16 SEP 2020

ONBUFF SMART CONTRACT AUDIT REPORT

Hexlant.

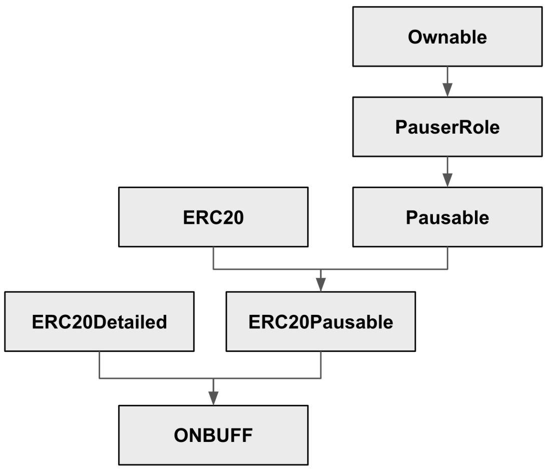
Analysis Purpose

This report details the audit conducted to (i) check whether the deployed codes meet the requirements or not, (ii) assess security vulnerabilities and potential threats in relations to the operation and (iii) devise solutions based upon the outcomes of this analysis.
This audit has been conducted to examine and evaluate the followings;

- Functionalities of implemented features
- Security vulnerabilities in operational environment
- Response and recovery toward the Off-Chain induced issues
- Readability and completeness of contract codes

Function Summary

Written with the contract codes provided by the Open-Zeppelin, ONBUFF's features are built upon the following contracts:



- **Ownable**
  It provides functions related to the ownership of the contract. Using onlyOwner Modifier, execution authority can be restricted to specific addresses.

- **Pausable**
  It provides functions related to the contract pause. If the contract is in pause state, the state can be restricted so that all token transfers cannot be made.

- **PauserRole**
  It provides functions related to contract pauser. Pauser can execute lock-up function and freeze function for specific addresses on ONBUFF contract.

- **ONBUFF**
  This is the main contract of ONBUFF. Essential functions such as lock-up, freeze, and upgrade are featured.

Contract

Used to express container-type contracts, including state variables and functions

Contract	Description
Ownable	Function related to contract ownership
PauserRole	Function related to contract pauser
Pausable	Function related to contract pause
ERC20	Function related to ERC20 standard interface
ERC20Pausable	Function related to token transfer regarding contract pause
ERC20Detailed	Function that provides basic token information
ONBUFF	Main function of ONBUFF

Interface

Used to define standard functions to be implemented in the contract

Interface	Description
IERC20	ERC20 standard interface

Library

As a contract library that cannot have state variables and does not support inheritance, functions within the library are called and executed in the context of the calling contract.

Library	Description
SafeMath	Control potential issues during arithmetic operations
Roles	Control authority on the contract

Variable

Variables expressing the state of the contract, used to store information necessary for the contract

Variable	Description
owner	Address of the contract owner
newOwner	Address of the contract's new owner
_pausers	Hash table of pauser's address
_paused	Contract in pause state
_balances	Token balance hash table of specific address
_allowed	Token balance hash table of specific address with withdrawal authority
_totalSupply	Total supply of token
_name	Token name
_symbol	Token symbol
_decimals	Representable Maximum decimal of token
implementation	Upgraded contract address
timelockList	List table of lockup data of specific address
frozenAccount	Hash table of whether a specific address is frozen

Modifier

As a limiting element of a function, it is used to allow execution only under limited conditions when performing a specific function.

Modifier	Description
onlyOwner	Executable only by contract owner
onlyNewOwner	Executable only by contract's new owner
onlyPauser	Executable only by contract pauser
whenNotPaused	Executable only when contract is not paused
whenPaused	Executable only when contract is paused
notFrozen	Executable when the specific address is not frozen

Event

Log event according to contract function execution, used to more easily respond to the contract situation in future application application

Event	Description
OwnershipTransferred	Event occurs when the contract ownership is transferred
PauserAdded	Event occurs when the contract pauser receives authority
PauserRemoved	Event occurs when the authority of contract pauser is removed
Paused	Event occurs when the contract is paused
Unpaused	Event occurs when the contract is unpaused
Transfer	Event occurs when transferring tokens
Approval	Event occurs when token withdrawal is approved
Freeze	Event occurs when the address is frozen
Unfreeze	Event occurs when the address is unfrozen
Lock	Event occurs when locked
Unlock	Event occurs when unlocked

Function

As contract functions, it is used to execute functions by containing specific logic necessary for the contract.

