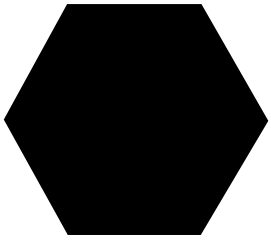
10 MAY 2021

Bifi
SMART CONTRACT AUDIT REPORT

Hexlant.



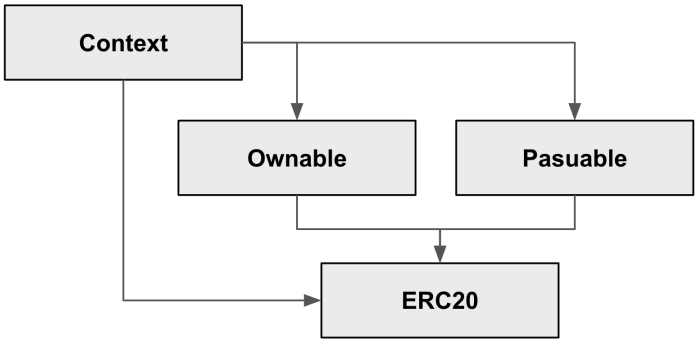
Analysis Purpose

This report analyzes and summarizes the results of published contract codes to determine whether they meet the requirements and identify security vulnerabilities and problems that may arise in practice. Hexlant Technical Team conducted this code analysis to verify the following factors:

- Proper operation of the implemented functions
- Security risks during the operation
- Preparation for the potential issues in off-chain transactions
- Readability and completeness of the contract codes

Function Summary

The Bifi contract is designed with the contract code provided by Open-Zeppelin and the contract code directly written and implemented. The Bifi function are built upon the following contracts:



- **Context**
  It provides information about execution context of current.

- **Ownable**
  It provides functions related to contract ownership. This function can limit the ability to execute functions to a specific address, the owner of the contract, by using the onlyOwner Modifier.

- **Pausable**
  It provides functions related to the pause of token transfer within the contract. Use the whenNotPaused and whenPaused modifiers to check the current contract's distribution restrictions when transferring tokens, and then make the transfer happen.

- **ERC20**
  It defines basic information of Bifi and implements detailed functions for ERC20.

Contract

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

Contract	Description
Context	Manage contract execution context information
Ownable	Manage contract owner authority
Pausable	Manage contract token pause status
ERC20	Main contract

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
Address	Functions related to address types

Variable

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

Variable	Description
_paused	Pause status of transferring contract tokens
_owner	Contract owner address
_balances	Hash table of token balances for a specific address
_allowances	Hash table of token balances delegated to a specific address
_totalSupply	Total token issuance
_name	Name of token
_symbol	Name of symbol
_decimals	The maximum number of representable decimal places for a token

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
whenNotPaused	Executable when contract token transfer is not paused
whenPaused	Executable when contract token transfer is paused
onlyOwner	Executable if it is the contract owner

Event

Log event according to contract function execution. It is used to more efficiently respond to the contract situation in future application application.

Event	Description
Paused	Event occurs when transitioning to contract token transfer state
UnPaused	Event occurs when releasing contract token transfer pause status
OwnershipTransferred	Event occurs when transferring the contract owner's authority
Transfer	Event occurs when transferring tokens
Approval	Event occurs when delegating token withdrawal

Function

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

Function	Description
_msgSender	Function caller (Sender)
_msgData	calldata
paused	Check the contract token transfer pause status
_pause	Pause the contract token transfer
_unpause	Release the Pause status of contract token transfer
owner	Check the contract owner address
renounceOwnership	Renounce the contract owner authority
transferOwnership	Transfer the contract owner authority
name	Check the name of token
symbol	Check the symbol of token
decimals	Check the decimals of token
totalSupply	Check the total token issuance
balanceOf	Check the token balance of a specific address
transfer	Transfer tokens to a specific address
allowance	Check the token balance delegated to a specific address
approve	Delegate withdrawal to a specific address
transferFrom	Transfer tokens that were delegated for withdrawal to a specific address
increaseAllowance	Increase the token balance delegated to a specific address
decreaseAllowance	Decrease the token balance delegated to a specific address
_transfer	Transfer tokens to a specific address
_burn	Burn tokens
_approve	Delegate withdrawal to a specific address
burn	Burn tokens
burnFrom	Burn tokens that are delegated for withdrawal
pause	Pause transferring contract tokens
unpause	Release the pause of transferring contract tokens
_beforeTokenTransfer	Check the pause status of transferring contract tokens before transferring tokens

Function Profile

Function Profile describes the details of the contract functions such as parameters, options and call relationships among functions via the call stacks. This function profile makes it possible to understand the function call relationship and quickly determine whether there is a logical conflict between functions.

