and architecture corresponds to the functionality in the code**.

We performed analysis of code functionality, manual audit and automated checks with tools, described in previous section. All found issues during automated analysis were manually reviewed and applicable vulnerabilities are presented in Issue List section.

During the analysis we found 4 informational issues and 1 low-level issue (Figure 1. Issues Distribution).

- Low-level issues are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution.
- Informational issues code style violations and info statements can't affect smart contract execution and can be ignored.

Regarding the issues found are low level as maximum, t**he code is considered secure and no major fixes are required.** The Code Review Verdict assigned is green.



Scope

On 10.03.2020 contract source codes were obtained

Name	URL
BNOXToken.sol	https://github.com/blockben-official/bnox/blob/master/contracts/BNOXToken.sol
BNOXStandardExt.sol	https://github.com/blockben-official/bnox/blob/master/contracts/BNOXStandardExt.sol
BNOXAdminRole.sol	https://github.com/blockben-official/bnox/blob/master/contracts/BNOXAdminRole.sol
BNOXAdminExt.sol	https://github.com/blockben-official/bnox/blob/master/contracts/BNOXAdminExt.sol
Migrations.sol	https://github.com/blockben-official/bnox/blob/master/contracts/Migrations.sol
0x39fe7e16220A4DBD9CCAc92cCF95e2164f831aFf	https://rinkeby.etherscan.io/address/0x39fe7e16220a4dbd9ccac92ccf95e2164f831aff

On 10.03.2020 Whitepapers were downloaded from specified addresses with their respective MD5 sums

URL	MD5 hash
https://data.blockben.com/whitepaper/BlockBen_WhitePaper.pdf	d4f1a2a87f778dd331252995888e6657

Review Methodology

Throughout the review process, care is taken to ensure that the token contract:

- Implements and adheres to existing ERC-20 Token standard appropriately and effectively
- Documentation and code comments match logic and behavior
- Distributes tokens in a manner that described in corresponding Token Sale T&C (the smart contract is compliant with the requirement of Customer logic, matching the initial constant values etc.)
- Follows best practices in efficient use of gas, without unnecessary waste
- Uses methods safe from reentrance attacks
- Is not affected by known vulnerabilities

To do so our team of experts review the code line-by-line documenting any issues as they are discovered.

We are scanning this smart contract for commonly known and more specific vulnerabilities. Here are some of the vulnerabilities that are considered (the full list includes them but is not limited to them):

- Reentrancy

- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

The static analysis portion of our audit is performed using a series of automated tools, purposefully designed to test the security of the contract, such as Remix, Oyente, Solhint.

All the issues are divided into several risk levels:

- Informational - The issue has no impact on the contract’s ability to operate (code style violations and info statements, can’t affect smart contract execution and can be ignored)
- Low - The issue has minimal impact on the contract’s ability to operate (mostly related to outdated, unused etc. code snippets)
- Medium - The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior
- High - High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
- Critical - The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss

A Code Review Verdict is assigned, which indicates a qualitative score for the Smart Contract source code. Verdict is represented by color mark: red, yellow or green, where

- red indicates the lowest possible source code quality, one or more high/critical issues found
- yellow indicates moderate source code quality, no high/critical issues found and one or more medium issues found
- green indicates the highest possible source code quality, no medium/high/critical issues found

Issues List

BNOX-1 / Informational: Compiler version not fixed - FIXED

Solidity source files indicate the versions of the compiler they can be compiled with:

```
pragma solidity ^0.5.0;
```

It is recommended to follow the example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

Safe alternative:

```
pragma solidity 0.5.0;
```

Specify the exact compiler version (pragma solidity x.y.z;).

BNOX-2 / Informational: Prefer external to public visibility level – HAS NO BUSINESS IMPACT

A following functions with *public* visibility modifier that is not called internally:

1) Migrations.sol function: setCompleted
2) BNOXStandardExt.sol functions: transfer, transferFrom, approve, increaseAllowance, decreaseAllowance, mint, burn, kill
3) BNOXAdminRole.sol functions: getSourceAccountWL, getDestinationAccountWL, setSourceAccountWL, setDestinationAccountWL, setUrl, setTreasuryAddress, setFeeAddress, setBsopoolAddress, setGeneralFee, setBsoFee, pause, unpause
4) BNOXAdminExt.sol functions: removeKYCAdmin, addKYCAdmin, removeTreasuryAdmin, addTreasuryAdmin, removeBNOXAdmin, addBNOXAdmin

Changing visibility level to *external* in many cases spend less gas comparing to functions with public visibility modifier.

BNOX-3 / Low: Using approve function of the ERC-20 token standard in BNOXStandardExt.sol - FIXED

The *approve* function of ERC-20 is vulnerable to front running attack. Using front-running attack one can spend approved tokens before change of *allowance* value.

Function approve is used in BNOXStandardExt.sol (lines 21-24):

```
function approve(address spender, uint256 value) public whenNotPaused returns (bool)
{
        return super.approve(spender, value);
    }
```

Only use the *approve* function of the ERC-20 standard to change allowed amount to 0 or from 0 (wait till transaction is mined and approved).

BNOX-4 / Informational: Private modifier in BNOXAdminRole.sol and BNOXAdminExt.sol - FIXED

In Ethereum, smart contracts bytecode and values of state variables are available for everyone. However, some developers use it to store sensitive data.

Contrary to a popular misconception, the private modifier does not make a variable invisible. Miners have access to all contracts' code and data. Developers must account for the lack of privacy in Ethereum. Make sure that no sensitive data stored in following variables:

1) BNOXAdminRole.sol: _superadmin, _BNOXAdmins, _TreasuryAdmins, _KYCAdmins
2) BNOXAdminExt.sol: _sourceAccountWL, _destinationAccountWL

BNOX-5 / Informational: Use of SafeNath – HAS NO IMPACT

SafeMath library is used:

```
contract ERC20 is Context, IERC20 {

    using SafeMath for uint256;
```

For example the library is used in following lines (927-929):

```
uint256 amountGeneral = amountTotal.sub(amountBso);

uint256 amountRest = amount.sub(amountTotal);
```

It is good practice to use explicit checks where it is really needed, and to avoid extra checks where overflow/underflow vulnerability is impossible.

