Key To Profit

Smart Contract Audit
Final Report

June 26, 2023

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Introduction

1. About KeyToProfit

The #1 payment system in the world. We are glad to present you a universal cryptocurrency payment system - Key to Profit. KTP opens up new opportunities for businesses and investors in the digital age. Visit https://keytoprofit.org/ to know more about.

2. About ContractSecurity

ContractSecurity is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ContractSecurity's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 105+ blockchain projects by providing security services on different frameworks. ContractSecurity team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit https://contractsecurity.finance to know more about the services.

Documentation Details

The KeyToProfit team has provided the following doc for the purpose of audit:

1. https://keytoprofit.org/about

Audit Process & Methodology

ContractSecurity team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Contract: 0x2559021Ea94Cb061761aA6c6b15D97A86fBA646E
- Compiler Version: v0.8.19+commit.7dd6d404

- Blockchain: Binance Smart Chain
- Contract Name: KeyToProfit
- Token Name: KeyToProfit
- Symbol: KTP
- Decimals: 18
- Total Supply: 21,000,000 KTP
- Owner Address: 0xA98FBc8D22C156B4488D0EAA516F51E910307d2A

- Deployment Date: Jun-25-2023 11:06:46 PM +UTC, Block 29422769

Conclusion:

The KeyToProfit Smart-Contract found no vulnerabilities, no backdoors, and no scam scripts.
The code was tested with compatible compilers and simulated manually reviewed for all commonly known and specific vulnerabilities.

So, KeyToProfit Smart-Contract is safe for use in the Binance Smart Chain main network.

Security Level References

Every issue in this report was assigned a severity level from the following:

HIGH ISSUES will bring problems and should be fixed.

ERRORS, WARNINGS and OPTIMIZATION could potentially bring problems and should eventually be fixed.

NOTICE are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

		CRITICAL ISSUES
HIGH ISSUES	(High, medium severity)	0
ERRORS, WARNINGS	(Medium, low severity)	0
OPTIMIZATION	(Low severity)	0
NOTICE	(Very low severity)	0

HIGH ISSUES

No issues were found.

ERRORS, WARNINGS

No issues were found.

OPTIMIZATION

No issues were found.

NOTICE

No issues were found.

NOTICE:

The owner can control such functions:

SetTaxSell

Sets the commission parameter for the sale, as well as the address of the commission recipient. The commission cannot be more than 30% (from 0 to 30)

SetBurnFeePercent

sets the burning percentage parameter (from 0 to 2)

