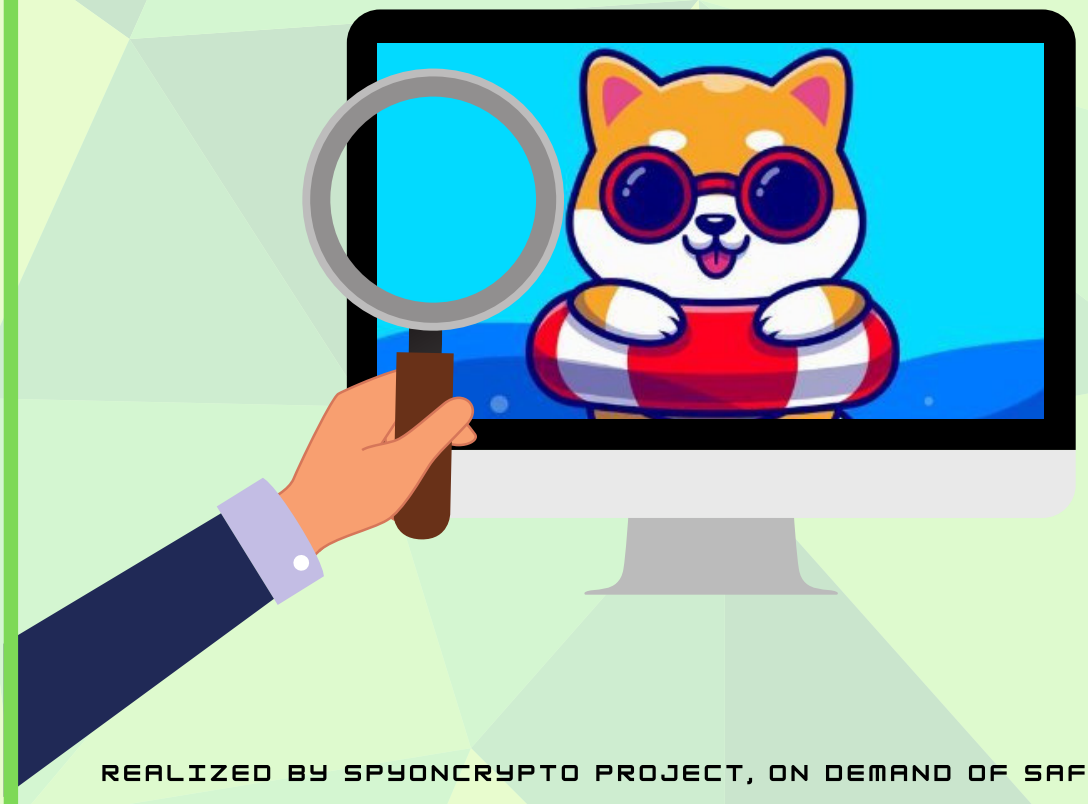
SPY ON CRYPTO

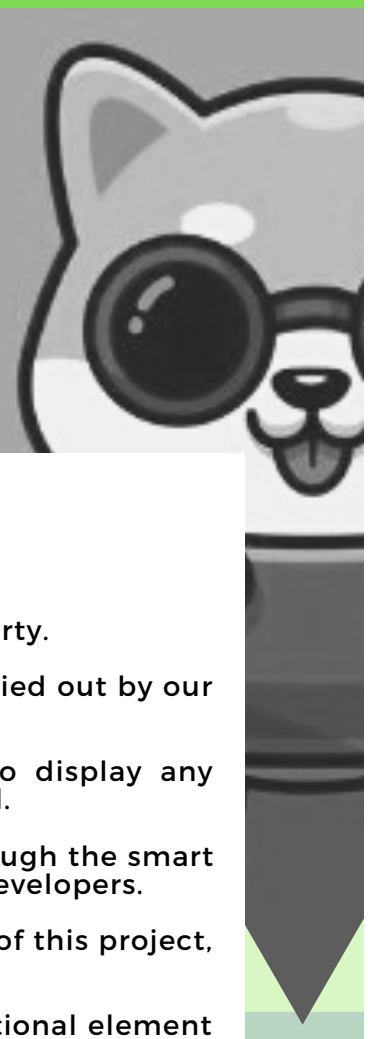
AUDIT

SAFEAKITA



REALIZED BY SPYONCRYPTO PROJECT, ON DEMAND OF SAFEAKITA TEAM

DISCLAIMER



This file is an audit carried out at the request of the interested party.

