Smart Contract
Security Audit Report

The SlowMist Security Team received the HillstoneFinance team's application for smart contract security audit of the HillstoneFinance on 2022.08.05. The following are the details and results of this smart contract security audit:

Token Name :

HillstoneFinance

The contract address :

https://etherscan.io/token/0xba6b0dbb2ba8daa8f5d6817946393aef8d3a4487

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed

NO.	Audit Items	Result
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002208090003

Audit Date : 2022.08.05 - 2022.08.09

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the tokenVault section and does not contain dark coin function. The total amount of contract tokens remains unchangeable. OpenZeppelin's SafeMath security module is used, which is a recommended approach. The contract does not have the Overflow and the Race Conditions issue.

During the audit we found the following information and issues:

1. The owner role can add or remove any user from the blacklist through the addBlackList or the removeBlackList functions.

The source code:

Ownable.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "../utils/Context.sol";
/**
 * @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.
 */
abstract 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 () {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual 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 {
        //SlowMist// This check is quite good in avoiding losing control of the
contract caused by user mistakes
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
```

ERC20.sol

```
// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.8.0;

import "./IERC20.sol";
import "./extensions/IERC20Metadata.sol";
import "../../utils/Context.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 {ERC20PresetMinterPauser}.
 *
 * 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, IERC20Metadata {
    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * The defaut value of {decimals} is 18. To select a different value for
     * {decimals} you should overload it.
     *
     * All two of these values are immutable: they can only be set once during
     * construction.
     */
    constructor (string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

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

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view virtual override 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 this function is
     * overridden;
     *
     * 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 virtual override returns (uint8) {
        return 18;
    }

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

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view virtual 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);
        //SlowMist// The return value conforms to the EIP20 specification
        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);
        //SlowMist// The return value conforms to the EIP20 specification
        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);

        uint256 currentAllowance = _allowances[sender][_msgSender()];
        require(currentAllowance >= amount, "ERC20: transfer amount exceeds
allowance");
        _approve(sender, _msgSender(), currentAllowance - amount);
        //SlowMist// The return value conforms to the EIP20 specification
        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] +
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
virtual returns (bool) {
        uint256 currentAllowance = _allowances[_msgSender()][spender];
        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance
below zero");
        _approve(_msgSender(), spender, currentAllowance - 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
virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to the loss of token during transfer
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        uint256 senderBalance = _balances[sender];
        require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
        _balances[sender] = senderBalance - amount;
        _balances[recipient] += 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 virtual {
    require(account != address(0), "ERC20: mint to the zero address");

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

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

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

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

    uint256 accountBalance = _balances[account];
    require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
    _balances[account] = accountBalance - amount;
    _totalSupply -= amount;

    emit Transfer(account, address(0), amount);
}

/**
 * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
 *
 * This 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");
        //SlowMist// This kind of check is very good, avoiding user mistake leading to approve errors
        require(spender != address(0), "ERC20: approve to the zero address");

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

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `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:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
}
```

IERC20.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/**
 * @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);
}
```

IERC20Metadata.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;
```

```
import "../IERC20.sol";

/**
 * @dev Interface for the optional metadata functions from the ERC20 standard.
 *
 * _Available since v4.1._
 */
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);

    /**
     * @dev Returns the symbol of the token.
     */
    function symbol() external view returns (string memory);

    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view returns (uint8);
}
```

Context.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/*
 * @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 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.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

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

SafeMath.sol

```
// SPDX-License-Identifier: MIT
//SlowMist// OpenZeppelin's SafeMath security module is used, which is a recommended
approach
pragma solidity ^0.8.0;

// CAUTION
// This version of SafeMath should only be used with Solidity 0.8 or later,
// because it relies on the compiler's built in overflow checks.

/**
 * @dev Wrappers over Solidity's arithmetic operations.
 *
 * NOTE: `SafeMath` is no longer needed starting with Solidity 0.8. The compiler
 * now has built in overflow checking.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
            return (true, c);
        }
    }
    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b > a) return (false, 0);
            return (true, a - b);
        }
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow
flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            // 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 (true, 0);
            uint256 c = a * b;
            if (c / a != b) return (false, 0);
            return (true, c);
        }
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero
flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a / b);
        }
    }
    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a % b);
        }
    }

    /**
     * @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) {
        return a + b;
    }

    /**
     * @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 a - b;
    }

    /**
     * @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) {
        return a * b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator.
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * reverting 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 a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom
```

```
message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for
the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        unchecked {
            require(b <= a, errorMessage);
            return a - b;
        }
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with
custom message on
     * division by zero. The result is rounded towards 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).
     *
     * 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) {
        unchecked {
            require(b > 0, errorMessage);
            return a / b;
        }
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for
the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * 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) {
        unchecked {
            require(b > 0, errorMessage);
            return a % b;
        }
    }
}
```

HillstoneFinance.sol

```
// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity 0.8.6;


import "@openzeppelin/contracts/utils/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

abstract contract BlackList is Ownable, ERC20 {

    mapping (address => bool) public isBlackListed;
    function getBlackListStatus(address _addr) external view returns (bool) {
        return isBlackListed[_addr];
    }

    function addBlackList (address _addr) public onlyOwner {
        isBlackListed[_addr] = true;
        emit AddedBlackList(_addr);
    }

    function removeBlackList (address _addr) public onlyOwner {
        isBlackListed[_addr] = false;
        emit RemovedBlackList(_addr);
    }

    event AddedBlackList(address _user);

    event RemovedBlackList(address _user);

}

contract HillstoneFinance is ERC20, BlackList {

    using SafeMath for uint256;
    uint256 constant private _initial_supply = 10**26;

    constructor() ERC20("Hillstone.Finance", "HSF") {
        _mint(msg.sender, _initial_supply);
    }

    function transfer(address _to, uint _value) public override returns (bool
success) {
        require(!isBlackListed[msg.sender], "HSF/transfer: Should not transfer from
blacklisted address");
        return super.transfer(_to, _value);
    }

    function transferFrom(address _from, address _to, uint _value) public override
returns (bool success) {
        require(!isBlackListed[_from], "HSF/transferFrom: Should not transfer from
blacklisted address");
        return super.transferFrom(_from, _to, _value);
    }
}
```

Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

SLOWMIST

Official Website

www.slowmist.com

✉

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist