Smart Contract Security Audit Report

2020.06.01

DISCLAIMER

The scope of this smart contract audit only covers smart contract code explicitly specified in this report. It does not represent a security analysis of any other contracts, employed software, or any other operational security of the company presenting the smart contract.

The audit makes no warranties or statements about the utility of the code, safety of the code, its commercial applications, viability of the business model, and legal compliance of the code in any jurisdiction. THIS DOCUMENT IS NOT AN INVESTMENT ADVICE. No statements or claims are being made about the fitness of the smart contracts for any purpose, or their bug free status.

Introduction

Center Coin Team engaged BLOBS to perform smart contract audit for the CenterCoin(CENT) project. The objective of the audit was to evaluate quality and security of the CenterCoin(CENT) smart contracts. Audit took place between may 1, 2020 and jun 1, 2020 in Seoul, Korea and was performed by the BLOBS engineering team.

Audit Methodology

What did we audit?

BLOBS has performed an initial review and thorough review of the smart contract code deployed to https://ropsten.etherscan.io

We audited the following smart contracts files containing smart contracts

- CenterCoin(CENT)

File	SHA1 hash
CenterCoin.sol	0x262b6a8b044ab9b7a345353ded8f541c1ac37bd3

How did we perform the audit?

BLOBS has followed best practices and industry-standard techniques to verify the CenterCoin(CENT)

smart contracts, namely:

- We performed manual review of the code-base line by line.
- We performed automated tests.
- Two separate engineers performed the audit and we crosschecked the results.
- We tested the underlying smart contracts against common attack vectors.
- We deployed the smart contracts on the Ethereum testnet and performed live tests

Smart Contract Structure

CenterCoin

This smart contract is an ERC20 compliant token. The token is free from reentrancy bugs

Coverage Report

The following table shows coverage statistics for each smart contract audited.

Smart Contract	Statements	Branches	Functions	Lines
CenterCoin	100%	100%	100%	100%

Issues and Vulnerabilities

Audit did not reveal any severe vulnerabilities or security issues with none of the audited smart contracts.

Conclusion

CenterCoin(CENT) smart contracts are well written and the analysis shows that they do not contain critical vulnerabilities.

BLOBS Co.,Ltd.

Source Code

```
/**
 *Submitted for verification at Etherscan.io on 2020-01-10
*/

pragma solidity 0.5.8;

// File: node_modules\openzeppelin-solidity\contracts\token\ERC20\IERC20.sol

/**
 * @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.
     *
     * > 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);
        }

        // File: node_modules\openzeppelin-solidity\contracts\math\SafeMath.sol

        /**
         * @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) {
                require(b <= a, "SafeMath: subtraction overflow");
                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-solidity/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) {
                // Solidity only automatically asserts when dividing by 0
                require(b > 0, "SafeMath: division by zero");
                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) {
                require(b != 0, "SafeMath: modulo by zero");
                return a % b;
            }
        }

        // File: node_modules\openzeppelin-solidity\contracts\token\ERC20\ERC20.sol

        /**
         * @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`.
 *
 * *For a detailed writeup see our guide [How to implement supply
 * mechanisms](https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-
mechanisms/226).*
 *
 * 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 IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) internal _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(msg.sender, 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 value) public returns (bool) {
        _approve(msg.sender, spender, value);
        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 `value`.
     * - 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, msg.sender, _allowances[sender][msg.sender].sub(amount));
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to `approve` that can be used as a mitigation for
     * problems described in `IERC20.approve`.
     *
     * Emits an `Approval` event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][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(msg.sender, spender,
_allowances[msg.sender][spender].sub(subtractedValue));
        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);
        _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 Destoys `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 value) internal {
        require(account != address(0), "ERC20: burn from the zero address");

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

    /**
     * @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 value) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");
```

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

        /**
         * @dev Destoys `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, msg.sender, _allowances[account][msg.sender].sub(amount));
        }
    }