Function	Description
isOwner	Check the contract ownership
transferOwnership	Transfer the contract ownership
acceptOwnership	Accept the transfer of contract ownership
isPauser	Check whether it is the contract pauser
addPauser	Give contract pauser authority
removePauser	Remove contract pauser authority
renouncePauser	Renounce contract pauser authority
_addPauser	Add contract pauser authority
_removePauser	Remove contract pauser authority
paused	Check whether the contract is paused
pause	Switch the contract to paused status
unpause	Switch the contract to unpaused status
totalSupply	Check total supply of token
balanceOf	Check the token balance of a specific address
allowance	Check the token balance of a specific address with withdrawal approval

transfer	Transfer token
approve	Approve withdrawal
transferFrom	Transfer token with withdrawal approval
increaseAllowance	Increase allowance of tokens with withdrawal approval
decreaseAllowance	Decrease allowance of tokens with withdrawal approval
_transfer	Transfer token
_mint	Mint token
mint	Mint token
_burn	Burn token
_burnFrom	Burn tokens with withdrawal approval
name	Check token name
symbol	Check token symbol
decimals	Check representable maximum decimal of token
freezeAccount	Freeze specific address
unfreezeAccount	Unfreeze specific address
lock	Lock up balance of specific address
transferWithLock	Transfer locked tokens to specific address
unlock	Unlock the locked tokens of specific address
upgradeTo	Upgrade the contract address
_lock	Lock up balance of specific address
_unlock	Unlock the locked tokens of specific address
_autoUnlock	Unlock the locked tokens with passed expiration date
_setImplementation	Insert upgraded contract address

Function Profile

Function Profile describes the details of the contract functions such as parameters, options and call relationships among functions via the call stacks. With this, investigation of call relationships are conducted with logical conflicts tested.

Function Name	(Ownable) isOwner		
Parameter	address	Return	bool
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
isOwner			

Function Name	(Ownable) transferOwnership		
Parameter	address	Return	-
Visibility	public	Modifier	onlyOwner
Constant	-	Inheritance	-
Callstack			
transferOwnership			

Function Name	(Ownable) acceptOwnership		
Parameter	-	Return	bool
Visibility	public	Modifier	onlyNewOwner
Constant	-	Inheritance	-
Callstack			
acceptOwnership			

Function Name	(PausableRole) isPauser		
Parameter	address	Return	bool
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
isPauser			

Function Name	(PausableRole) addPauser		
Parameter	address	**Return**	-
Visibility	public	**Modifier**	onlyPauser
Constant	-	**Inheritance**	-
Callstack			
addPauser ⎿ _addPauser			

Function Name	(PausableRole) removePauser		
Parameter	address	**Return**	-
Visibility	public	**Modifier**	onlyOwner
Constant	-	**Inheritance**	-
Callstack			
removePauser ⎿ _removePauser			

Function Name	(PausableRole) renouncePauser		
Parameter	-	**Return**	-
Visibility	public	**Modifier**	-
Constant	-	**Inheritance**	-
Callstack			
renouncePauser ⎿_removePauser			

Function Name	(PausableRole) _addPauser		
Parameter	address	**Return**	-
Visibility	internal	**Modifier**	-
Constant	-	**Inheritance**	-
Callstack			
_addPauser			

Function Name	(PausableRole) _removePauser		
Parameter	address	**Return**	-
Visibility	internal	**Modifier**	-
Constant	-	**Inheritance**	-
Callstack			
removePauser			

Function Name	(Pausable) paused		
Parameter	-	Return	bool
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
paused			

Function Name	(Pausable) pause		
Parameter	-	Return	-
Visibility	public	Modifier	onlyPauser whenNotPaused
Constant	-	Inheritance	-
Callstack			
decimals			

Function Name	(Pausable) unpause		
Parameter	-	Return	-
Visibility	public	Modifier	onlyPauser whenPaused
Constant	-	Inheritance	-
Callstack			
unpause			

Function Name	(ERC20) totalSupply		
Parameter	-	Return	uint256
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
totalSupply			

Function Name	(ERC20) balanceOf		
Parameter	address	Return	uint256
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
balanceOf			

