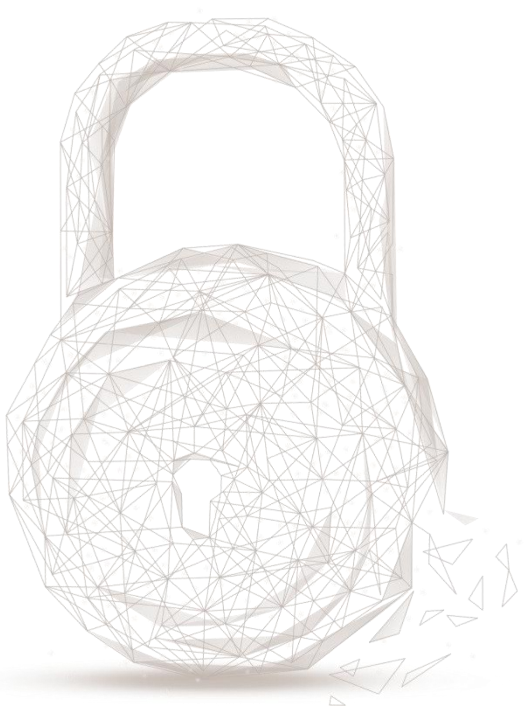
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202011181334

Smart Contract Name：

BonFi (BNF)

Smart Contract Address：

0x1DE5e000C41C8d35b9f1f4985C23988f05831057

Smart Contract Address Link：

https://etherscan.io/address/0x1DE5e000C41C8d35b9f1f4985C23988f05831057#code

Start Date：2020.11.18

Completion Date：2020.11.18

Overall Result：Pass（Distinction）

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	ERC20 Token Standards	Pass
		Compiler Version Security	Pass
		Visibility Specifiers	Pass
		Gas Consumption	Pass
		SafeMath Features	Pass
		Fallback Usage	Pass
		tx.origin Usage	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		Overriding Variables	Pass
2	Function Call Audit	Authorization of Function Call	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		selfdestruct Function Security	Pass

3	Business Security	Access Control of Owner	Pass
		Business Logics	Pass
		Business Implementations	Pass
4	Integer Overflow/Underflow	-	Pass
5	Reentrancy	-	Pass
6	Exceptional Reachable State	-	Pass
7	Transaction-Ordering Dependence	-	Pass
8	Block Properties Dependence	-	Pass
9	Pseudo-random Number Generator (PRNG)	-	Pass
10	DoS (Denial of Service)	-	Pass
11	Token Vesting Implementation	-	N/A
12	Fake Deposit	-	Pass
13	event security	-	Pass

Note: Audit results and suggestions in code comments

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract BNF, including Coding Standards, Security, and Business Logic. **BNF contract passed all audit items. The overall result is Pass (Distinction). The smart contract is able to function properly.** Please find below the basic information of the smart contract:

1、Basic Token Information

Token name	BonFi
Token symbol	BNF
decimals	18
totalSupply	Initial total supply is 1 billion (Burnable)
Token type	ERC20

Table 1 – Basic Token Information

2、Token Vesting Information

N/A

Audited Source Code with Comments:

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-08
*/

// File: contracts\open-zeppelin-contracts\token\ERC20\IERC20.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see `ERC20Detailed`.
 */
// Beosin (Chengdu LianAn) // Define the function interfaces required by ERC20 Token standard.
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); // Beosin (Chengdu
LianAn) // Declare the events 'Transfer'.

    /**
     * @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); // Beosin
(Chengdu LianAn) // Declare the events 'Approval'.
}

// File: contracts\open-zeppelin-contracts\math\SafeMath.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
 * @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.
 */
// Beosin (Chengdu LianAn) // The SafeMath library declares functions for safe mathematical
operation.
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: contracts\open-zeppelin-contracts\token\ERC20\ERC20.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
 * @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; // Beosin (Chengdu LianAn) // Use the SafeMath library for
mathematical operation. Avoid integer overflow/underflow.

    mapping (address => uint256) private _balances; // Beosin (Chengdu LianAn) // Declare the
mapping variable '_balances' for storing the token balance of corresponding address.

    mapping (address => mapping (address => uint256)) private _allowances; // Beosin (Chengdu
LianAn) // Declare the mapping variable '_allowances' for storing the allowance between two
addresses.

    uint256 private _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable
'_totalSupply' for storing the total token supply.