    // File: contracts\CenterCoin.sol

    contract CenterCoin is ERC20 {
        string public constant name = "Center Coin";
        string public constant symbol = "CENT";
        uint8 public constant decimals = 18;
        uint256 public constant initialSupply = 5000000000 * (10 ** uint256(decimals));

        constructor() public {
            super._mint(msg.sender, initialSupply);
            owner = msg.sender;
        }

        //ownership
        address public owner;

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

        modifier onlyOwner() {
            require(msg.sender == owner, "Not owner");
            _;
        }

      /**
       * @dev Allows the current owner to relinquish control of the contract.
       * @notice Renouncing to ownership will leave the contract without an owner.
       * It will not be possible to call the functions with the `onlyOwner`
       * modifier anymore.
       */
        function renounceOwnership() public onlyOwner {
            emit OwnershipRenounced(owner);
            owner = address(0);
        }

      /**
       * @dev Allows the current owner to transfer control of the contract to a newOwner.
       * @param _newOwner The address to transfer ownership to.
       */
        function transferOwnership(address _newOwner) public onlyOwner {
            _transferOwnership(_newOwner);
        }

      /**
       * @dev Transfers control of the contract to a newOwner.
       * @param _newOwner The address to transfer ownership to.
       */
        function _transferOwnership(address _newOwner) internal {
            require(_newOwner != address(0), "Already owner");
            emit OwnershipTransferred(owner, _newOwner);
```

```
    owner = _newOwner;
}

//pausable
event Pause(uint256 _value);
event Unpause(uint256 _value);

bool public paused = false;

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

/**
* @dev Modifier to make a function callable only when the contract is paused.
*/
modifier whenPaused() {
    require(paused, "Not paused now");
    _;
}

/**
* @dev called by the owner to pause, triggers stopped state
*/
function pause(uint256 _value) public onlyOwner whenNotPaused {
    paused = true;
    emit Pause(_value);
}

/**
* @dev called by the owner to unpause, returns to normal state
*/
function unpause(uint256 _value) public onlyOwner whenPaused {
    paused = false;
    emit Unpause(_value);
}

//freezable
event Frozen(address target);
event Unfrozen(address target);

mapping(address => bool) internal freezes;

modifier whenNotFrozen() {
    require(!freezes[msg.sender], "Sender account is locked.");
    _;
}

function freeze(address _target) public onlyOwner {
    freezes[_target] = true;
    emit Frozen(_target);
}

function unfreeze(address _target) public onlyOwner {
    freezes[_target] = false;
    emit Unfrozen(_target);
}

function isFrozen(address _target) public view returns (bool) {
    return freezes[_target];
}

function transfer(
    address _to,
    uint256 _value
```

```
)
  public
  whenNotFrozen
  whenNotPaused
  returns (bool)
{
    releaseLock(msg.sender);
    return super.transfer(_to, _value);
}

function transferFrom(
    address _from,
    address _to,
    uint256 _value
)
  public
  whenNotPaused
  returns (bool)
{
    require(!freezes[_from], "From account is locked.");
    releaseLock(_from);
    return super.transferFrom(_from, _to, _value);
}

//mintable
event Mint(address indexed to, uint256 amount);

function mint(
    address _to,
    uint256 _amount
)
  public
  onlyOwner
  returns (bool)
{
    super._mint(_to, _amount);
    emit Mint(_to, _amount);
    return true;
}

//burnable
event Burn(address indexed burner, uint256 value);

function burn(address _who, uint256 _value) public onlyOwner {
    require(_value <= super.balanceOf(_who), "Balance is too small.");

    _burn(_who, _value);
    emit Burn(_who, _value);
}

//lockable
struct LockInfo {
    uint256 releaseTime;
    uint256 balance;
}
mapping(address => LockInfo[]) internal lockInfo;