Best practice example for sub-function:

```
function sub(uint a, uint b) public returns(uint) {

        // explicit check

        require(a >= b, "the result of the subtraction is negative");

        return(a - b);

    }
```

BNOX-6 / Informational: Hardcoded values in demo-setup (BNOXToken in Rinkeby Testnet Network) - FIXED

During the static analysis of BNOXToken in Rinkeby Testnet Network (0x39fe7e16220A4DBD9CCAc92cCF95e2164f831aFf) we identified following hardcoded values (lines: 1007-1045):

```
// add bnox admins

        addBNOXAdmin(address(0x5638Dbf4cEa900Ca98ee34C8F1b5a22781d5c44b));

        setDestinationAccountWL(address(0x5638Dbf4cEa900Ca98ee34C8F1b5a22781d5c44b),
true);

        setSourceAccountWL(address(0x5638Dbf4cEa900Ca98ee34C8F1b5a22781d5c44b),
true);
```

```
        addBNOXAdmin(address(0xf3b9f8D148993A99A92935f5DAe1E35279491A86));
        setDestinationAccountWL(address(0xf3b9f8D148993A99A92935f5DAe1E35279491A86),
true);
        setSourceAccountWL(address(0xf3b9f8D148993A99A92935f5DAe1E35279491A86),
true);

        // add treasury admin
        addTreasuryAdmin(address(0x2EE2466ffE9bd134Cc92E3D3b43dae8892279D62));
        setDestinationAccountWL(address(0x2EE2466ffE9bd134Cc92E3D3b43dae8892279D62),
true);
        setSourceAccountWL(address(0x2EE2466ffE9bd134Cc92E3D3b43dae8892279D62),
true);

        // bsoFee
        setBsoFee(120);
        setBsopoolAddress(address(0x68A374a079a7c5FB41111371Fb2Ad4b528896D0d));
        setDestinationAccountWL(address(0x68A374a079a7c5FB41111371Fb2Ad4b528896D0d),
true);
        setSourceAccountWL(address(0x68A374a079a7c5FB41111371Fb2Ad4b528896D0d),
true);

        // generalFee
        setGeneralFee(80);
        setFeeAddress(address(0x82184BE9F135D345566D4D146fD93cBCacE96Afa));
        setDestinationAccountWL(address(0x82184BE9F135D345566D4D146fD93cBCacE96Afa),
true);
        setSourceAccountWL(address(0x82184BE9F135D345566D4D146fD93cBCacE96Afa),
true);

        //user 1
        setDestinationAccountWL(address(0x5a6b353e9AbeD3b162c559fde03fAf032bc8dE85),
true);
        setSourceAccountWL(address(0x5a6b353e9AbeD3b162c559fde03fAf032bc8dE85),
true);

        //user 2
        setDestinationAccountWL(address(0x9560876BF79a0a78DD7cBFA5B9fBB2e065983f91),
true);
```

```
        setSourceAccountWL(address(0x9560876BF79a0a78DD7cBFA5B9fBB2e065983f91),
true);


        //user 3

        setDestinationAccountWL(address(0x4e1f35776Cb0BF51190eC01c86eF8b7Bb629Ce62),
true);

        setSourceAccountWL(address(0x4e1f35776Cb0BF51190eC01c86eF8b7Bb629Ce62),
true);
```

Please check hardcoded address and its usage. Remove the address from main network.

Appendix A

Reviewed contracts' source code listing can be found below.

BNOXToken

```
pragma solidity ^0.5.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol";
import "./BNOXStandardExt.sol";

/// @author Blockben
/// @title BNOXToken
/// @notice BNOXToken implementation
contract BNOXToken is BNOXStandardExt, ERC20Detailed {

    /// @notice Constructor: creating initial supply and setting one admin
    /// @dev Not working with decimal numbers
    /// @param superadmin superadmnin of the token
    constructor(address superadmin) BNOXStandardExt(superadmin)
ERC20Detailed("BlockNoteX", "BNOX", 2) public {
    }
}
```