Function Name	(ERC20) allowance		
Parameter	address, address	Return	uint256
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
allowance			

Function Name	(ERC20) transfer		
Parameter	address, uint256	Return	bool
Visibility	public	Modifier	-
Constant	-	Inheritance	-
Callstack			
transfer ⎣_transfer			

Function Name	(ERC20) approve		
Parameter	address, uint256	Return	bool
Visibility	public	Modifier	-
Constant	-	Inheritance	-
Callstack			
approve			

Function Name	(ERC20) transferFrom		
Parameter	address, address, uint256	Return	bool
Visibility	public	Modifier	-
Constant	-	Inheritance	-
Callstack			
transferFrom ⎣ _transfer			

Function Name	(ERC20) increaseAllowance		
Parameter	address, uint256	Return	bool
Visibility	public	Modifier	-
Constant	-	Inheritance	-
Callstack			
increaseAllowance			

Function Name	(ERC20) decreaseAllowance		
Parameter	address, uint256	Return	bool
Visibility	public	Modifier	-
Constant	-	Inheritance	-
Callstack			
decreaseAllowance			

Function Name	(ERC20) _transfer		
Parameter	address, address, uint256	Return	bool
Visibility	public	Modifier	-
Constant	-	Inheritance	-
Callstack			
_transfer			

Function Name	(ERC20) _mint		
Parameter	address, uint256	Return	-
Visibility	internal	Modifier	-
Constant	-	Inheritance	-
Callstack			
_mint			

Function Name	(ERC20) _burn		
Parameter	address, uint256	Return	-
Visibility	internal	Modifier	-
Constant	-	Inheritance	-
Callstack			
_burn			

Function Name	(ERC20) _burnFrom		
Parameter	address, uint256	Return	-
Visibility	internal	Modifier	-
Constant	-	Inheritance	-
Callstack			
_burnFrom └_burn			

Function Name	(ERC20Pausable) transfer		
Parameter	address, uint256	Return	bool
Visibility	public	Modifier	whenNotPaused
Constant	-	Inheritance	-
Callstack			
transfer ⎣super.transfer			

Function Name	(ERC20Pausable) transferFrom		
Parameter	address, address, uint256	Return	bool
Visibility	public	Modifier	whenNotPaused
Constant	-	Inheritance	-
Callstack			
transferFrom ⎣ super.transferFrom			

Function Name	(ERC20Detailed) name		
Parameter	-	Return	string
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
name			

Function Name	(ERC20Detailed) symbol		
Parameter	-	Return	string
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
symbol			

Function Name	(ERC20Detailed) decimals		
Parameter	-	Return	uint8
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
decimals			

Function Name	(ONBUFF) balanceOf		
Parameter	address	**Return**	uint256
Visibility	public	**Modifier**	-
Constant	view	**Inheritance**	-
Callstack			
balanceOf ⎣super.balanceOf			

Function Name	(ONBUFF) transfer		
Parameter	address, uint256	**Return**	bool
Visibility	public	**Modifier**	notFrozen
Constant	-	**Inheritance**	-
Callstack			
transfer ⎣_autoUnlock ⎣super.transfer			

Function Name	(ONBUFF) transferFrom		
Parameter	address, address, uint256	**Return**	bool
Visibility	public	**Modifier**	notFrozen
Constant	-	**Inheritance**	-
Callstack			
transferFrom ⎣_autoUnlock ⎣super.transferFrom			

Function Name	(ONBUFF) freezeAccount		
Parameter	address	**Return**	bool
Visibility	public	**Modifier**	onlyPauser
Constant	-	**Inheritance**	-
Callstack			
freezeAccount			

Function Name	(ONBUFF) unfreezeAccount		
Parameter	address	**Return**	bool
Visibility	public	**Modifier**	onlyPauser
Constant	-	**Inheritance**	-
Callstack			
unfreezeAccount			

Function Name	(ONBUFF) lock		
Parameter	address, uint256, uin256	Return	bool
Visibility	public	Modifier	onlyPauser
Constant	-	Inheritance	-
Callstack			
lock └_lock			

Function Name	(ONBUFF) transferWithLock		
Parameter	address, uint256, uint256	Return	bool
Visibility	public	Modifier	onlyPauser
Constant	-	Inheritance	-
Callstack			
transferWithLock └_transfer └_lock			