Function Name	(Context) _msgSender		
Parameter	-	Return	address
Visibility	internal	Modifier	-
Constant	view	Inheritance	virtual
Callstack			
_msgSender			

Function Name	(Context) _msgData		
Parameter	-	Return	bytes
Visibility	internal	Modifier	-
Constant	view	Inheritance	virtual
Callstack			
_msgData			

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

Function Name	(Pausable) _pause		
Parameter	-	Return	-
Visibility	internal	Modifier	whenNotPaused
Constant	-	Inheritance	virtual
Callstack			
_pause			

Function Name	(Pausable) _unpause		
Parameter	-	Return	-
Visibility	internal	Modifier	whenPaused
Constant	-	Inheritance	virtual
Callstack			
_unpause			

Function Name	(Ownable) owner		
Parameter	-	Return	-
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
owner			

Function Name	(Ownable) renounceOwnership		
Parameter	-	Return	-
Visibility	public	Modifier	onlyOwner
Constant	-	Inheritance	virtual
Callstack			
renounceOwnership			

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

Function Name	(ERC20) name		
Parameter	-	Return	string
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
name			

Function Name	(ERC20) symbol		
Parameter	-	Return	string
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
symbol			

Function Name	(ERC20) decimals		
Parameter	-	Return	uint8
Visibility	public	Modifier	-
Constant	view	Inheritance	-
Callstack			
decimals			

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

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

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

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

Function Name	(ERC20) approve		
Parameter	address, uint256	Return	bool
Visibility	public	Modifier	-
Constant	-	Inheritance	virtual, override
Callstack			
approve └ _approve └ _msgSender			

Function Name	(ERC20) transferFrom		
Parameter	address, address, uint256	Return	bool
Visibility	public	Modifier	-
Constant	-	Inheritance	virtual, override
Callstack			
transferFrom └ _transfer └ _approve └ _msgSender			

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

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

decreaseAllowance
└ _approve
└ _msgSender

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

_transfer
└ _beforeTokenTransfer

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

_burn
└ _beforeTokenTransfer

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

_approve

Function Name	(ERC20) burn		
Parameter	uint256	Return	-
Visibility	public	Modifier	-
Constant	-	Inheritance	virtual
Callstack			

burn
└ _burn
└ _msgSender

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

burnFrom
└ allowance
└ _msgSender
└ _approve
└ _msgSender

Function Name	(ERC20) pause		
Parameter	-	Return	-
Visibility	public	Modifier	onlyOwner
Constant	-	Inheritance	-
Callstack			

pause
└ _pause

Function Name	(ERC20) unpause		
Parameter	-	Return	-
Visibility	public	Modifier	onlyOwner
Constant	-	Inheritance	-
Callstack			

unpause
└ _unpause

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

_beforeTokenTransfer
└ _paused

Test Result

Code Coverage

Code coverage is a quantitative index of how much the written test has tested the functionality of the contact code.

There are cases in which additional calls are not made to the library and functions implemented in some contracts in the Bifi contract.

The coverage index below is the result that reflects the details above.