BNOXStandardExt

```
pragma solidity ^0.5.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Roles.sol";
import "./BNOXAdminExt.sol";

/// @author Blockben
/// @title BNOXToken standard extentions for the token functionalities
/// @notice BNOXToken extentions for mint, burn and kill
contract BNOXStandardExt is BNOXAdminExt, ERC20 {

    // constructor delegating superadmin role to the BNOXAdminRole
    constructor (address superadmin) BNOXAdminExt(superadmin) internal {
    }

    /// @notice transfer BNOX token, only if not paused
    /// @dev ...
    /// @param to The address of the account to be transferred
    /// @param value The amount of token to be transferred
    /// @return true if everything is cool
    function transfer(address to, uint256 value) public whenNotPaused returns (bool)
{
        _transfer(_msgSender(), to, value);
        return true;
    }

    /// @notice transferFrom BNOX token, only if not paused
    /// @dev ...
    /// @param from The address transferred from
    /// @param to The amount transferred to
    /// @param value The amount of token to be transferred
    function transferFrom(address from, address to, uint256 value) public
whenNotPaused returns (bool) {
        return super.transferFrom(from, to, value);
    }

    /// @notice approve BNOX token to be moved with tranferFrom, only if not paused
    /// @dev ...
    /// @param spender The address to be approved
    /// @param value The amount of token to be allowed to be transferred
    function approve(address spender, uint256 value) public whenNotPaused returns
(bool) {
        return super.approve(spender, value);
    }

    /// @notice increase approved BNOX token, only if not paused
    /// @dev ...
    /// @param spender The address to be approved
    /// @param addedValue The amount of token to be allowed to be transferred
    function increaseAllowance(address spender, uint256 addedValue) public
whenNotPaused returns (bool) {
        return super.increaseAllowance(spender, addedValue);
    }

    /// @notice decrease approved BNOX token, only if not paused
    /// @dev ...
    /// @param spender The address to be approved
    /// @param subtractedValue The amount of token to be allowed to be transferred
    function decreaseAllowance(address spender, uint256 subtractedValue) public
whenNotPaused returns (bool) {
        return super.decreaseAllowance(spender, subtractedValue);
    }


    /// @notice mint BNOX token, only Treasury admin, only if no paused
    /// @dev ...
    /// @param account The address of the account to be minted
    /// @param amount The amount of token to be minted
    /// @return true if everything is cool
    function mint(address account, uint256 amount) public onlyTreasuryAdmin
whenNotPaused returns (bool) {
        _mint(account, amount);
        return true;
    }

    /// @notice burning BNOX token from the treasury account, only if not paused
    /// @dev ...
    /// @param amount The amount of token to be burned
    function burn(uint256 amount) public onlyTreasuryAdmin whenNotPaused {
        require(getSourceAccountWL(treasuryAddress) == true, "Treasury address is
locked by the source account whitelist");
        _burnFrom(treasuryAddress, amount);
    }

    /// @notice killing the contract, only paused contract can be killed by the admin
    /// @dev ...
    /// @param toChashOut The address where the ether of the token should be sent
    function kill(address payable toChashOut) public onlyBNOXAdmin {
        require (paused == true, "only paused contract can be killed");
        selfdestruct(toChashOut);
    }

    /// @notice mint override to consider address lock for KYC
    /// @dev ...
    /// @param account The address where token is mineted
    /// @param amount The amount to be minted
    function _mint(address account, uint256 amount) internal {
        require(getDestinationAccountWL(account) == true, "Target account is locked
by the destination account whitelist");

        super._mint(account, amount);
    }

    /// @notice transfer override to consider locks for KYC
    /// @dev ...
    /// @param sender The address from where the token sent
    /// @param recipient Recipient address
    /// @param amount The amount to be transferred
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(getSourceAccountWL(sender) == true, "Sender account is not unlocked
by the source account whitelist");
        require(getDestinationAccountWL(recipient) == true, "Target account is not
unlocked by the destination account whitelist");
        require(getDestinationAccountWL(feeAddress) == true, "General fee account is
not unlocked by the destination account whitelist");
        require(getDestinationAccountWL(bsopoolAddress) == true, "Bso pool account is
not unlocked by the destination account whitelist");

        // transfer to the trasuryAddress or transfer from the treasuryAddress do not
cost transaction fee
        if((sender == treasuryAddress) || (recipient == treasuryAddress)){
            super._transfer(sender, recipient, amount);
        }
        else {

            // three decimal in percent
            // The decimalcorrection is 100.000, but to avoid rounding errors, first
use 10.000 and
            // where we use decimalCorrection the calculation must add 5 and divide
10 at the and
            uint256 decimalCorrection = 10000;

            // calculate and transfer fee
            uint256 generalFee256 = generalFee;

            uint256 bsoFee256 = bsoFee;

            uint256 totalFee = generalFee256.add(bsoFee256);

            // To avoid rounding errors add 5 and then div by 10. Read comment at
decimalCorrection
            uint256 amountTotal =
amount.mul(totalFee).div(decimalCorrection).add(5).div(10);

            // To avoid rounding errors add 5 and then div by 10. Read comment at
decimalCorrection
            uint256 amountBso =
amount.mul(bsoFee256).div(decimalCorrection).add(5).div(10);
```

```
            uint256 amountGeneral = amountTotal.sub(amountBso);

            uint256 amountRest = amount.sub(amountTotal);

            super._transfer(sender, recipient, amountRest);
            super._transfer(sender, feeAddress, amountGeneral);
            super._transfer(sender, bsopoolAddress, amountBso);
        }
    }
}
```