    /**
     * @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); // Beosin (Chengdu LianAn) // Call the
internal function '_transfer' to transfer tokens.
        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.
     */
    // Beosin (Chengdu LianAn) // 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.
    // Beosin (Chengdu LianAn) // Using function 'increaseAllowance' and 'decreaseAllowance' to alter allowance is recommended.
    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to set the allowance which 'msg.sender' allowed to 'spender'.
        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); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' to transfer tokens.
        _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount)); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to alter the allowance between two addresses.
        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)); //
Beosin (Chengdu LianAn) // Call the internal function '_approve' to increase the allowance
between two addresses.
            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));
// Beosin (Chengdu LianAn) // Call the internal function '_approve' to decrease the allowance
between two addresses.
            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"); // Beosin (Chengdu LianAn) // The non-zero address check for 'sender'.
        require(recipient != address(0), "ERC20: transfer to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'recipient'. Avoid losing transferred tokens.

        _balances[sender] = _balances[sender].sub(amount); // Beosin (Chengdu LianAn) // Alter the token balance of 'sender'.
        _balances[recipient] = _balances[recipient].add(amount); // Beosin (Chengdu LianAn) // Alter the token balance of 'recipient'.
        emit Transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }

    /** @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"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.

        _totalSupply = _totalSupply.add(amount); // Beosin (Chengdu LianAn) // Update the total token supply.
        _balances[account] = _balances[account].add(amount); // Beosin (Chengdu LianAn) // Alter the token balance of 'account'.
        emit Transfer(address(0), account, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }

     /**
     * @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 value) internal {
            require(account != address(0), "ERC20: burn from the zero address"); // Beosin (Chengdu
LianAn) // The non-zero address check for 'account'.

            _totalSupply = _totalSupply.sub(value); // Beosin (Chengdu LianAn) // Update the total
token supply.
            _balances[account] = _balances[account].sub(value); // Beosin (Chengdu LianAn) //
Alter the token balance of 'account'.
            emit Transfer(account, address(0), value); // Beosin (Chengdu LianAn) // Trigger the
event 'Transfer'.
        }

        /**
         * @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"); // Beosin
(Chengdu LianAn) // The non-zero address check for 'owner'.
            require(spender != address(0), "ERC20: approve to the zero address"); // Beosin (Chengdu
LianAn) // The non-zero address check for 'spender'.

            _allowances[owner][spender] = value; // Beosin (Chengdu LianAn) // The allowance
which 'owner' allowed to 'spender' is set to 'value'.
            emit Approval(owner, spender, value); // Beosin (Chengdu LianAn) // Trigger the event
'Approval'.
        }

        /**
         * @dev Destoys `amount` tokens from `account`.`amount` is then deducted
         * from the caller's allowance.
         *
         * See `_burn` and `_approve`.
         */
        // Beosin (Chengdu LianAn) // Redundant code, it is recommended to delete.
        function _burnFrom(address account, uint256 amount) internal {
                _burn(account, amount);
                _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
        }
}

// File: contracts\ERC20\Test.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
  * @title Test
  * @author Test
  *
  * @dev Standard ERC20 token with burning and optional functions implemented.
  * For full specification of ERC-20 standard see:
  * https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
  */
contract BonFi is ERC20 {

        string private _name; // Beosin (Chengdu LianAn) // Declare the variable '_name' for
storing the token name.
        string private _symbol; // Beosin (Chengdu LianAn) // Declare the variable '_symbol' for
storing the token symbol.
        uint8 private _decimals; // Beosin (Chengdu LianAn) // Declare the variable '_decimals' for
storing the token decimals.

        /**
          * @dev Constructor.
          * @param name name of the token
          * @param symbol symbol of the token, 3-4 chars is recommended
          * @param decimals number of decimal places of one token unit, 18 is widely used
          * @param totalSupply total supply of tokens in lowest units (depending on decimals)
          * @param tokenOwnerAddress address that gets 100% of token supply
          */
        constructor(string memory name, string memory symbol, uint8 decimals, uint256 totalSupply,
address tokenOwnerAddress) public payable {
            _name = name; // Beosin (Chengdu LianAn) // Initialize the token name.
            _symbol = symbol; // Beosin (Chengdu LianAn) // Initialize the token symbol.
            _decimals = decimals; // Beosin (Chengdu LianAn) // Initialize the token decimals.

            // set tokenOwnerAddress as owner of all tokens
            _mint(tokenOwnerAddress, totalSupply); // Beosin (Chengdu LianAn) // Call internal
function '_mint' to mint the specified amount of tokens to the specified address.
        }

        /**
          * @dev Burns a specific amount of tokens.
```

```
     * @param value The amount of lowest token units to be burned.
     */
    function burn(uint256 value) public {
        _burn(msg.sender, value); // Beosin (Chengdu LianAn) // Call the internal function
'_burn' to destroy tokens.
    }

    // optional functions from ERC20 stardard

    /**
     * @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;
    }
}
// Beosin (Chengdu LianAn) // Recommend the main contract to inherit 'Pausable' module to
grant owner the authority of pausing all transactions when serious issue occurred.
```

BEOSIN
Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com