AddAddressLiquidity

sets liquidity address

RemoveAddressLiquidity

removes liquidity address

Source code:

```
/**
*Submitted for verification at BscScan.com on 2023-06-25
*/

// SPDX-License-Identifier: MIT

pragma solidity >=0.8.19;

interface IERC20 {
                    function totalSupply() external view returns (uint256);

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

    function allowance(address owner, address spender) external view returns (uint256);

    function transfer(address recipient, uint256 amount) external returns (bool);

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

    function transferFrom(address sender, address recipient, uint256 amount) external
returns (bool);

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
library SafeMath {
    // Counterpart to Solidity's `+` operator.
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    // Counterpart to Solidity's `-` operator.
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    // Counterpart to Solidity's `-` operator.
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;
        return c;
    }
    // Counterpart to Solidity's `*` operator.
    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;
    }

    // Counterpart to Solidity's `/` operator.
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    // Counterpart to Solidity's `/` operator.
    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;
    }

    // Counterpart to Solidity's `%` operator.
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    // Counterpart to Solidity's `%` operator.
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () { }

    function _msgSender() internal view returns (address payable) {
        return payable(msg.sender);
    }
function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://
github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}
contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
    constructor () {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    function owner() public view returns (address) {
        return _owner;
    }

    modifier onlyOwner() {
        require(_owner == _msgSender(), "onlyOwner");
        _;
    }

    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

contract SwapBurn is Ownable {
    using SafeMath for uint256;

    mapping(address=>bool) addressesLiquidity;

    uint256[] private percentsTaxSell;
    uint256 public _burnFee;
    address[] private addressesTaxSell;

    function getTaxSum(uint256[] memory _percentsTax) internal pure returns (uint256)
{
        uint256 TaxSum = 0;
        for (uint i; i < _percentsTax.length; i++) {
            TaxSum = TaxSum.add(_percentsTax[i]);
        }
        return TaxSum;
    }
    function getPercentsTaxSell() public view returns (uint256[] memory) {
        return percentsTaxSell;
    }

    function getAddressesTaxSell() public view returns (address[] memory) {
        return addressesTaxSell;
    }
```

```
function checkAddressLiquidity(address _addressLiquidity) external view returns (bool) {
        return addressesLiquidity[_addressLiquidity];
    }

    function addAddressLiquidity(address _addressLiquidity) public onlyOwner {
        addressesLiquidity[_addressLiquidity] = true;
    }

    function removeAddressLiquidity (address _addressLiquidity) public onlyOwner {
        addressesLiquidity[_addressLiquidity] = false;
    }

    function setTaxSell(uint256[] memory _percentsTaxSell, address[] memory
_addressesTaxSell) public onlyOwner {
        require(_percentsTaxSell.length == _addressesTaxSell.length,
"_percentsTaxSell.length != _addressesTaxSell.length");

        uint256 TaxSum = getTaxSum(_percentsTaxSell);
        require(TaxSum <= 30, "TaxSum > 30"); // Set the maximum tax limit

        percentsTaxSell = _percentsTaxSell;
        addressesTaxSell = _addressesTaxSell;
    }

    function showTaxSell() public view returns (uint[] memory, address[] memory) {
        return (percentsTaxSell, addressesTaxSell);
    }

    function showTaxSellSum() public view returns (uint) {
        return getTaxSum(percentsTaxSell);
    }

    function setBurnFeePercent(uint256 burnFee) external onlyOwner {
        require(burnFee <= 2, "Burn fee cannot be more than 2%");
        _burnFee = burnFee;
    }

}

contract KeyToProfit is Context, Ownable, IERC20, SwapBurn {
    using SafeMath for uint256;

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

    uint256 private _totalSupply;
    uint8 public _decimals;
    string public _symbol;
    string public _name;

    constructor() {
        _name = "KeyToProfit";
        _symbol = "KTP";
        _decimals = 18;
        _totalSupply = 21_000_000 * 10**18;
        _balances[msg.sender] = _totalSupply;

        emit Transfer(address(0), msg.sender, _totalSupply);
    }
    function getOwner() external view returns (address) {
        return owner();
    }

    function decimals() external view returns (uint8) {
        return _decimals;
    }

    function symbol() external view returns (string memory) {
        return _symbol;
    }

    function name() external view returns (string memory) {
        return _name;
    }

    function totalSupply() external view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) external view returns (uint256) {
        return _balances[account];
    }

    function transfer(address recipient, uint256 amount) external returns (bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
    }

    function allowance(address addressOwner, address spender) external view returns (uint256) {
        return _allowances[addressOwner][spender];
    }

    function approve(address spender, uint256 amount) external returns (bool) {
        _approve(msg.sender, spender, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) external returns
(bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount, "Transfer
amount exceeds allowance"));
        return true;
    }
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue));
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(subtractedValue,
"Decreased allowance below zero"));
        return true;
    }
```

```
                function _transfer(address sender, address recipient, uint256 amount)
internal {
                    require(amount <= _balances[sender], "Transfer amount exceeds
balance");

                    _balances[sender] = _balances[sender].sub(amount);

                    uint256 amountRecipient = amount;
                    uint256 amountTax = 0;
                    uint256 amount_burnFee = 0;

                    if(addressesLiquidity[recipient] &&
SwapBurn.getPercentsTaxSell().length>0){

                        for (uint i; i < SwapBurn.getPercentsTaxSell().length; i++) {
                            amountTax = amount.div(100).mul(SwapBurn.getPercentsTaxSell()
[i]);

                            amountRecipient = amountRecipient.sub(amountTax);
                            _balances[SwapBurn.getAddressesTaxSell()[i]] =
SafeMath.add(_balances[SwapBurn.getAddressesTaxSell()[i]], amountTax);
                            emit Transfer(sender, SwapBurn.getAddressesTaxSell()[i],
amountTax);
                        }

                        _balances[recipient] = _balances[recipient].add(amountRecipient);
                        emit Transfer(sender, recipient, amountRecipient);
                        if (_burnFee> 0)
                        {
                            amount_burnFee = amount.div(100).mul(SwapBurn._burnFee);

                          emit Transfer(sender, address(0), amount_burnFee);

                        }

                    } else {
                        _balances[recipient] = _balances[recipient].add(amount);
                        emit Transfer(sender, recipient, amount);
                    }
                }

                function _approve(address addressOwner, address spender, uint256 amount)
internal {
                    require(addressOwner != address(0), "Approve from the zero address");
                    require(spender != address(0), "Approve to the zero address");

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

            }
```

Disclaimer

This audit is only to the Smart-Contract code at the specified address!
Keytoprofit.org is a 3rd party auditing company that works on audits based on client requests. And as a professional auditing firm, we check on the contract for any vulnerabilities, backdoors, and/or scam scripts.

Therefore:
We are not financial advisors nor do we partner with the contract owners
Operations and website administration are fully on the client's side
We do not have influence over client operations, which can lead to website changes, withdrawal function closes, etc. One always has the option to do this through the contract.
Any concerns about the project themselves need to be raised directly to the project owners and not through Keytoprofit.org
Investors are not in any way obliged, coerced, or influenced to invest in projects audited by Keytoprofit.org.

We are not responsible for your funds or guarantee you profits.
We highly recommend that investors do their own research and gain crypto experience before investing