BNOXAdminExt

```
pragma solidity ^0.5.0;

import "./BNOXAdminRole.sol";

/// @author Blockben
/// @title BNOXToken KYC specific extentions of the token functionalities
/// @notice BNOXToken extentions for handling source and destination KYC
contract BNOXAdminExt is BNOXAdminRole {

    /// @notice administrating locks for the source contracts
    mapping(address => bool) private _sourceAccountWL;

    /// @notice administrating locks for the destination contracts
    mapping(address => bool) private _destinationAccountWL;

    /// @notice url for external verification
    string public url;

    /// @notice addres for collecting and burning tokens
    address public treasuryAddress;

    /// @notice addres for fee
    address public feeAddress;

    /// @notice addres for bsopool
    address public bsopoolAddress;

    /// @notice general transaction fee
    uint16 public generalFee;

    /// @notice bso transaction fee
    uint16 public bsoFee;

    /// @notice basic functionality can be paused
    bool public paused;

    /// @notice Event for locking or unlocking a source account
    event BNOXSourceAccountWL(address indexed account, bool lockValue);
    /// @notice Event for locking or unlocking a destination account
    event BNOXDestinationAccountWL(address indexed account, bool lockValue);
    /// @notice Event for locking or unlocking a destination account
    event BNOXUrlSet(string ulr);
    /// @notice Event for changing the terasury address
    event BNOXTreasuryAddressChange(address newAddress);
    /// @notice Event for changing the fee address
    event BNOXFeeAddressChange(address newAddress);
    /// @notice Event for changing the bsopool address
    event BNOXBSOPOOLAddressChange(address newAddress);
    /// @notice Event for changing the general fee
    event BNOXGeneralFeeChange(uint256 newFee);
    /// @notice Event for changing the bso fee
    event BNOXBSOFeeChange(uint256 newFee);
    /// @notice Token is paused by the account
    event BNOXPaused(address account);
    /// @notice Token is un-paused by the account
    event BNOXUnpaused(address account);

    // constructor setting the contract unpaused and delegating the superadmin
    constructor (address superadmin) BNOXAdminRole(superadmin) internal {
        paused = false;
    }

    /// @notice Modifier only if not paused
    modifier whenNotPaused() {
        require(!paused, "The token is paused!");
        _;
    }

    /// @notice getting if an address is locked as a source address
    /// @dev ...
    /// @param sourceAddress The address of the account to be checked
    function getSourceAccountWL(address sourceAddress) public view returns (bool) {
        return _sourceAccountWL[sourceAddress];
    }

    /// @notice getting if an address is locked as a destinationAddress
    /// @dev ...
    /// @param destinationAddress The address of the account to be checked
    function getDestinationAccountWL(address destinationAddress) public view returns
(bool) {
        return _destinationAccountWL[destinationAddress];
    }

    /// @notice setting if an address is locked as a source address
    /// @dev ...
    /// @param sourceAddress The address of the account to be checked
    function setSourceAccountWL(address sourceAddress, bool lockValue) public
onlyKYCAdmin {
        _sourceAccountWL[sourceAddress] = lockValue;
        emit BNOXSourceAccountWL(sourceAddress, lockValue);
    }

    /// @notice setting if an address is locked as a destination address
    /// @dev ...
    /// @param destinationAddress The address of the account to be checked
    function setDestinationAccountWL(address destinationAddress, bool lockValue)
public onlyKYCAdmin {
        _destinationAccountWL[destinationAddress] = lockValue;
        emit BNOXDestinationAccountWL(destinationAddress, lockValue);
    }

    /// @notice setting the url referring to the documentation
    /// @dev ...
    /// @param newUrl The new url
```

```
function setUrl(string memory newUrl) public onlyBNOXAdmin {
    url = newUrl;
    emit BNOXUrlSet(newUrl);
}

/// @notice setting a new address for treasuryAddress
/// @dev ...
/// @param newAddress The new address to set
function setTreasuryAddress(address newAddress) public onlyBNOXAdmin {
    treasuryAddress = newAddress;
    emit BNOXTreasuryAddressChange(newAddress);
}

/// @notice setting a new address for feeAddress
/// @dev ...
/// @param newAddress The new address to set
function setFeeAddress(address newAddress) public onlyBNOXAdmin {
    feeAddress = newAddress;
    emit BNOXFeeAddressChange(newAddress);
}

/// @notice setting a new address for feeAddress
/// @dev ...
/// @param newAddress The new address to set
function setBsopoolAddress(address newAddress) public onlyBNOXAdmin {
    bsopoolAddress = newAddress;
    emit BNOXBSOPOOLAddressChange(newAddress);
}

/// @notice setting a new general fee
/// @dev ...
/// @param newFee The new fee to set
function setGeneralFee(uint16 newFee) public onlyBNOXAdmin {
    generalFee = newFee;
    emit BNOXGeneralFeeChange(newFee);
}

/// @notice setting a new bsoFee fee
/// @dev ...
/// @param newFee The new fee to set
function setBsoFee(uint16 newFee) public onlyBNOXAdmin {
    bsoFee = newFee;
    emit BNOXBSOFeeChange(newFee);
}

/// @notice pause the contract
/// @dev ...
function pause() public onlyBNOXAdmin {
    require(paused == false, "The contract is already paused");
    paused = true;
    emit BNOXPaused(_msgSender());
}

/// @notice un-pause the contract
/// @dev ...
function unpause() public onlyBNOXAdmin {
    paused = false;
    emit BNOXUnpaused(_msgSender());
}
```

```
}
```

BNOXAdminRole

```
pragma solidity ^0.5.0;

import "@openzeppelin/contracts/GSN/Context.sol";
import "@openzeppelin/contracts/access/Roles.sol";

/// @author BlockBen
/// @title BNOXToken Adminrole
/// @notice BNOXToken Adminrole implementation
contract BNOXAdminRole is Context {
    using Roles for Roles.Role;

    /// @notice superadmin on paper wallet for worst case key compromise
    address private _superadmin;

    /// @notice list (mapping) of BNOX admins
    Roles.Role private _BNOXAdmins;

    /// @notice list (mapping) of Treasury admins can only mint or burn
    Roles.Role private _TreasuryAdmins;

    /// @notice list (mapping) of KYC admins can only whitelist and blacklist
addresses
    Roles.Role private _KYCAdmins;

    /// @notice Event for Admin addedd
    event BNOXAdminAdded(address indexed account);
    /// @notice Event for Admin removed
    event BNOXAdminRemoved(address indexed account);
    /// @notice Event for adding treasury admin
    event BNOXTreasuryAdminAdded(address indexed account);
    /// @notice Event for rmoving treasury admin
    event BNOXTreasuryAdminRemoved(address indexed account);
    /// @notice Event for adding KYC admin
    event BNOXKYCAdminAdded(address indexed account);
    /// @notice Event for rmoving KYC admin
    event BNOXKYCAdminRemoved(address indexed account);

    // constructor setting the superadmin and adding deployer as admin
    constructor (address superadmin) internal {
        _superadmin = superadmin;
        _BNOXAdmins.add(_msgSender());
        emit BNOXAdminAdded(_msgSender());
    }

    /// @notice Modifyer checking if the caller is a BNOX admin
    modifier onlyBNOXAdmin() {
        require((isBNOXAdmin(_msgSender()) || (_msgSender() == _superadmin)),
"BNOXAdmin: caller does not have the BNOXAdmin role");
        _;
    }

    /// @notice Modifyer checking if the caller is a Treasury admin
    modifier onlyTreasuryAdmin() {
        require(isTreasuryAdmin(_msgSender()), "Treasury admin: caller does not have
the TreasuryAdmin role");
        _;
    }

    /// @notice Modifyer checking if the caller is a KYC admin
    modifier onlyKYCAdmin() {
        require(isKYCAdmin(_msgSender()), "KYC admin: caller does not have the
KYCAdmin role");
        _;
    }

    /// @notice Checking if the address is a KYC admin
    /// @dev ...
    /// @param account The address of the account to be checked
    /// @return true if the account is a KYC admin
    function isKYCAdmin(address account) public view returns (bool) {
        return _KYCAdmins.has(account);
    }

    /// @notice Checking if the address is a Treasury admin
    /// @dev ...
    /// @param account The address of the account to be checked
    /// @return true if the account is a treasury admin
    function isTreasuryAdmin(address account) public view returns (bool) {
        return _TreasuryAdmins.has(account);
    }

    /// @notice Checking if the address is a BNOX admin
    /// @dev ...
    /// @param account The address of the account to be checked
    /// @return true if the account is an admin
    function isBNOXAdmin(address account) public view returns (bool) {
        return _BNOXAdmins.has(account);
    }

    /// @notice Adding an account as a BNOX admin
    /// @dev ...
    /// @param account The address of the account to be added
    function addBNOXAdmin(address account) public onlyBNOXAdmin {
        _BNOXAdmins.add(account);
        emit BNOXAdminAdded(account);
    }

    /// @notice Removing an account as a BNOX admin
    /// @dev ...
    /// @param account The address of the account to be added
    function removeBNOXAdmin(address account) public onlyBNOXAdmin {
        _BNOXAdmins.remove(account);
        emit BNOXAdminRemoved(account);
    }

    /// @notice Adding an account as a Treasury admin
    /// @dev ...
    /// @param account The address of the account to be added
    function addTreasuryAdmin(address account) public onlyBNOXAdmin {
        _TreasuryAdmins.add(account);
        emit BNOXTreasuryAdminAdded(account);
    }
    /// @notice Removing an account as a Treasury admin
    /// @dev ...
    /// @param account The address of the account to be removed
    function removeTreasuryAdmin(address account) public onlyBNOXAdmin {
        _TreasuryAdmins.remove(account);
        emit BNOXTreasuryAdminRemoved(account);
    }

    /// @notice Adding an account as a KYC admin
    /// @dev ...
    /// @param account The address of the account to be added
    function addKYCAdmin(address account) public onlyBNOXAdmin {
        _KYCAdmins.add(account);
        emit BNOXKYCAdminAdded(account);
    }

    /// @notice Removing an account as a KYC admin
    /// @dev ...
    /// @param account The address of the account to be removed
    function removeKYCAdmin(address account) public onlyBNOXAdmin {
        _KYCAdmins.remove(account);
        emit BNOXKYCAdminRemoved(account);
    }

}
```

