BlockSAFU

ADVANCE MANUAL
SMART CONTRACT AUDIT

Project: Token Lock Endless Burn

Website: endlessburn.com

PASSED

BlockSAFU Score: 82

Contract Address:

0x9EE064EDdF07e7B41fF6432dcB2E97DDc9d0207B

Disclamer: BlockSAFU is not responsible for any financial losses.
Nothing in this contract audit is financial advice, please do your own reasearch.

DISCLAMER

BlockSAFU has completed this report to provide a summary of the Smart Contract functions, and any security, dependency, or cybersecurity vulnerabilities. This is often a constrained report on our discoveries based on our investigation and understanding of the current programming versions as of this report's date. To understand the full scope of our analysis, it is vital for you to at the date of this report. To understand the full scope of our analysis, you need to review the complete report. Although we have done our best in conducting our investigation and creating this report, it is vital to note that you should not depend on this report and cannot make any claim against BlockSAFU or its Subsidiaries and Team members on the premise of what has or has not been included in the report. Please remember to conduct your independent examinations before making any investment choices. We do not provide investment advice or in any way claim to determine if the project will be successful or not.

By perusing this report or any portion of it, you concur to the terms of this disclaimer. In the unlikely situation where you do not concur to the terms, you should immediately terminate reading this report, and erase and discard any duplicates of this report downloaded and/or printed by you. This report is given for data purposes as it were and on a non-reliance premise and does not constitute speculation counsel. No one should have any right to depend on the report or its substance, and BlockSAFU and its members (including holding companies, shareholders, backups, representatives, chiefs, officers, and other agents) BlockSAFU and its subsidiaries owe no obligation of care towards you or any other person, nor does BlockSAFU make any guarantee or representation to any individual on the precision or completeness of the report.

ABOUT THE AUDITOR:

BlockSAFU (BSAFU) is an Anti-Scam Token Utility that reviews Smart Contracts and Token information to Identify Rug Pull and Honey Pot scamming activity. BlockSAFUs Development Team consists of several Smart Contract creators, Auditors Developers, and Blockchain experts. BlockSAFU provides solutions, prevents, and hunts down scammers. BSAFU is a utility token with features Audit, KYC, Token Generators, and Bounty Scammers. It will enrich the crypto ecosystem.

OVERVIEW

BlockSAFU was commissioned by Endless Burn to complete a Smart Contract audit. The objective of the Audit is to achieve the following:

- Review the Project and experience and Development team
- Ensure that the Smart Contract functions are necessary and operate as intended.
- Identify any vulnerabilities in the Smart Contract code.

DISCLAIMER: This Audit is intended to inform about token Contract Risks, the result does not imply an endorsement or provide financial advice in any way, all investments are made at your own risk. (https://blocksafu.com/)

SMART CONTRACT REVIEW

Token Name	TokenLock
Contract Address	0x9EE064EDdF07e7B41fF6432dcB2E97DDc9d0207B
Deployer Address	0x9e6145008c3AeAdb8025EF3866220D0eb6623434
Owner Address	0xad25B4d86E6d274dd7A9b48cB4F031e1b0D15f2A
Contract Created	Jun-22-2022 01:52:19 AM +UTC
Unlocked Date	*will be updated after the DEX listing*
Verified CA	Yes
Compiler	v0.7.4+commit.3f05b770
Optimization	Enable with 200 runs
Sol License	MIT License
Top 5 Holders	*will be updated after the DEX listing*
Other	default evmVersion

MANUAL CODE REVIEW

🟢 Minor-risk

0 minor-risk code issues found

Could be fixed, and will not bring problems.

🟡 Medium-risk

0 medium-risk code issues found

Should be fixed, could bring problems.

🔴 High-Risk

0 high-risk code issues found

Must be fixed, and will bring problem.

🔴 Critical-Risk

0 critical-risk code issues found

Must be fixed, and will bring problem.

EXTRA NOTES SMART CONTRACT

1. IERC20

```
library SafeMathInt {
    int256 private constant MIN_INT256 = int256(1) << 255;
    int256 private constant MAX_INT256 = ~(int256(1) << 255);
    function mul(int256 a, int256 b) internal pure returns (int256) {
        int256 c = a * b;
        require(c != MIN_INT256 || (a & MIN_INT256) != (b & MIN_INT256));
        require((b == 0) || (c / b == a), 'mul overflow');
        return c;
    }
    function div(int256 a, int256 b) internal pure returns (int256) {
        require(b != -1 || a != MIN_INT256);
        return a / b;
    }
    function sub(int256 a, int256 b) internal pure returns (int256) {
        int256 c = a - b;
        require((b >= 0 && c <= a) || (b < 0 && c > a),
            'sub overflow');
        return c;
    }
    function add(int256 a, int256 b) internal pure returns (int256) {
        int256 c = a + b;
        require((b >= 0 && c >= a) || (b < 0 && c < a),
            'add overflow');
        return c;
    }
    function abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256,
            'abs overflow');
        return a < 0 ? -a : a;
    }
    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }
    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a < b ? a : b;
    }
}
```

SafeMathInt Normal Base Template

2. SafeMath Contract

```
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);
        }
    }

  ...
    function mod(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b > 0, errorMessage);
            return a % b;
        }
    }
}
```