Function Name	(ONBUFF) unlock		
Parameter	address, uint256	Return	bool
Visibility	public	Modifier	onlyPauser
Constant	-	Inheritance	-
Callstack			
unlock └_unlock			

Function Name	(ONBUFF) _lock		
Parameter	address, uint256, uint256	Return	bool
Visibility	internal	Modifier	-
Constant	-	Inheritance	-
Callstack			
_lock			

Function Name	(ONBUFF) _unlock		
Parameter	address, uint256	Return	bool
Visibility	internal	Modifier	-
Constant	-	Inheritance	-
Callstack			
_unlock			

Function Name	(ONBUFF) _autoUnlock		
Parameter	address	**Return**	bool
Visibility	internal	**Modifier**	-
Constant	-	**Inheritance**	-
Callstack			
_autoUnlock			

Function Name	(ONBUFF) upgradeTo		
Parameter	address	**Return**	-
Visibility	-	**Modifier**	onlyOwner
Constant	-	**Inheritance**	-
Callstack			
upgradeTo			

Function Name	(ONBUFF) _setImplementation		
Parameter	address	**Return**	-
Visibility	internal	**Modifier**	-
Constant	-	**Inheritance**	-
Callstack			
_setImplementation			

Test Result

Code Coverage

Code Coverage is a quantitative indicator of this report in relations to the testings of the BASIC contract codes' functions.

In the ONBUFF contract, there are cases where no additional calls are made to the library and functions implemented in some contracts.

File Name	Statements	Functions	Lines
ONBUFF.sol	100% (128/128)	100% (54/54)	100% (135/135)

Test cases

Test case	Result
Name of token matches with the assigned name.	PASS
Symbol of token matches with the assigned symbol.	PASS
Decimals of token matches with the assigned decimals.	PASS
Designated initial issuance volume is assigned as the total issuance volume.	PASS
Designated initial issuance volume is assigned to contract owner (address executed the distribution).	PASS
After distribution, all addresses except the owner's address have 0 token balance.	PASS
When transferring tokens, does it make an exception if the receiving address is 0?	PASS
Does it make an exception when it is attempted to transfer a negative amount?	PASS
Does it make an exception when it is attempted to transfer more that the amount in possession?	PASS
If approval is granted to a specific address, does the approved token balance of the address correctly increase?	PASS
Can the quantity of tokens with approval be increased or reduced?	PASS
Is it possible to transfer the tokens with withdrawal approval?	PASS
When transferring tokens with approval, does the balance of related addresses correctly reflect the changes?	PASS
When transferring tokens with approval, does it make an exception if the receiving address is 0?	PASS
Does it make an exception when it is attempted to send more tokens with approval than the amount in possession?	PASS
Does it make an exception when the token balance of the addresses with approval is insufficient?	PASS
Is it possible to verify the contract ownership?	PASS

Test case	Result
Does it make an exception when someone other than the contract owner attempts to transfer the contract ownership?	PASS
Is it possible for the contract owner to transfer the contract ownership to other address?	PASS
Can the address receiving the contract ownership accept the transfer?	PASS
After the transferring the contract ownership, does the new owner's address become 0x0?	PASS
Does it make an exception if addresses other than contract owner attempts to pause the contract?	PASS
Does it make an exception when sending tokens in pause status?	PASS
Does it make an exception when sending tokens with withdrawal approval while the contract is in pause status?	PASS
Does it make an exception when address other than the contract owner attempts to unpause the contract?	PASS
Can the contract pauser unpause the contract?	PASS
Does it make an exception when address other than contract pauser attempts to send locked tokens?	PASS
Does it make an exception when address other than contract pauser attempts to lock up the tokens possessed by the specific address?	PASS
Does it make an exception when address other than contract pauser attempts to unlock the locked tokens?	PASS
Can the contract pauser lock up the tokens possessed by the specific address?	PASS
Can the contract pauser transfer the locked tokens to the specific address?	PASS
Can the contract pauser unlock the locked tokens of the specific address?	PASS
Does it make an exception when transferring tokens that has not passed the expiration period?	PASS
Does it make an exception when transferring tokens (not passed the expiration period) through withdrawal approval?	PASS
Does it make an exception when address other than the contract pauser attempts to freeze the specific address?	PASS
Does it make an exception when address other than the contract pauser attempts to unfreeze the frozen address?	PASS
Can the contract pauser freeze the specific address?	PASS
Can the contract pauser unfreeze the specific address that is frozen?	PASS
Does it make an exception when transferring tokens from frozen address?	PASS
Does it make an exception when transferring tokens with withdrawal approval from the frozen address?	PASS
Does it make an exception when address other than the contract pauser attempts to grant pauser authority to the specific address?	PASS
Does it make an exception when address other than the contract pauser attempts to remove pauser authority from the specific address?	PASS
Can the contract pauser grant pauser authority to the specific address?	PASS