Migrations

```
pragma solidity >=0.4.21 <0.7.0;

contract Migrations {
  address public owner;
  uint public last_completed_migration;

  constructor() public {
    owner = msg.sender;
  }

  modifier restricted() {
    if (msg.sender == owner) _;
  }

  function setCompleted(uint completed) public restricted {
    last_completed_migration = completed;
  }
}
```

BNOXToken (0x39fe7e16220A4DBD9CCAc92cCF95e2164f831aFf)

```
/**

 *Submitted for verification at Etherscan.io on 2020-02-14

*/
```

```
pragma solidity ^0.5.0;

/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

    /**
     * @dev Give an account access to this role.
     */
    function add(Role storage role, address account) internal {
        require(!has(role, account), "Roles: account already has role");
        role.bearer[account] = true;
    }

    /**
     * @dev Remove an account's access to this role.
     */
    function remove(Role storage role, address account) internal {
        require(has(role, account), "Roles: account does not have role");
        role.bearer[account] = false;
    }

    /**
     * @dev Check if an account has this role.
     * @return bool
     */
    function has(Role storage role, address account) internal view returns (bool) {
        require(account != address(0), "Roles: account is the zero address");
        return role.bearer[account];
    }
}

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
```

```
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

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

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but
the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

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

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with
custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}


/*
 * @dev 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.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
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);
}


/**
```

```
 * @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 {ERC20Mintable}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-
mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

    /**
```

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

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

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

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view returns (uint256)
{
        return _allowances[owner][spender];
    }
    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(_msgSender(), 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};
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for `sender`'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public
returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount,
"ERC20: transfer amount exceeds allowance"));
        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 returns
(bool) {
        _approve(_msgSender(), spender,
_allowances[_msgSender()][spender].add(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
returns (bool) {
        _approve(_msgSender(), spender,
_allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance
below zero"));
```