Standard Safemath contract

3. Contract TokenLock

```
contract TokenLock is Ownable {

    using SafeMath for uint256;
    using SafeMathInt for int256;

    address public constant lockToken =
0x1FEA9245376F256228cBEA767a2e0fD48FbD0fEc;
    uint256 public startingTime;
    uint256 public lockAmount;
    uint256 public withdrawnAmount;

    constructor () {
        startingTime = block.timestamp;
        address _owner =
0xad25B4d86E6d274dd7A9b48cB4F031e1b0D15f2A;
        transferOwnership(_owner);
    }

    function lock (uint256 _amount) public onlyOwner {
        IBEP20 _token = IBEP20 (lockToken);
        bool success = _token.transferFrom(msg.sender,
address(this), _amount);
        require(success, "Token transfer failed");
        lockAmount = lockAmount.add(_amount);
    }

    function unlock () public onlyOwner {
        uint256 _time = ((block.timestamp).sub(startingTime)).div(30
days);
        uint256 _amount = lockAmount.mul(_time).div(16);
        uint256 _amountEligible = _amount.sub(withdrawnAmount);

        IBEP20 _token = IBEP20 (lockToken);
        require (_amountEligible>0 , "No unlocked amount withdraw
currently");
        bool success = _token.transfer(msg.sender,
_amountEligible);
        require (success, "Transfer failed");

        withdrawnAmount = _amount;

    }

    // this function is to withdraw extra tokens locked in the
contract.
```

```
    function withdrawLockedTokens (address _tokenAddress)
external onlyOwner returns (bool) {
        require (_tokenAddress != lockToken, "You can't withdraw
locked token. Please use unlock function");
        IBEP20 token = IBEP20 (_tokenAddress);
        uint256 balance = token.balanceOf(address(this));

        bool success = token.transfer(msg.sender, balance);
        return success;
    }


    // this function is to withdraw BNB sent to this address by
mistake
    function withdrawEth () external onlyOwner returns (bool) {
        uint256 balance = address(this).balance;
        (bool success, ) = payable(msg.sender).call{
            value: balance
        }("");
        return success;
    }
}
```

TokenLock Contract

4. Function for Locking Token

```
address public constant lockToken =
0x1FEA9245376F256228cBEA767a2e0fD48FbD0fEc;
function lock (uint256 _amount) public onlyOwner {
        IBEP20 _token = IBEP20 (lockToken);
        bool success = _token.transferFrom(msg.sender, address(this), _amount);
        require(success, "Token transfer failed");
        lockAmount = lockAmount.add(_amount);
    }
```

Function for lock token Endless Burn (LESS)
0x1FEA9245376F256228cBEA767a2e0fD48FbD0fEc
Other than the token with the contract address, it is unlocked.

5. Function Vesting

```
function unlock () public onlyOwner {
        uint256 _time = ((block.timestamp).sub(startingTime)).div(30 days);
        uint256 _amount = lockAmount.mul(_time).div(16);
        uint256 _amountEligible = _amount.sub(withdrawnAmount);
        IBEP20 _token = IBEP20 (lockToken);
        require (_amountEligible>0 , "No unlocked amount withdraw currently");
        bool success = _token.transfer(msg.sender, _amountEligible);
        require (success, "Transfer failed");
        withdrawnAmount = _amount;
    }
```

Function For unlocking token with vesting 16 cycles, and lockup time every 30 days.

6. Function Withdrawal Token

```
// this function is to withdraw extra tokens locked in the contract.
    function withdrawLockedTokens (address _tokenAddress) external
onlyOwner returns (bool) {
        require (_tokenAddress != lockToken, "You can't withdraw locked token.
Please use unlock function");
        IBEP20 token = IBEP20 (_tokenAddress);
        uint256 balance = token.balanceOf(address(this));
        bool success = token.transfer(msg.sender, balance);
        return success;
    }
```

Only the owner has access to withdraw tokens. This function allows all token contract addresses to be retrieved except the locked token. Not limited to reflection tokens only.

6. Function Withdrawal BNB

```
// this function is to withdraw BNB sent to this address by mistake
   function withdrawEth () external onlyOwner returns (bool) {
      uint256 balance = address(this).balance;
      (bool success, ) = payable(msg.sender).call{
         value: balance
      }("");
      return success;
   }
```

Only the owner has access to withdraw BNB

READ CONTRACT

1. lockAmount
   0 *unint256*
   (Shows the number of tokens that are locked)

2. lockToken
   0x1fea9245376f256228cbea767a2e0fd48fbd0fec *address*
   (Shows the smart contract token which is locked)

3. owner
   0xad25b4d86e6d274dd7a9b48cb4f031e1b0d15f2a *address*
   (Shows the address of the token owner)

4. startingTime
   1655862739 *uint256*
   (Token locking starts on: Wed Jun 22 2022 01:52:19 GMT+0000)

5. withdrawnAmount
   0 *uint256*
   (Shows the number of tokens that have been unlocked and can be withdrawn.)

WRITE CONTRACT

1. lock
   _amount (uint256)
   (The form is filled with the number of tokens to be locked)

2. renounceOwnership
   (Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner)

3. transferOwnership
   newOwner (address)
   (Its function is to change the owner)

4. unlock
   (Withdraw all LESS tokens amount to the owner's wallet)

5. withdrawEth
   (Withdraw all BNB tokens amount to the owner's wallet)

6. withdrawLockedTokens
   (Withdraw all token amounts by triggering the contract address to the wallet owner.)

CONCLUSION

Based on the explanation above, it can be concluded:

- Only the owner has access to all contract functions.
- Only contracts with address 0x1FEA9245376F256228cBEA767a2e0fD48FbD0fEc can be locked.
- Token locking starts on: Wed Jun 22 2022 01:52:19 GMT+0000.
- Unlocking token with vesting 16 cycles.
- Lockup time every 30 days.
- The owner can withdraw BNB.
- The owner can withdraw unlock LESS token.
- The owner can withdraw the BEP20 token.
- The contract owner can change.