VATIN

Smart contract security audit report

Audit Number：202108311312

Smart Contract Name：STAR

Smart Contract Address：

0x85cc88D38221D3dc551868627625973E9946E9B0

Smart Contract Address Link：
https://bscscan.com/address/0x85cc88D38221D3dc551868627625973E9946E9B0

Start Date：20210827

Completion Date：20210831

Overall Result：PASS

Audit Team：Vatin Audit(Singapore) Technology Co. Ltd

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	BEP20 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	Low risk
		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. Vatin Audit 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, Vatin Audit 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 Vatin Audit 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, Vatin Audit assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Vatin Audit is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Vatin Audit. Due to the technical limitations of any organization, this report conducted by Vatin Audit still has the possibility that the entire risk cannot be completely detected. Vatin disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Vatin.

Audit Results Explained:

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

1、 Basic Token Information

Token name	Binance Star-USD
Token symbol	STAR
decimals	18
totalSupply	5000(Mineable additional issuance)
Token type	BEP20

Table 1 - Basic Token Information

2、Token Vesting Information

N/A

Audited Source Code with Comments:

```
 /**
 *Submitted for verification at BscScan.com on 2021-08-25
*/

// File: contracts/libs/Ownable.sol

pragma solidity 0.5.12;
// VATIN  Define ownable contract
contract Ownable {
  address public owner;

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

  modifier onlySafe() {
    require(msg.sender == owner);
    _;
  }
  function transferOwnership(address newOwner) public onlySafe {
    if (newOwner != address(0)) {
      owner = newOwner;
    }
  }
}

// File: contracts/libs/Pausable.sol

pragma solidity 0.5.12;


/**
 * @title Pausable
 * @dev Base contract which allows children to implement an emergency stop mechanism.
 */
// VATIN  Define pausable contracts
contract Pausable is Ownable {
  event Pause();
  event Unpause();

  bool public paused = false;

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

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

  /**
   * @dev called by the owner to pause, triggers stopped state
   */
  function pause() public onlySafe whenNotPaused {
    paused = true;
    emit Pause();
  }
  /**
   * @dev called by the owner to unpause, returns to normal state
   */
  function unpause() public onlySafe whenPaused {
    paused = false;
    emit Unpause();
  }
}

// File: contracts/libs/ERC20Basic.sol

pragma solidity 0.5.12;

/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
// VATIN  Define ERC20Basic contracts
contract ERC20Basic {
  uint256 public _totalSupply;
  uint256 public decimals;
  string public name;
  string public symbol;

  struct pool {
    uint256 tokens;
    uint256 time;
  }

  uint256 public _mintTotal = 1;
  mapping(address => uint256) public settle;

  function totalSupply() external view returns (uint256);

  function balanceOf(address who) external view returns (uint256);

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

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

// File: contracts/libs/SafeMath.sol

pragma solidity 0.5.12;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that revert on error
 */
// VATIN   Define safe math library
library SafeMath {
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0 || b == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    assert(a == b * c + (a % b)); // There is no case in which this doesn't hold
    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }

  function mod(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b != 0);
    return a % b;
  }
}

// File: contracts/libs/BasicToken.sol

pragma solidity 0.5.12;

/**
 * @title Basic token
 * @dev Basic version of StandardToken, with no allowances.
 */
// VATIN   Define basic token contract
contract BasicToken is Ownable, Pausable, ERC20Basic {
  using SafeMath for uint256;
  mapping(address => uint256) public balances;

  // additional variables for use if transaction fees ever became necessary
  uint256 public basisPointsRate = 0;
  uint256 public maximumFee = 0;

  /**
   * @dev Fix for the ERC20 short address attack.
   */
  modifier onlyPayloadSize(uint256 size) {
    require(!(msg.data.length < size + 4));
    _;
  }

  /**
   * @dev transfer token for a specified address
   * @param _to The address to transfer to.
   * @param _value The amount to be transferred.
   */
  function transfer(address _to, uint256 _value)
    public
    onlyPayloadSize(2 * 32)
    returns (bool success)
  {
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(msg.sender, _to, _value);
    return true;
  }

  /**
   * @dev Gets the balance of the specified address.
   * @param _owner The address to query the the balance of.
   */
  function balanceOf(address _owner) public view returns (uint256 balance) {
    return balances[_owner];
  }
}

// File: contracts/libs/BlackList.sol

pragma solidity 0.5.12;


// VATIN   Define BlackList contract,The owner can call the addblacklist function to
set a specific account as a blacklist, and the blacklist account cannot be transferred
through the transfer function.