```
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount
exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, 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
     *
```

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

         _totalSupply = _totalSupply.add(amount);
         _balances[account] = _balances[account].add(amount);
         emit Transfer(address(0), account, amount);
     }

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

         _balances[account] = _balances[account].sub(amount, "ERC20: burn amount
exceeds balance");
         _totalSupply = _totalSupply.sub(amount);
         emit Transfer(account, address(0), amount);
     }

     /**
      * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
      *
      * This is internal function is equivalent to `approve`, and can be used to
      * e.g. set automatic allowances for certain subsystems, etc.
      *
      * Emits an {Approval} event.
      *
      * Requirements:
      *
      * - `owner` cannot be the zero address.
      * - `spender` cannot be the zero address.
      */
     function _approve(address owner, address spender, uint256 amount) internal {
         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 Destroys `amount` tokens from `account`.`amount` is then deducted
      * from the caller's allowance.
      *
      * See {_burn} and {_approve}.
      */
     function _burnFrom(address account, uint256 amount) internal {
         _burn(account, amount);
         _approve(account, _msgSender(),
 _allowances[account][_msgSender()].sub(amount, "ERC20: burn amount exceeds
 allowance"));
     }
 }

 /// @author BlockBen
 /// @title BNOXToken Adminrole
 /// @notice BNOXToken Adminrole implementation
```

```
contract BNOXAdminRole is Context {
    using Roles for Roles.Role;

    /// @notice superadmin on paper wallet for worst case key compromise
    address private _superadmin;

    /// @notice list (mapping) of BNOX admins
    Roles.Role private _BNOXAdmins;

    /// @notice list (mapping) of Treasury admins can only min or burn
    Roles.Role private _TreasuryAdmins;

    /// @notice Event for Admin addedd
    event BNOXAdminAdded(address indexed account);
    /// @notice Event for Admin removed
    event BNOXAdminRemoved(address indexed account);
    /// @notice Event for adding treasury admin
    event BNOXTreasuryAdminAdded(address indexed account);
    /// @notice Event for rmoving treasury admin
    event BNOXTreasuryAdminRemoved(address indexed account);

    // constructor setting the superadmin and adding deployer as admin
    constructor (address superadmin) internal {
        _superadmin = superadmin;
        _addBNOXAdmin(_msgSender());
    }

    /// @notice Modifyer checking if the caller is a BNOX admin
    modifier onlyBNOXAdmin() {
        require((isBNOXAdmin(_msgSender()) || (_msgSender() == _superadmin)),
"BNOXAdmin: caller does not have the BNOXAdmin role");
        _;
    }
    /// @notice Modifyer checking if the caller is a Treasury
    modifier onlyTreasuryAdmin() {
        require(isTreasuryAdmin(_msgSender()), "Treasury admin: caller does not have
the TreasuryAdmin role");
        _;
    }

    /// @notice Checking if the address is a Treasury admin
    /// @dev ...
    /// @param account The address of the account to be checked
    /// @return true if the account is a treasury admin
    function isTreasuryAdmin(address account) public view returns (bool) {
        return _TreasuryAdmins.has(account);
    }

    /// @notice Checking if the address is a BNOX admin
    /// @dev ...
    /// @param account The address of the account to be checked
    /// @return true if the account is an admin
    function isBNOXAdmin(address account) public view returns (bool) {
        return _BNOXAdmins.has(account);
    }

    /// @notice Adding an account as a BNOX admin
    /// @dev ...
    /// @param account The address of the account to be added
    function addBNOXAdmin(address account) public onlyBNOXAdmin {
        _addBNOXAdmin(account);
    }

    /// @notice Removing an account as a BNOX admin
    /// @dev ...
    /// @param account The address of the account to be added
    function removeBNOXAdmin(address account) public onlyBNOXAdmin {
        _removeBNOXAdmin(account);
    }

    /// @notice adding an account as a BNOX admin internal
    /// @dev ...
    /// @param account The address of the account to be added
    function _addBNOXAdmin(address account) internal {
        _BNOXAdmins.add(account);
        emit BNOXAdminAdded(account);
    }

    /// @notice Removing an account as a BNOX admin internal
    /// @dev ...
    /// @param account The address of the account to be removed
    function _removeBNOXAdmin(address account) internal {
        _BNOXAdmins.remove(account);
        emit BNOXAdminRemoved(account);
    }

    /// @notice Adding an account as a Treasury admin
    /// @dev ...
    /// @param account The address of the account to be added
    function addTreasuryAdmin(address account) public onlyBNOXAdmin {
        _addTreasuryAdmin(account);
    }

    /// @notice Removing an account as a Treasury admin
    /// @dev ...
    /// @param account The address of the account to be removed
    function removeTreasuryAdmin(address account) public onlyBNOXAdmin {
        _removeTreasuryAdmin(account);
    }

    /// @notice adding an account as a Treasury admin internal
    /// @dev ...
    /// @param account The address of the account to be added
    function _addTreasuryAdmin(address account) internal {
        _TreasuryAdmins.add(account);
        emit BNOXTreasuryAdminAdded(account);
    }

    /// @notice Removing an account as a BNOX admin internal
    /// @dev ...
    /// @param account The address of the account to be removed
    function _removeTreasuryAdmin(address account) internal {
        _TreasuryAdmins.remove(account);
        emit BNOXTreasuryAdminRemoved(account);
    }

}

/// @author Blockben
/// @title BNOXToken KYC specific extentions of the token functionalities
/// @notice BNOXToken extentions for handling source and destination KYC
contract BNOXAdminExt is BNOXAdminRole {

    /// @notice administrating locks for the source contracts
    mapping(address => bool) private _sourceAccountWL;

    /// @notice administrating locks for the destination contracts
    mapping(address => bool) private _destinationAccountWL;

    /// @notice url for external verification
    string public url;

    /// @notice addres for collecting and burning tokens
    address public treasuryAddress;
```

```
/// @notice addres for fee
address public feeAddress;

/// @notice addres for bsopool
address public bsopoolAddress;

/// @notice general transaction fee
uint16 public generalFee;

/// @notice bso transaction fee
uint16 public bsoFee;

/// @notice basic functionality can be paused
bool public paused;

/// @notice Event for locking or unlocking a source account
event BNOXSourceAccountWL(address indexed account, bool lockValue);