Test case	Result
Can the contract pauser remove pauser authority from the specific address?	PASS
Can the contract pauser renounce his own authority?	PASS
Does the token transfer event occur when distributing the contract?	PASS
Does an event occur when transferring token?	PASS
Does an event occur when approving withdrawal?	PASS
Does an event occur when tokens with withdrawal approval increase or decrease?	PASS
Does an event occur when transferring tokens with withdrawal approval?	PASS
Does an event occur when transferring contract ownership?	PASS
Does an event occur when granting or renouncing contract pauser authority?	PASS

Vulnerability Analysis

Critical Severity

Severity is a metric for classifying the level of risk which a security vulnerability poses. Severity level classified as Critical poses a serious threat to the securities within the normal operating environment parameters and must be fixed.

Not Applicable

High Severity

Severity level classified as High poses a threat to the securities only outside of normal operating parameters when certain conditions are met. It is crucial to analyze these parameters or corner cases to make revisions necessary to prevent errors .

Not Applicable

Medium Severity

Severity level classified as Medium does not pose a serious threat to the securities but is recommended to revise for more efficient functioning.

Not Applicable

Low Severity

Severity level classified as Low does not pose a threat to the securities or function but is recommended to revise for better code readability or structural efficiency .

Not Applicable

Conclusion

ONBUFF contract follows the ERC-20 interface with added features such as lockup, freeze, mint, and upgrade. Having the contract pauser authority allows the restriction of token transfers in the ecosystem and freezing of the specific address.

The implemented functions on the contract are well-designed and straightforward; thus, no significant issues are found.