File Name	Statements	Functions	Lines
ERC20.sol	100% (67/67)	100% (36/36)	100% (70/70)

Test cases

Test case	Result
The name of the token matches the given name.	PASS
The symbol of the token matches the given symbol.	PASS
The token decimals match the given decimals.	PASS
The given initial amount of issuance is allocated to the total issued amount.	PASS
The given initial amount of issuance is allocated to the contract owner, which is address executing distribution.	PASS
The token balance of addresses other than the owner is zero after distribution.	PASS
Is an exception handled if the receiving address is 0x0 when transferring tokens?	PASS
Is an exception handled if the amount to be sent is the negative number when transferring tokens?	PASS
Is an exception handled if exceeding the holding amount when transferring tokens?	PASS
When delegating withdrawal to a specific address, does the withdrawal delegation balance of that address increase?	PASS
Is it possible to increase or decrease the balance of delegated for withdrawal to a specific address?	PASS

Test case	Result
Does it become 0 if reducing the excess amount from the token balance delegated withdrawal to a specific address?	PASS
Is it possible to transfer tokens delegated for withdrawal?	PASS
Are the token balances of the relevant addresses updated correctly when transferring delegated tokens for withdrawal?	PASS
Is an exception handled if the receiving address is 0 when transferring delegated tokens for withdrawal?	PASS
Is an exception handled when transferring an amount that exceeds the delegated token balance for withdrawal?	PASS
Is an exception handled if the address delegating withdrawal's holding balance of its token is insufficient?	PASS
Is an exception handled when executing the token transfer pause function from addresses other than the contract pause owner(pauser)?	PASS
Is an exception handled when transferring tokens during the contract token transfer pause status?	PASS
Is an exception handled when transferring tokens through withdrawal delegation during the contract token transfer pause status?	PASS
Is it possible to release the contract token pause status?	PASS
Is it possible to check the contract owner's address?	PASS
Can the contract owner give up its authority? And does the owner's address become 0x0 after the renunciation?	PASS
Is an exception handled when transferring the owner's authority from addresses other than the contract owner's?	PASS
Can the contract owner transfer its authority?	PASS
Is an exception handled when burning tokens exceeding the holding balance?	PASS
Is an exception handled when burning tokens that exceed the balance more than delegated for withdrawal?	PASS
Are the holder's holding balance and total issuance reduced together when burning tokens?	PASS
Are the holding balance of the deputed person and total issuance reduced together when burning tokens delegated for withdrawal?	PASS
Does an event occur when transferring tokens?	PASS
Does an event occur when delegating withdrawals to a specific address?	PASS
Does an event occur when increasing or decreasing the balance of tokens delegated for withdrawals to a specific address?	PASS
Does an event occur when transferring the tokens delegated for withdrawals to a specific address?	PASS
Does an event occur when transferring the contract owner authority?	PASS
Does an event occur when burning tokens?	PASS

```
 *Submitted for verification at Etherscan.io on 2021-01-04
*/

pragma solidity ^0.6.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }

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

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

/**
 * @dev Contract module which allows children to implement an emergency stop
 * mechanism that can be triggered by an authorized account.
 *
 * This module is used through inheritance. It will make available the
 * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
 * the functions of your contract. Note that they will not be pausable by
 * simply including this module, only once the modifiers are put in place.
 */
contract Pausable is Context {
    /**
     * @dev Emitted when the pause is triggered by `account`.
     */
    event Paused(address account);

    /**
     * @dev Emitted when the pause is lifted by `account`.
     */
    event Unpaused(address account);

    bool private _paused;

    /**
     * @dev Initializes the contract in unpaused state.
     */
    constructor () internal {
        _paused = false;
    }
```

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

    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    modifier whenNotPaused() {
        require(!_paused, "Pausable: paused");
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    modifier whenPaused() {
        require(_paused, "Pausable: not paused");
        _;
    }

    /**
     * @dev Triggers stopped state.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    function _pause() internal virtual whenNotPaused {
        _paused = true;
        emit Paused(_msgSender());
    }

    /**
     * @dev Returns to normal state.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    function _unpause() internal virtual whenPaused {
        _paused = false;
        emit Unpaused(_msgSender());
    }
}
// SPDX-License-Identifier: MIT
/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
```

```
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
    address private _owner;

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
```

```
    function totalSupply() external view returns (uint256);
    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

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

        return c;
    }

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

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

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
```

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

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

        return c;
    }

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

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

        return c;
    }

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

```
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

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


pragma solidity ^0.6.0;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
```

```
https://medium.consensys.net/posts/2019/09/stop-using-solidity-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-
interactions-pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}



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

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;
    string private constant _name = "Bifi";
    string private constant _symbol = "Bifi";
    uint8 private constant _decimals = 18;