/// @notice Event for locking or unlocking a destination account
event BNOXDestinationAccountWL(address indexed account, bool lockValue);
/// @notice Event for locking or unlocking a destination account
event BNOXUrlSet(string ulr);
/// @notice Event for changing the terasury address
event BNOXTreasuryAddressChange(address newAddress);
/// @notice Event for changing the fee address
event BNOXFeeAddressChange(address newAddress);
/// @notice Event for changing the bsopool address
event BNOXBSOPOOLAddressChange(address newAddress);
/// @notice Event for changing the general fee
event BNOXGeneralFeeChange(uint256 newFee);
/// @notice Event for changing the bso fee
event BNOXBSOFeeChange(uint256 newFee);
/// @notice Token is paused by the account
event BNOXPaused(address account);
/// @notice Token is un-paused by the account
```

```
    event BNOXUnpaused(address account);

    // constructor setting the contract unpaused and delegating the superadmin
    constructor (address superadmin) BNOXAdminRole(superadmin) internal {
        paused = false;
    }

    /// @notice Modifier only if not paused
    modifier whenNotPaused() {
        require(!paused, "Pausable: paused");
        _;
    }

    /// @notice getting if an address is locked as a source address
    /// @dev ...
    /// @param sourceAddress The address of the account to be checked
    function getSourceAccountWL(address sourceAddress) public view returns (bool) {
        return _sourceAccountWL[sourceAddress];
    }

    /// @notice getting if an address is locked as a destinationAddress
    /// @dev ...
    /// @param destinationAddress The address of the account to be checked
    function getDestinationAccountWL(address destinationAddress) public view returns
(bool) {
        return _destinationAccountWL[destinationAddress];
    }

    /// @notice setting if an address is locked as a source address
    /// @dev ...
    /// @param sourceAddress The address of the account to be checked
    function setSourceAccountWL(address sourceAddress, bool lockValue) public
onlyBNOXAdmin {
        _sourceAccountWL[sourceAddress] = lockValue;
        emit BNOXSourceAccountWL(sourceAddress, lockValue);
    }

    /// @notice setting if an address is locked as a destination address
    /// @dev ...
    /// @param destinationAddress The address of the account to be checked
    function setDestinationAccountWL(address destinationAddress, bool lockValue)
public onlyBNOXAdmin {
        _destinationAccountWL[destinationAddress] = lockValue;
        emit BNOXDestinationAccountWL(destinationAddress, lockValue);
    }

    /// @notice setting the url referring to the documentation
    /// @dev ...
    /// @param newUrl The new url
    function setUrl(string memory newUrl) public onlyBNOXAdmin {
        url = newUrl;
        emit BNOXUrlSet(newUrl);
    }

    /// @notice setting a new address for treasuryAddress
    /// @dev ...
    /// @param newAddress The new address to set
    function setTreasuryAddress(address newAddress) public onlyBNOXAdmin {
        treasuryAddress = newAddress;
        emit BNOXTreasuryAddressChange(newAddress);
    }

    /// @notice setting a new address for feeAddress
    /// @dev ...
    /// @param newAddress The new address to set
    function setFeeAddress(address newAddress) public onlyBNOXAdmin {
        feeAddress = newAddress;
        emit BNOXFeeAddressChange(newAddress);
    }

    /// @notice setting a new address for feeAddress
    /// @dev ...
    /// @param newAddress The new address to set
    function setBsopoolAddress(address newAddress) public onlyBNOXAdmin {
        bsopoolAddress = newAddress;
        emit BNOXBSOPOOLAddressChange(newAddress);
    }

    /// @notice setting a new general fee
    /// @dev ...
    /// @param newFee The new fee to set
    function setGeneralFee(uint16 newFee) public onlyBNOXAdmin {
        generalFee = newFee;
        emit BNOXGeneralFeeChange(newFee);
    }

    /// @notice setting a new bsoFee fee
    /// @dev ...
    /// @param newFee The new fee to set
    function setBsoFee(uint16 newFee) public onlyBNOXAdmin {
        bsoFee = newFee;
        emit BNOXBSOFeeChange(newFee);
    }

    /// @notice pause the contract
    /// @dev ...
    function pause() public onlyBNOXAdmin {
        require(paused == false, "The contract is already paused");
        paused = true;
        emit BNOXPaused(_msgSender());
    }
    /// @notice un-pause the contract
    /// @dev ...
    function unpause() public onlyBNOXAdmin {
        paused = false;
        emit BNOXUnpaused(_msgSender());
    }

}



/// @author Blockben
/// @title BNOXToken standard extentions for the token functionalities
/// @notice BNOXToken extentions for mint, burn and kill
contract BNOXStandardExt is BNOXAdminExt, ERC20 {

    // constructor delegating superadmin role to the BNOXAdminRole
    constructor (address superadmin) BNOXAdminExt(superadmin) internal {
    }

    /// @notice transfer BNOX token, only if not paused
    /// @dev ...
    /// @param to The address of the account to be transferred
    /// @param value The amount of token to be transferred
    /// @return true if everything is cool
    function transfer(address to, uint256 value) public whenNotPaused returns (bool)
{
        _transfer(_msgSender(), to, value);
        return true;
    }

    /// @notice transferFrom BNOX token, only if not paused
    /// @dev ...
    /// @param from The address transferred from
    /// @param to The amount transferred to
    /// @param value The amount of token to be transferred

    function transferFrom(address from, address to, uint256 value) public
whenNotPaused returns (bool) {

        return super.transferFrom(from, to, value);

    }


    /// @notice approve BNOX token to be moved with tranferFrom, only if not paused

    /// @dev ...

    /// @param spender The address to be approved

    /// @param value The amount of token to be allowed to be transferred

    function approve(address spender, uint256 value) public whenNotPaused returns
(bool) {

        return super.approve(spender, value);

    }


    /// @notice increase approved BNOX token, only if not paused

    /// @dev ...

    /// @param spender The address to be approved

    /// @param addedValue The amount of token to be allowed to be transferred

    function increaseAllowance(address spender, uint256 addedValue) public
whenNotPaused returns (bool) {

        return super.increaseAllowance(spender, addedValue);

    }


    /// @notice decrease approved BNOX token, only if not paused

    /// @dev ...

    /// @param spender The address to be approved

    /// @param subtractedValue The amount of token to be allowed to be transferred

    function decreaseAllowance(address spender, uint256 subtractedValue) public
whenNotPaused returns (bool) {

        return super.decreaseAllowance(spender, subtractedValue);

    }



    /// @notice mint BNOX token, only Treasury admin, only if no paused