Declare

This audit report complements Hexlant's Smart Contract security audit results. The scope of this report does not include the feasibility of the business model, legal fitness or investment advice. Other unknown security vulnerabilities such as main-net or virtual machines are beyond the auditing responsibilities of this report.

```
/**
 *Submitted for verification at Etherscan.io on 2020-07-17
*/

pragma solidity ^0.5.0;

library SafeMath {
    /**
    * @dev Multiplies two unsigned integers, reverts on 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-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
    * @dev Integer division of two unsigned integers truncating the quotient, reverts on division by zero.
    */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
    * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend is greater than
minuend).
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
    * @dev Adds two unsigned integers, reverts on overflow.
    */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
    * @dev Divides two unsigned integers and returns the remainder (unsigned integer modulo),
    * reverts when dividing by zero.
    */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}

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(account != address(0));
        require(!has(role, account));

        role.bearer[account] = true;
    }
    /**
     * @dev remove an account's access to this role
     */
    function remove(Role storage role, address account) internal {
        require(account != address(0));
        require(has(role, account));

        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));
        return role.bearer[account];
    }
}

contract Ownable {
    address public owner;
    address public newOwner;

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

    constructor() public {
        owner = msg.sender;
        newOwner = address(0);
    }

    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }
    modifier onlyNewOwner() {
        require(msg.sender != address(0));
        require(msg.sender == newOwner);
        _;
    }

    function isOwner(address account) public view returns (bool) {
        if( account == owner ){
            return true;
        }
        else {
            return false;
        }
    }

    function transferOwnership(address _newOwner) public onlyOwner {
        require(_newOwner != address(0));
        newOwner = _newOwner;
    }

    function acceptOwnership() public onlyNewOwner returns(bool) {
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
        newOwner = address(0);
    }
}

contract PauserRole is Ownable{
    using Roles for Roles.Role;

    event PauserAdded(address indexed account);
    event PauserRemoved(address indexed account);

    Roles.Role private _pausers;

    constructor () internal {
        _addPauser(msg.sender);
    }

    modifier onlyPauser() {
        require(isPauser(msg.sender)|| isOwner(msg.sender));
        _;
    }

    function isPauser(address account) public view returns (bool) {
        return _pausers.has(account);
    }
    function addPauser(address account) public onlyPauser {
        _addPauser(account);
    }

    function removePauser(address account) public onlyOwner {
        _removePauser(account);
    }

    function renouncePauser() public {
        _removePauser(msg.sender);
    }

    function _addPauser(address account) internal {
        _pausers.add(account);
        emit PauserAdded(account);
    }

    function _removePauser(address account) internal {
        _pausers.remove(account);
        emit PauserRemoved(account);
    }
}

contract Pausable is PauserRole {
    event Paused(address account);
    event Unpaused(address account);

    bool private _paused;

    constructor () internal {
        _paused = false;
    }

    /**
     * @return true if the contract is paused, false otherwise.
     */
    function paused() public view returns (bool) {
        return _paused;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
     */
    modifier whenNotPaused() {
        require(!_paused);
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     */
    modifier whenPaused() {
        require(_paused);
        _;
    }

    /**
     * @dev called by the owner to pause, triggers stopped state
     */
    function pause() public onlyPauser whenNotPaused {
        _paused = true;
        emit Paused(msg.sender);
    }

    /**
     * @dev called by the owner to unpause, returns to normal state
     */
    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(msg.sender);
    }
}


interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external returns (bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);
    function allowance(address owner, address spender) external view returns (uint256);

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) internal _balances;

    mapping (address => mapping (address => uint256)) internal _allowed;

    uint256 private _totalSupply;

    /**
    * @dev Total number of tokens in existence
    */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
    * @dev Gets the balance of the specified address.
    * @param owner The address to query the balance of.
    * @return An uint256 representing the amount owned by the passed address.
    */
    function balanceOf(address owner) public view returns (uint256) {
        return _balances[owner];
    }

    /**
     * @dev Function to check the amount of tokens that an owner allowed to a spender.
     * @param owner address The address which owns the funds.
     * @param spender address The address which will spend the funds.
     * @return A uint256 specifying the amount of tokens still available for the spender.
     */
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowed[owner][spender];
    }

    /**
    * @dev Transfer token for a specified address
    * @param to The address to transfer to.
    * @param value The amount to be transferred.
    */
    function transfer(address to, uint256 value) public returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
    }

    /**
     * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
     * 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
     * @param spender The address which will spend the funds.
     * @param value The amount of tokens to be spent.
     */
    function approve(address spender, uint256 value) public returns (bool) {
        require(spender != address(0));

        _allowed[msg.sender][spender] = value;
        emit Approval(msg.sender, spender, value);
        return true;
    }

    /**
     * @dev Transfer tokens from one address to another.
     * Note that while this function emits an Approval event, this is not required as per the
specification,
     * and other compliant implementations may not emit the event.
     * @param from address The address which you want to send tokens from
     * @param to address The address which you want to transfer to
     * @param value uint256 the amount of tokens to be transferred
     */
    function transferFrom(address from, address to, uint256 value) public returns (bool) {
        _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
        _transfer(from, to, value);
        emit Approval(from, msg.sender, _allowed[from][msg.sender]);
        return true;
    }

    /**
     * @dev Increase the amount of tokens that an owner allowed to a spender.
     * approve should be called when allowed_[_spender] == 0. To increment
     * allowed value is better to use this function to avoid 2 calls (and wait until
     * the first transaction is mined)
     * From MonolithDAO Token.sol
     * Emits an Approval event.
     * @param spender The address which will spend the funds.
     * @param addedValue The amount of tokens to increase the allowance by.
     */
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        require(spender != address(0));

        _allowed[msg.sender][spender] = _allowed[msg.sender][spender].add(addedValue);
        emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
        return true;
    }

    /**
     * @dev Decrease the amount of tokens that an owner allowed to a spender.
     * approve should be called when allowed_[_spender] == 0. To decrement
     * allowed value is better to use this function to avoid 2 calls (and wait until
     * the first transaction is mined)
     * From MonolithDAO Token.sol
     * Emits an Approval event.
     * @param spender The address which will spend the funds.
     * @param subtractedValue The amount of tokens to decrease the allowance by.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        require(spender != address(0));