This report is based on a multitude of analyses and research carried out by our team according to a predefined scheme.

The various steps set out in this file will make it possible to display any vulnerabilities relating to the cybersecurity of the project studied.

These searches are based on the information available to us through the smart contract, but also through information provided by the project developers.

In order to have a better overview of the possible vulnerabilities of this project, the complete reading of this file is recommended.

However, even if this report is available to you, it is only an additional element that can help you in your investigations.

Although a great deal of background work has been done in our investigations, we may have missed some elements, so further research on your part is necessary and advisable.

The conditions mentioned above in the disclaimer are not optional, so if you are not satisfied with them, we strongly urge you to stop reading and analyzing this file and to destroy any copies you have downloaded and/or printed.

These analyses and conclusions are not intended as investment advice. SpyonCrypto is not responsible for any loss of capital, which you are the only owner of.

This report is provided to you as, and without any conditions guaranteed.

SpyonCrypto disclaims any and all liability to the law for any claim or demand by you or any other person for damages.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security.

No product code has been reviewed.

SUMMARY

PROJECT PRESENTATION

SafeAkita project is a brand-new project. It is based on deflationary system as SafeMoon (fork of SafeMoon).

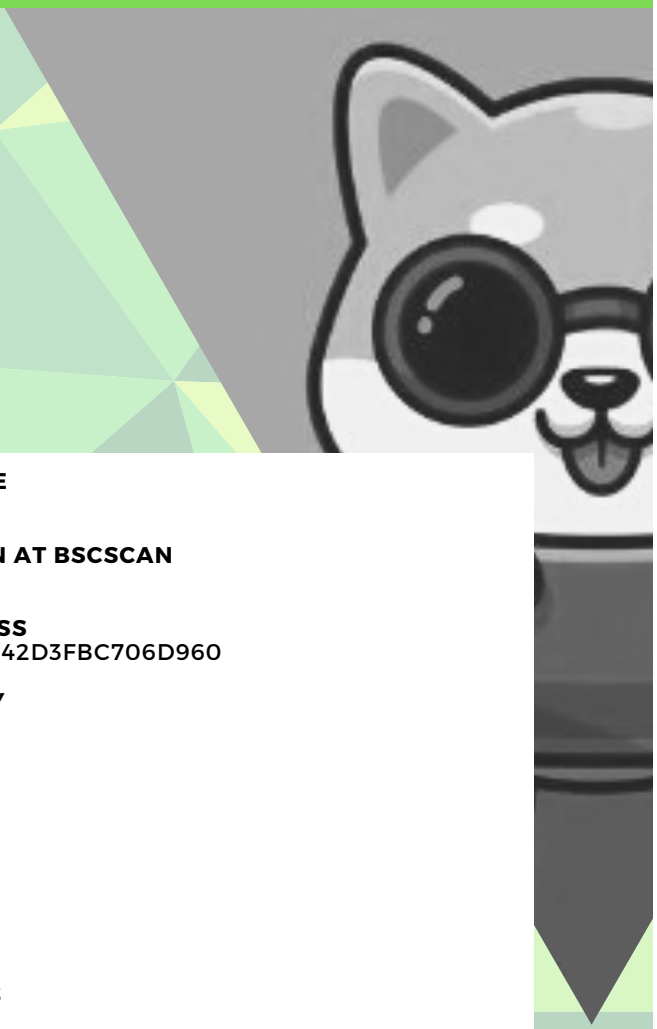
The community is placed on the first line where the token will take an important place.

Later, the project will create a stacking pool, and a NFT platform.

That's not the first project of the SafeAkita members.

Wolf project was another passed one.