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

function balanceOf(address _holder) public view returns (uint256 balance) {
    uint256 lockedBalance = 0;
    for(uint256 i = 0; i < lockInfo[_holder].length ; i++ ) {
        lockedBalance = lockedBalance.add(lockInfo[_holder][i].balance);
    }
    return super.balanceOf(_holder).add(lockedBalance);
}

function releaseLock(address _holder) internal {
```

```
                for(uint256 i = 0; i < lockInfo[_holder].length ; i++ ) {
                    if (lockInfo[_holder][i].releaseTime <= now) {
                        _balances[_holder] = _balances[_holder].add(lockInfo[_holder][i].balance);
                        emit Unlock(_holder, lockInfo[_holder][i].balance);
                        lockInfo[_holder][i].balance = 0;

                        if (i != lockInfo[_holder].length - 1) {
                            lockInfo[_holder][i] = lockInfo[_holder][lockInfo[_holder].length - 1];
                            i--;
                        }
                        lockInfo[_holder].length--;

                    }
                }
            }
            function lockCount(address _holder) public view returns (uint256) {
                return lockInfo[_holder].length;
            }
            function lockState(address _holder, uint256 _idx) public view returns (uint256,
uint256) {
                return (lockInfo[_holder][_idx].releaseTime, lockInfo[_holder][_idx].balance);
            }

            function lock(address _holder, uint256 _amount, uint256 _releaseTime) public onlyOwner
{
                require(super.balanceOf(_holder) >= _amount, "Balance is too small.");
                _balances[_holder] = _balances[_holder].sub(_amount);
                lockInfo[_holder].push(
                    LockInfo(_releaseTime, _amount)
                );
                emit Lock(_holder, _amount, _releaseTime);
            }

            function lockAfter(address _holder, uint256 _amount, uint256 _afterTime) public
onlyOwner {
                require(super.balanceOf(_holder) >= _amount, "Balance is too small.");
                _balances[_holder] = _balances[_holder].sub(_amount);
                lockInfo[_holder].push(
                    LockInfo(now + _afterTime, _amount)
                );
                emit Lock(_holder, _amount, now + _afterTime);
            }

            function unlock(address _holder, uint256 i) public onlyOwner {
                require(i < lockInfo[_holder].length, "No lock information.");

                _balances[_holder] = _balances[_holder].add(lockInfo[_holder][i].balance);
                emit Unlock(_holder, lockInfo[_holder][i].balance);
                lockInfo[_holder][i].balance = 0;

                if (i != lockInfo[_holder].length - 1) {
                    lockInfo[_holder][i] = lockInfo[_holder][lockInfo[_holder].length - 1];
                }
                lockInfo[_holder].length--;
            }

            function transferWithLock(address _to, uint256 _value, uint256 _releaseTime) public
onlyOwner returns (bool) {
                require(_to != address(0), "wrong address");
                require(_value <= super.balanceOf(owner), "Not enough balance");

                _balances[owner] = _balances[owner].sub(_value);
                lockInfo[_to].push(
                    LockInfo(_releaseTime, _value)
                );
                emit Transfer(owner, _to, _value);
                emit Lock(_to, _value, _releaseTime);
```

```
                return true;
            }

            function transferWithLockAfter(address _to, uint256 _value, uint256 _afterTime) public
onlyOwner returns (bool) {
                require(_to != address(0), "wrong address");
                require(_value <= super.balanceOf(owner), "Not enough balance");

                _balances[owner] = _balances[owner].sub(_value);
                lockInfo[_to].push(
                    LockInfo(now + _afterTime, _value)
                );
                emit Transfer(owner, _to, _value);
                emit Lock(_to, _value, now + _afterTime);

                return true;
            }

            function currentTime() public view returns (uint256) {
                return now;
            }

            function afterTime(uint256 _value) public view returns (uint256) {
                return now + _value;
            }
        }
```

Home Page: https://blobs.kr

E-mail : support@blobs.kr