        _allowed[msg.sender][spender] = _allowed[msg.sender][spender].sub(subtractedValue);
        emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
        return true;
    }

    /**
    * @dev Transfer token for a specified addresses
    * @param from The address to transfer from.
    * @param to The address to transfer to.
    * @param value The amount to be transferred.
    */
    function _transfer(address from, address to, uint256 value) internal {
        require(to != address(0));

        _balances[from] = _balances[from].sub(value);
        _balances[to] = _balances[to].add(value);
        emit Transfer(from, to, value);
    }

    /**
     * @dev Internal function that mints an amount of the token and assigns it to
     * an account. This encapsulates the modification of balances such that the
     * proper events are emitted.
     * @param account The account that will receive the created tokens.
     * @param value The amount that will be created.
     */
    function _mint(address account, uint256 value) internal {
        require(account != address(0));

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

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account.
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    function _burn(address account, uint256 value) internal {
        require(account != address(0));

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }
    /**
     * @dev Internal function that burns an amount of the token of a given
     * account, deducting from the sender's allowance for said account. Uses the
     * internal burn function.
     * Emits an Approval event (reflecting the reduced allowance).
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    function _burnFrom(address account, uint256 value) internal {
        _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(value);
        _burn(account, value);
        emit Approval(account, msg.sender, _allowed[account][msg.sender]);
    }
}



contract ERC20Pausable is ERC20, Pausable {
    function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transfer(to, value);
    }

    function transferFrom(address from, address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transferFrom(from, to, value);
    }

    /*
     * approve/increaseApprove/decreaseApprove can be set when Paused state
     */

    /*
     * function approve(address spender, uint256 value) public whenNotPaused returns (bool) {
     *     return super.approve(spender, value);
     * }
     *
     * function increaseAllowance(address spender, uint addedValue) public whenNotPaused returns (bool
success) {
     *     return super.increaseAllowance(spender, addedValue);
     * }
     *
     * function decreaseAllowance(address spender, uint subtractedValue) public whenNotPaused returns
(bool success) {
     *     return super.decreaseAllowance(spender, subtractedValue);
     * }
     */
}

contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }

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

    /**
     * @return the symbol of the token.
     */
    function symbol() public view returns (string memory) {
        return _symbol;
    }
    /**
     * @return the number of decimals of the token.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

contract ONBUFF is ERC20Detailed, ERC20Pausable {

    struct LockInfo {
        uint256 _releaseTime;
        uint256 _amount;
    }
    address public implementation;

    mapping (address => LockInfo[]) public timelockList;
       mapping (address => bool) public frozenAccount;

    event Freeze(address indexed holder);
    event Unfreeze(address indexed holder);
    event Lock(address indexed holder, uint256 value, uint256 releaseTime);
    event Unlock(address indexed holder, uint256 value);

    modifier notFrozen(address _holder) {
        require(!frozenAccount[_holder]);
        _;
    }

    constructor() ERC20Detailed("ONBUFF", "ONIT",18) public  {

        _mint(msg.sender, 1000000000 * (10 ** 18));
    }

    function balanceOf(address owner) public view returns (uint256) {

        uint256 totalBalance = super.balanceOf(owner);
        if( timelockList[owner].length >0 ){
            for(uint i=0; i<timelockList[owner].length;i++){
                totalBalance = totalBalance.add(timelockList[owner][i]._amount);
            }
        }

        return totalBalance;
    }

    function transfer(address to, uint256 value) public notFrozen(msg.sender) returns (bool) {
        if (timelockList[msg.sender].length > 0 ) {
            _autoUnlock(msg.sender);
        }
        return super.transfer(to, value);
    }

    function transferFrom(address from, address to, uint256 value) public notFrozen(from) returns (bool) {
        if (timelockList[from].length > 0) {
            _autoUnlock(from);
        }
        return super.transferFrom(from, to, value);
    }

    function freezeAccount(address holder) public onlyPauser returns (bool) {
        require(!frozenAccount[holder]);
        frozenAccount[holder] = true;
        emit Freeze(holder);
        return true;
    }

    function unfreezeAccount(address holder) public onlyPauser returns (bool) {
        require(frozenAccount[holder]);
        frozenAccount[holder] = false;
        emit Unfreeze(holder);
        return true;
    }

    function lock(address holder, uint256 value, uint256 releaseTime) public onlyPauser returns (bool) {
        require(_balances[holder] >= value,"There is not enough balances of holder.");
        _lock(holder,value,releaseTime);


        return true;
    }

    function transferWithLock(address holder, uint256 value, uint256 releaseTime) public onlyPauser
returns (bool) {
        _transfer(msg.sender, holder, value);
        _lock(holder,value,releaseTime);
        return true;
    }

    function unlock(address holder, uint256 idx) public onlyPauser returns (bool) {
        require( timelockList[holder].length > idx, "There is not lock info.");
        _unlock(holder,idx);
        return true;
    }
    /**
     * @dev Upgrades the implementation address
     * @param _newImplementation address of the new implementation
     */
    function upgradeTo(address _newImplementation) public onlyOwner {
        require(implementation != _newImplementation);
        _setImplementation(_newImplementation);
    }

    function _lock(address holder, uint256 value, uint256 releaseTime) internal returns(bool) {
        _balances[holder] = _balances[holder].sub(value);
        timelockList[holder].push( LockInfo(releaseTime, value) );

        emit Lock(holder, value, releaseTime);
        return true;
    }

    function _unlock(address holder, uint256 idx) internal returns(bool) {
        LockInfo storage lockinfo = timelockList[holder][idx];
        uint256 releaseAmount = lockinfo._amount;

        delete timelockList[holder][idx];
        timelockList[holder][idx] = timelockList[holder][timelockList[holder].length.sub(1)];
        timelockList[holder].length -=1;

        emit Unlock(holder, releaseAmount);
        _balances[holder] = _balances[holder].add(releaseAmount);

        return true;
    }

    function _autoUnlock(address holder) internal returns(bool) {
        for(uint256 idx =0; idx < timelockList[holder].length ; idx++ ) {
            if (timelockList[holder][idx]._releaseTime <= now) {
                // If lockupinfo was deleted, loop restart at same position.
                if( _unlock(holder, idx) ) {
                    idx -=1;
                }
            }
        }
        return true;
    }

    function mint(uint256 value) public onlyOwner returns(bool) {
       _mint(msg.sender, value);
       return true;
    }

    /**
     * @dev Sets the address of the current implementation
     * @param _newImp address of the new implementation
     */
    function _setImplementation(address _newImp) internal {
        implementation = _newImp;
    }

    /**
     * @dev Fallback function allowing to perform a delegatecall
     * to the given implementation. This function will return
     * whatever the implementation call returns
     */
    function () payable external {
        address impl = implementation;
        require(impl != address(0));
        assembly {
            let ptr := mload(0x40)
            calldatacopy(ptr, 0, calldatasize)
            let result := delegatecall(gas, impl, ptr, calldatasize, 0, 0)
            let size := returndatasize
            returndatacopy(ptr, 0, size)

            switch result
            case 0 { revert(ptr, size) }
            default { return(ptr, size) }
        }
    }
}
```