    /**
     * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
     * a default value of 18.
     *
     * To select a different value for {decimals}, use {_setupDecimals}.
     *
     * All three of these values are immutable: they can only be set once during
     * construction.
     */
```

```
contract BIFI is ERC20, Ownable {
        _totalSupply = (10 ** 9) * (10 ** uint256(_decimals));
        _balances[msg.sender] = _totalSupply;
    }

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

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
     * called.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }

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

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view override 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 virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }
```

```
    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20};
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override
returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
amount exceeds allowance"));
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
```

```
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20:
decreased allowance below zero"));
        return true;
    }

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

        _beforeTokenTransfer(sender, recipient, amount);

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

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

        _beforeTokenTransfer(account, address(0), amount);

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

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
     *
     * This is internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
```

```
     * emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

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

    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the caller's
     * allowance.
     *
     * See {ERC20-_burn} and {ERC20-allowance}.
     *
     * Requirements:
     *
     * - the caller must have allowance for ``accounts``'s tokens of at least
     * `amount`.
     */
    function burnFrom(address account, uint256 amount) public virtual {
        uint256 decreasedAllowance = allowance(account, _msgSender()).sub(amount, "ERC20: burn amount
exceeds allowance");

        _approve(account, _msgSender(), decreasedAllowance);
        _burn(account, amount);
    }

    /**
     * @dev Triggers stopped state.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */

    function pause() public onlyOwner {
        _pause();
    }

    /**
     * @dev Returns to normal state.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    function unpause() public onlyOwner {
        _unpause();
    }
```

```
    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:using-hooks.adoc[Using Hooks].
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual {
        require(!paused(), "ERC20Pausable: token transfer while paused");
    }
}
```

Vulnerability Analysis

Critical Severity

The critical-severity phase is a significant security flaw and causes fatal issues such as asset theft, freezing, and additional issuance. This defect must be corrected.

N/A

High Severity

The high-severity phase is an item that can cause security defects due to special conditions and is strongly recommended for correction. Additional exceptions or corner cases need to be analyzed and corrected to prevent errors.

N/A

Medium Severity

The medium-severity phase is not a security flaw but causes inefficient contract behavior. It is an item that is recommended modification to operate the contract efficiently.

N/A

Low Severity

The low-severity phase is an item with no security issues but is recommended for modifications to improve the contract structure and code readability.

N/A

Conclusion

The Bifi contract was written based on the ERC-20 interface, and the contract of Openzeppelin, a verified open-source, was mainly referenced. The main functions are the pause of all contract token transfers and token burn, but other functions such as lockup, additional issuance, and address freezing are not implemented.

The token transfer pause function can minimize damage in hacking and token theft situations and can be easily operated during token swap required for main network conversion. The contract owner has the main authority of this function, and the contract owner can take the form of decentralization through the function of giving up rights later.

Any security issues were found in this contract because it was written based on Openzeppelin's verified code and the contract's low complexity and conciseness.

Declare

The report is based on Hexlant's smart contract security audit results. This report does not warrant the suitability of the business model, legal regulation, or investment opinion. In addition to the problems described in the report, there may be undiscovered problems, including mainnet technology or virtual machines. This report is intended for discussion purposes only.

HEXLANT CONTRACT CERTIFICATION

This audit report and contract certification specify that the Hexlant Technical Team validated and notifies that it does not have any technical defects.

PUBLISHIED INFORMATION

REPORT NUMBER	ERC20200602
DATE	2021/05/10
PUBLISHIER	Henry henry@hexlant.com

TOKEN INFORMATION

TOKEN NAME	Bifi		
SYMBOL	Bifi		
PLATFORM	ETHEREUM	**TOKEN TYPE**	ERC-20
TOTAL SUPPLY	1,000,000,000 Bifi		
CONTRACT ADDRESS	0x2791BfD60D232150Bff86b39B7146c0eaAA2BA81		

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	No	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	No	Ability to block token transfers for a period of time (Administrators can use to set lockout periods for investors, team members, advisors, etc.)
BURN	YES	Ability to reduce total supply by burning tokens
MINT	NO	Ability to increase total supply by minting tokens

Certified by Hexlant.

Hexlant.

Blockchain Technology Research Institute

-

contact@hexlant.com

www.hexlant.com