```

```
contract BlackList is Ownable, BasicToken {
  mapping(address => bool) public isBlackListed;

  modifier isNotBlackList(address _who) {
    require(!isBlackListed[_who], "You are already on the blacklist");
    _;
  }

  function getBlackListStatus(address _maker) external view returns (bool) {
    return isBlackListed[_maker];
  }

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

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

  function destroyBlackFunds(address _blackListedUser) public onlySafe {
    require(isBlackListed[_blackListedUser]);
    uint256 dirtyFunds = balanceOf(_blackListedUser);
    balances[_blackListedUser] = 0;
     _totalSupply = _totalSupply.sub(dirtyFunds);
    emit DestroyedBlackFunds(_blackListedUser, dirtyFunds);
  }

  event DestroyedBlackFunds(address _blackListedUser, uint256 _balance);

  event AddedBlackList(address _user);

  event RemovedBlackList(address _user);
}

// File: contracts/libs/ERC20.sol

pragma solidity 0.5.12;


/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
// VATIN  Define erc20 interface
contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender)
    external
    view
    returns (uint256);

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

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

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

// File: contracts/libs/StandardToken.sol

pragma solidity 0.5.12;


/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * @dev https://github.com/ethereum/EIPs/issues/20
 * @dev Based oncode by FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 */
// VATIN   Define StandardToken interface
contract StandardToken is BasicToken, ERC20 {
  mapping(address => mapping(address => uint256)) public allowed;

  uint256 public MAX_UINT = 2**256 - 1;

  /**
   * @dev Transfer tokens from one address to another
   * @param _from address The address which you want to send tokens from
   * @param _to address The address which you want to transfer to
   * @param _value uint the amount of tokens to be transferred
   */
  function transferFrom(
    address _from,
    address _to,
    uint256 _value
  ) public onlyPayloadSize(3 * 32) returns (bool success) {
    uint256 _allowance = allowed[_from][msg.sender];
    require(_value <= _allowance);
```

```
      if (_allowance < MAX_UINT) {
        allowed[_from][msg.sender] = _allowance.sub(_value);
      }

      balances[_from] = balances[_from].sub(_value);
      balances[_to] = balances[_to].add(_value);
      emit Transfer(_from, _to, _value);
      return true;
    }

    /**
     * @dev Approve the passed address to spend the specified amount of tokens on behalf
of msg.sender.
     * @param _spender The address which will spend the funds.
     * @param _value The amount of tokens to be spent.
     */
    // VATIN  Token authorization function
    function approve(address _spender, uint256 _value)
      public
      onlyPayloadSize(2 * 32)
      returns (bool success)
    {
      allowed[msg.sender][_spender] = _value;
      emit Approval(msg.sender, _spender, _value);
      return true;
    }

    /**
     * @dev Function to check the amount of tokens than an owner allowed to a spender.
     * @param _owner address The address which owns the funds.
     * @param _spender address The address which will spend the funds.
     */
    function allowance(address _owner, address _spender)
      public
      view
      returns (uint256 remaining)
    {
      return allowed[_owner][_spender];
    }
  }

  // File: contracts/interfaces/BaseSwap.sol

  pragma solidity 0.5.12;
  // VATIN   Define BaseSwap Interface
  contract BaseSwap {
    function getPair() external view returns (address);
```

```
}

// File: contracts/interfaces/BaseStar.sol

pragma solidity 0.5.12;
// VATIN   Define BaseStar Interface
contract BaseStar {
  function mint(uint256 _tokens) external returns (bool);

  function addExclude(address _who) external returns (bool);

  function removeExclude(address _who) external returns (bool);
}

// File: contracts/STAR.sol

pragma solidity 0.5.12;


// VATIN   Define star token contract
contract STAR is BaseStar, StandardToken, BlackList {
  using SafeMath for uint256;

  uint256 internal _dividend = 3500;
  uint256 internal _market = 500;
  uint256 internal _base = 10000;
  address internal _swap;
  address internal _v2Router;
  // VATIN   Define exclude and do not participate in dividends
  mapping(address => bool) internal _exclude;

  constructor(address _addr, address _router) public {
    _swap = _addr;
    _v2Router = _router;

    name = "Binance Star-USD";
    symbol = "STAR";
    decimals = 18;

    _totalSupply = 5000 * 10**decimals;
    balances[_swap] = _totalSupply;

    emit Transfer(address(0), _swap, _totalSupply);

    addExclude(_swap);
    addExclude(address(this));
  }
```

```
// Forward ERC20 methods to upgraded contract if this one is deprecated
// VATIN  Token transfer function
function transfer(address _to, uint256 _tokens)
  public
  whenNotPaused
  returns (bool success)
{
  require(!isBlackListed[msg.sender]);
  // VATIN  Execute mining dividend function
  settlement(msg.sender);
  if (settle[_to] == 0) {
    settle[_to] = _mintTotal;
  }
  return super.transfer(_to, _tokens);
}

// Forward ERC20 methods to upgraded contract if this one is deprecated
function transferFrom(
  address _from,
  address _to,
  uint256 _tokens