Hexlant.
Blockchain Lab
-
contact@hexlant.com
www.hexlant.com

HEXLANT CONTRACT CERTIFICATION

This contract speicifics that it has been validated by the Hexlant Technical Team and notifies that it has not any technical defects.

PUBLISHIED INFORMATION

REPORT NUMBER	ERC20200916
DATE	2020/09/16
PUBLISHIER	SEONGEUN CHO eun@hexlant.com

TOKEN INFORMATION

TOKEN NAME	ONBUFF		
SYMBOL	ONIT		
PLATFORM	ETHEREUM	**TOKEN TYPE**	ERC-20
TOTAL SUPPLY	1,000,000,000 ONBUFF		
CONTRACT ADDRESS	0x2716bdb7d96c43c8ef3b120eba43f6ca4a814217		

VULNERABLILLITY ANALYSIS

CRITICAL	0	No relevant provision
HIGH	0	No relevant provision
MEDIUM	0	No relevant provision
LOW	0	No relevant provision

CENTRALIZED FUNCTIONS

FREEZE	YES	Ability to freeze tokens in accounts. (The administrator can freeze the hacker's account in case of hacking.)
PAUSE	YES	Ability to pause functions related to token transmission in a contract. (This is used when the administrator needs to prevent the movement of assets due to token swaps or hacking.)
LOCKUP	YES	Ability to block token transfers for a period of time (Administrators can use to set lockout periods for investors, team members, advisors, etc.)
BURN	NO	Ability to reduce total supply by burning tokens
MINT	YES	Ability to increase total supply by minting tokens

Certified by Hexlant. ____________