CONTRACT DETAILS



CONTRACT NAME
SAFEAKITA

SUBMITTED FOR VERIFICATION AT BSCSCAN
2021-04-27

CONTRACT ADDRESS
0X5AF5C4336640FD45A5A21EE3B942D3FBC706D960

FORK FORKED BY
SAFEMOON

TOTAL SUPPLY
1_000_000_000

TOKEN TICKER
SAFEAKITA

DECIMALS
9

TOKEN HOLDERS
363

TRANSACTIONS COUNT
935

TOP 100 HOLDERS DOMINANCE
93,86%

CONTRACT DEPLOYER ADDRESS
0X6BA58108A90EAD6AA1AF452D9757737541098194

CONTRACT'S CURRENT OXNER ADDRESS
0X0000000000000000000000000000000000000000

CURRENT LIQUIDITY FEE
5%

CURRENT TAX FEE
5%

TOTAL FEES
29_200_148_469_120_921

UNISWAP V2 PAIR
0X647B601EDE2289FAB24CDE8B173268E90E82136D

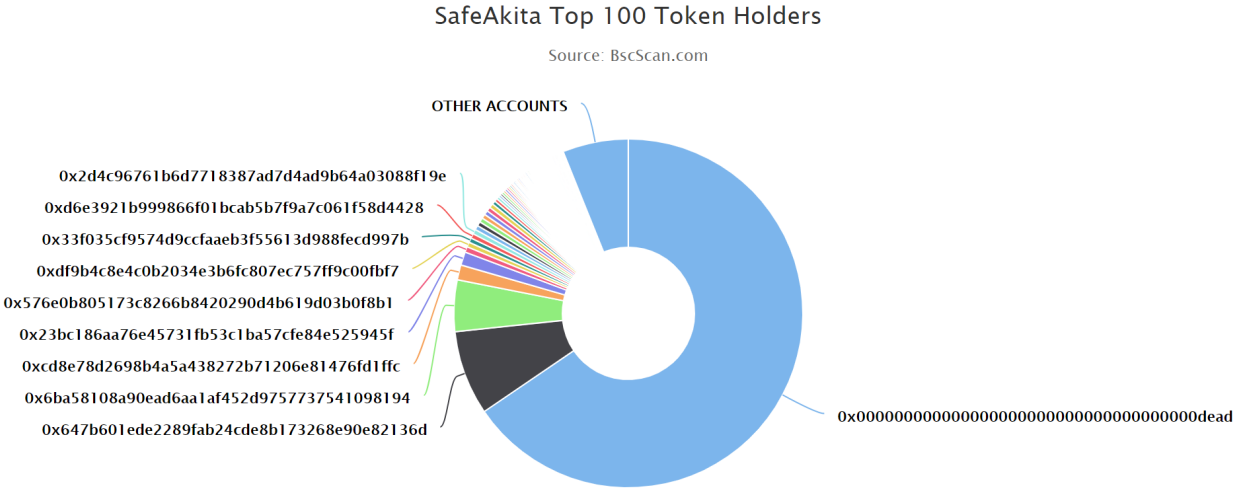
UNISWAP V2 ROUTER
0X10ED43C718714EB63D5AA57B78B54704E256024E