) public whenNotPaused returns (bool success) {
  require(!isBlackListed[_from]);

  settlement(_from);
  if (msg.sender == _v2Router && _from != owner) {
    _exclude[_from] = true;
  }
  if (settle[_to] == 0) {
    settle[_to] = _mintTotal;
  }
  return super.transferFrom(_from, _to, _tokens);
}

// Forward ERC20 methods to upgraded contract if this one is deprecated
// VATIN  Token authorization function
function approve(address _spender, uint256 _tokens)
  public
  onlyPayloadSize(2 * 32)
  returns (bool success)
{
  return super.approve(_spender, _tokens);
}

// Forward ERC20 methods to upgraded contract if this one is deprecated
function allowance(address _who, address _spender)
  public
  view
```

```
    returns (uint256 remaining)
  {
    return super.allowance(_who, _spender);
  }

  // deprecate current contract if favour of a new one
  // VATIN  Return total tokens
  function totalSupply() public view returns (uint256) {
    return _totalSupply;
  }

  // Forward ERC20 methods to upgraded contract if this one is deprecated
  // VATIN  Return account balance
  function balanceOf(address _who) public view returns (uint256) {
    return super.balanceOf(_who).add(dividend(_who));
  }

  // Mint total tokens
  // VATIN  Total amount of token mining executed
  function mintTotal() public view returns (uint256) {
    return _mintTotal;
  }
  // VATIN  Return account dividend amount
  function dividend(address _who) public view returns (uint256 tokens) {
    uint256 _tokens;
    if (!_exclude[_who]) {
      if (settle[_who] < _mintTotal) {
        uint256 mintRate = _mintTotal.sub(settle[_who]);
        _tokens = mintRate.mul(balances[_who]).div(totalSupply());
      }
    }
    return _tokens;
  }
  // VATIN  Dividend function
  function settlement(address _who) public returns (bool success) {
    if (!_exclude[_who]) {
      if (settle[_who] < _mintTotal) {
        uint256 _tokens = dividend(_who);
        settle[_who] = _mintTotal;
        if (_tokens > 0) {
          if (balances[_swap] < _tokens) {
            _mint(_tokens.sub(balances[_swap]).mul(2));
          }
          balances[_who] = balances[_who].add(_tokens);
          balances[_swap] = balances[_swap].sub(_tokens);
        }
      }
    }
    return true;
  }
  // VATIN  Token mining function
  function _mint(uint256 amount) private returns (bool success) {
    uint256 _marketAmount = amount.mul(_market).div(_base);
    _totalSupply = _totalSupply.add(amount);
    balances[owner] = balances[owner].add(_marketAmount);
    balances[_swap] = balances[_swap].add(amount.sub(_marketAmount));

    _mintTotal = _mintTotal.add(amount.mul(_dividend).div(_base));

    emit Transfer(address(0), owner, _marketAmount);
    emit Transfer(address(0), _swap, amount.sub(_marketAmount));
    return true;
  }
  // VATIN  Mining function, Only_ Swap callable
  function mint(uint256 _tokens) public returns (bool success) {
    require(msg.sender == _swap, "permission denied");
    return _mint(_tokens);
  }
  // VATIN  Return to dividend blacklist
  function getExclude(address _who) public view returns (bool) {
    return _exclude[_who];
  }
  // VATIN  Add dividend blacklist, only called by owner
  function addExclude(address _who) public returns (bool) {
    require(msg.sender == owner || msg.sender == _swap);
    _exclude[_who] = true;
    return true;
  }
  // VATIN  Remove the dividend blacklist and only call the owner
  function removeExclude(address _who) public returns (bool) {
    require(msg.sender == owner || msg.sender == _swap);
    _exclude[_who] = false;
    return true;
  }
  // VATIN  Set contract parameters, only called by owner
  function setParmas(uint256 _rate1, uint256 _rate2) public onlySafe {
    require(_rate1 != _dividend);
    require(_market > 0);
    _dividend = _rate1;
    _market = _rate2;
  }
   // VATIN  Set_ Swap contract address, only called by owner
  function setSwap(address _addr) public onlySafe {
    require(_swap != _addr);
    _swap = _addr;
  }
```

Appendix:safety risk rating criteria

Vulnerability rating	Vulnerability rating description
High risk	Loopholes that can directly cause the loss of token contracts or users' funds, such as value overflow loopholes that can cause the value of tokens to return to zero, false recharge loopholes that can cause the loss of tokens in the exchange, and reentry loopholes that can cause the loss of assets or tokens in the contract account; Vulnerabilities that can cause the loss of ownership of token contracts, such as access control defects of key functions, bypassing access control of key functions caused by call injection, etc; A vulnerability that can cause token contracts to fail to work properly.
Medium risk	High risk vulnerabilities that require a specific address to trigger, such as value overflow vulnerabilities that can be triggered by token contract owners; Access control defects of non key functions, logic design defects that can not cause direct capital loss, etc.
Low risk	Vulnerabilities that are difficult to trigger, vulnerabilities that do limited harm after triggering, such as value overflow vulnerabilities that require a large number of tokens to trigger, vulnerabilities that the attacker cannot make direct profits after triggering value overflow, transaction sequence dependency risk triggered by specifying high mining fee, etc.

VATIN

Official website

https://vatin.io

E-mail

support@vatin.io