    /// @dev ...

    /// @param account The address of the account to be minted

    /// @param amount The amount of token to be minted

    /// @return true if everything is cool

    function mint(address account, uint256 amount) public onlyTreasuryAdmin
whenNotPaused returns (bool) {

        _mint(account, amount);

        return true;

    }


    /// @notice burning BNOX token from a given account, only if not paused

    /// @dev ...

    /// @param account The address of the account to be burned

    /// @param amount The amount of token to be burned

    function burnFrom(address account, uint256 amount) public onlyTreasuryAdmin
whenNotPaused {

        require(getSourceAccountWL(account) == true, "Account is locked by the source
account whitelist");

        require(treasuryAddress == account, "Burn is possible only from the
treasuryAddress");

        _burnFrom(account, amount);

    }


    /// @notice killing the contract, only paused contract can be killed by the admin

    /// @dev ...

    /// @param toChashOut The address where the ether of the token should be sent

    function kill(address payable toChashOut) public onlyBNOXAdmin {

        require (paused == true, "only paused contract can be killed");

        selfdestruct(toChashOut);

    }


    /// @notice mint override to consider address lock for KYC

    /// @dev ...

    /// @param account The address where token is mineted

    /// @param amount The amount to be minted
    function _mint(address account, uint256 amount) internal {
        require(getDestinationAccountWL(account) == true, "Target account is locked
by the destination account whitelist");

        super._mint(account, amount);
    }

    /// @notice transfer override to consider locks for KYC
    /// @dev ...
    /// @param sender The address from where the token sent
    /// @param recipient Recipient address
    /// @param amount The amount to be transferred
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(getSourceAccountWL(sender) == true, "Sender account is not unlocked
by the source account whitelist");
        require(getDestinationAccountWL(recipient) == true, "Target account is not
unlocked by the destination account whitelist");
        require(getDestinationAccountWL(feeAddress) == true, "General fee account is
not unlocked by the destination account whitelist");
        require(getDestinationAccountWL(bsopoolAddress) == true, "Bso pool account is
not unlocked by the destination account whitelist");

        // transfer to the trasuryAddress or transfer from the treasuryAddress do not
cost transaction fee
        if((sender == treasuryAddress) || (recipient == treasuryAddress)){
            super._transfer(sender, recipient, amount);
        }
        else {

            // three decimal in percent
            uint256 decimalCorrection = 100000;

            // calculate and transfer fee
            uint256 generalFee256 = generalFee;

            uint256 bsoFee256 = bsoFee;
            uint256 totalFee = generalFee256.add(bsoFee256);

            uint256 amountTotal = amount.mul(totalFee).div(decimalCorrection);

            uint256 amountBso = amount.mul(bsoFee256).div(decimalCorrection);

            uint256 amountGeneral = amountTotal.sub(amountBso);

            uint256 amountRest = amount.sub(amountTotal);

            super._transfer(sender, recipient, amountRest);
            super._transfer(sender, feeAddress, amountGeneral);
            super._transfer(sender, bsopoolAddress, amountBso);
        }
    }
}

/**
 * @dev Optional functions from the ERC20 standard.
 */
contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /**
     * @dev Sets the values for `name`, `symbol`, and `decimals`. All three of
     * these values are immutable: they can only be set once during
     * construction.
     */
    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }

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

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view 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.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}
```

```
/// @author Blockben

/// @title BNOXToken

/// @notice BNOXToken implementation

contract BNOXToken is BNOXStandardExt, ERC20Detailed {


    /// @notice Constructor: creating initial supply and setting one admin

    /// @dev Not working with decimal numbers

    /// @param initialSupply The inital supply for the token

    constructor(uint256 initialSupply, address superadmin)
BNOXStandardExt(superadmin) ERC20Detailed("BlockNoteX", "BNOX", 2) public {

        setDestinationAccountWL(msg.sender, true);

        setSourceAccountWL(msg.sender, true);

        _mint(msg.sender, initialSupply);


        // DEMO SETUP - default values


        // set treasury address

        setTreasuryAddress(msg.sender);


        // add bnox admins

        addBNOXAdmin(address(0x5638Dbf4cEa900Ca98ee34C8F1b5a22781d5c44b));

        setDestinationAccountWL(address(0x5638Dbf4cEa900Ca98ee34C8F1b5a22781d5c44b),
true);

        setSourceAccountWL(address(0x5638Dbf4cEa900Ca98ee34C8F1b5a22781d5c44b),
true);


        addBNOXAdmin(address(0xf3b9f8D148993A99A92935f5DAe1E35279491A86));

        setDestinationAccountWL(address(0xf3b9f8D148993A99A92935f5DAe1E35279491A86),
true);

        setSourceAccountWL(address(0xf3b9f8D148993A99A92935f5DAe1E35279491A86),
true);


        // add treasury admin
        addTreasuryAdmin(address(0x2EE2466ffE9bd134Cc92E3D3b43dae8892279D62));
        setDestinationAccountWL(address(0x2EE2466ffE9bd134Cc92E3D3b43dae8892279D62),
true);
        setSourceAccountWL(address(0x2EE2466ffE9bd134Cc92E3D3b43dae8892279D62),
true);

        // bsoFee
        setBsoFee(120);
        setBsopoolAddress(address(0x68A374a079a7c5FB41111371Fb2Ad4b528896D0d));
        setDestinationAccountWL(address(0x68A374a079a7c5FB41111371Fb2Ad4b528896D0d),
true);
        setSourceAccountWL(address(0x68A374a079a7c5FB41111371Fb2Ad4b528896D0d),
true);

        // generalFee
        setGeneralFee(80);
        setFeeAddress(address(0x82184BE9F135D345566D4D146fD93cBCacE96Afa));
        setDestinationAccountWL(address(0x82184BE9F135D345566D4D146fD93cBCacE96Afa),
true);
        setSourceAccountWL(address(0x82184BE9F135D345566D4D146fD93cBCacE96Afa),
true);

        //user 1
        setDestinationAccountWL(address(0x5a6b353e9AbeD3b162c559fde03fAf032bc8dE85),
true);
        setSourceAccountWL(address(0x5a6b353e9AbeD3b162c559fde03fAf032bc8dE85),
true);

        //user 2
        setDestinationAccountWL(address(0x9560876BF79a0a78DD7cBFA5B9fBB2e065983f91),
true);
        setSourceAccountWL(address(0x9560876BF79a0a78DD7cBFA5B9fBB2e065983f91),
true);

        //user 3
        setDestinationAccountWL(address(0x4e1f35776Cb0BF51190eC01c86eF8b7Bb629Ce62),
true);
```

```
        setSourceAccountWL(address(0x4e1f35776Cb0BF51190eC01c86eF8b7Bb629Ce62), 
true);


    }

}
```

www.kaspersky.com/
www.securelist.com