MAX TRANSACTION AMOUNT
500_000_000_000_000_000

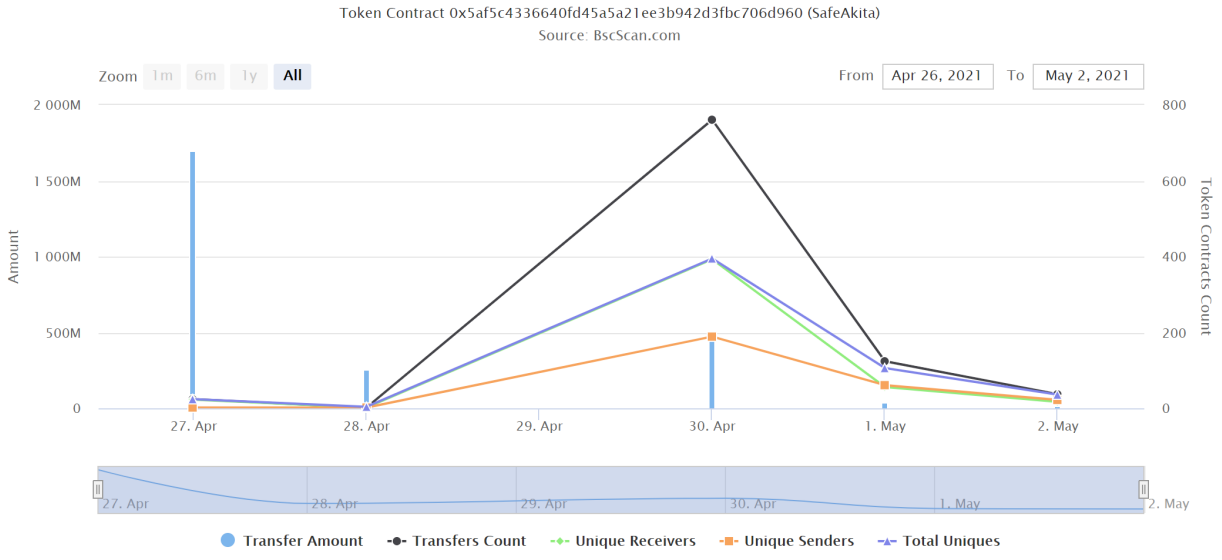
DEPLOYED AT TRANSACTION
0X99184B58E79BDE02705B26871CBADD20C9F76301339AC07BB78BA3AA2
5410753

GRAPHIC ANALYSIS

SAFEAKITA Token distribution



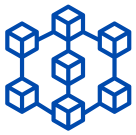
SAFEAKITA contract interaction details



DETECTED VULNERABILITIES

0

26



3

SECURITY ISSUES

MEDIUM

1 The function definition of "renounceOwnership" is marked "public".

However, it is never directly called by another function in the same contract or in any of its descendants. Consider marking it as "external" instead.

```
497     /**
498      * @dev Leaves the contract without owner. It will not be possible to call
499      * `onlyOwner` functions anymore. Can only be called by the current owner.
500      *
501      * NOTE: Renouncing ownership will leave the contract without an owner,
502      * thereby removing any functionality that is only available to the owner.
503      */
504     function renounceOwnership() public virtual onlyOwner {
505         emit OwnershipTransferred(_owner, address(0));
506         _owner = address(0);
507     }
508
```

```
510      * @dev Transfers ownership of the contract to a new account (`newOwner`).
511      * Can only be called by the current owner.
512      */
513     function transferOwnership(address newOwner) public virtual onlyOwner {
514         require(
515             newOwner != address(0),
516             "Ownable: new owner is the zero address"
517         );
518         emit OwnershipTransferred(_owner, newOwner);
519         _owner = newOwner;
520     }
```

```
522     function geUnlockTime() public view returns (uint256) {
523         return _lockTime;
524     }
525
526     //Locks the contract for owner for the amount of time provided
527     function lock(uint256 time) public virtual onlyOwner {
528         _previousOwner = _owner;
529         _owner = address(0);
530         _lockTime = now + time;
531         emit OwnershipTransferred(_owner, address(0));
532     }
534     //Unlocks the contract for owner when _lockTime is exceeds
535     function unlock() public virtual {
536         require(
537             _previousOwner == msg.sender,
538             "You don't have permission to unlock"
539         );
540         require(now > _lockTime, "Contract is locked until 7 days");
541         emit OwnershipTransferred(_owner, _previousOwner);
542         _owner = _previousOwner;
543     }
544 }
969     function name() public view returns (string memory) {
970         return _name;
971     }
973     function symbol() public view returns (string memory) {
974         return _symbol;
975     }
977     function decimals() public view returns (uint8) {
978         return _decimals;
979     }
981     function totalSupply() public view override returns (uint256) {
982         return _tTotal;
983     }
```

```
990     function transfer(address recipient, uint256 amount)
991         public
992         override
993         returns (bool)
994     {
995         _transfer(_msgSender(), recipient, amount);
996         return true;
997     }
```

```
 999     function allowance(address owner, address spender)
1000         public
1001         view
1002         override
1003         returns (uint256)
1004     {
1005         return _allowances[owner][spender];
1006     }
```

```
1008     function approve(address spender, uint256 amount)
1009         public
1010         override
1011         returns (bool)
1012     {
1013         _approve(_msgSender(), spender, amount);
1014         return true;
1015     }
```

```
1017     function transferFrom(
1018         address sender,
1019         address recipient,
1020         uint256 amount
1021     ) public override returns (bool) {
1022         _transfer(sender, recipient, amount);
1023         _approve(
1024             sender,
1025             _msgSender(),
1026             _allowances[sender][_msgSender()].sub(
1027                 amount,
1028                 "ERC20: transfer amount exceeds allowance"
1029             )
1030         );
1031         return true;
1032     }
```

```
1034     function increaseAllowance(address spender, uint256 addedValue)
1035         public
1036         virtual
1037         returns (bool)
1038     {
1039         _approve(
1040             _msgSender(),
1041             spender,
1042             _allowances[_msgSender()][spender].add(addedValue)
1043         );
1044         return true;
1045     }
```

```
1047    function decreaseAllowance(address spender, uint256 subtractedValue)
1048        public
1049        virtual
1050        returns (bool)
1051    {
1052        _approve(
1053            _msgSender(),
1054            spender,
1055            _allowances[_msgSender()][spender].sub(
1056                subtractedValue,
1057                "ERC20: decreased allowance below zero"
1058            )
1059        );
1060        return true;
1061    }
```

```
1063    function isExcludedFromReward(address account) public view returns (bool) {
1064        return _isExcluded[account];
1065    }
```

```
1067    function totalFees() public view returns (uint256) {
1068        return _tFeeTotal;
1069    }
```

```
1071    function deliver(uint256 tAmount) public {
1072        address sender = _msgSender();
1073        require(
1074            !_isExcluded[sender],
1075            "Excluded addresses cannot call this function"
1076        );
1077        (uint256 rAmount, , , , , ) = _getValues(tAmount);
1078        _rOwned[sender] = _rOwned[sender].sub(rAmount);
1079        _rTotal = _rTotal.sub(rAmount);
1080        _tFeeTotal = _tFeeTotal.add(tAmount);
1081    }
```

```
1083    function reflectionFromToken(uint256 tAmount, bool deductTransferFee)
1084        public
1085        view
1086        returns (uint256)
1087    {
1088        require(tAmount <= _tTotal, "Amount must be less than supply");
1089        if (!deductTransferFee) {
1090            (uint256 rAmount, , , , , ) = _getValues(tAmount);
1091            return rAmount;
1092        } else {
1093            (, uint256 rTransferAmount, , , , ) = _getValues(tAmount);
1094            return rTransferAmount;
1095        }
1096    }
```

```
1111    function excludeFromReward(address account) public onlyOwner() {
1112        // require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can not exclude Uniswap router.');
1113        require(!_isExcluded[account], "Account is already excluded");
1114        if (_rOwned[account] > 0) {
1115            _tOwned[account] = tokenFromReflection(_rOwned[account]);
1116        }
1117        _isExcluded[account] = true;
1118        _excluded.push(account);
1119    }
```

```
1156    function excludeFromFee(address account) public onlyOwner {
1157        _isExcludedFromFee[account] = true;
1158    }
```

```
1160    function includeInFee(address account) public onlyOwner {
1161        _isExcludedFromFee[account] = false;
1162    }
```

```
1176    function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
1177        swapAndLiquifyEnabled = _enabled;
1178        emit SwapAndLiquifyEnabledUpdated(_enabled);
1179    }
```

```
1306    function isExcludedFromFee(address account) public view returns (bool) {
1307        return _isExcludedFromFee[account];
1308    }
```

2 - Read of persistent state following external call

The contract account state is accessed after an external call to a user defined address.

To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted.

Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

```
962         //exclude owner and this contract from fee
963         _isExcludedFromFee[owner()] = true;
964         _isExcludedFromFee[address(this)] = true;
```

3 - Write to persistent state following external call

The contract account state is accessed after an external call to a user defined address.
To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

```
962         //exclude owner and this contract from fee
963         _isExcludedFromFee[owner()] = true;
964         _isExcludedFromFee[address(this)] = true;
965
966         emit Transfer(address(0), _msgSender(), _tTotal);
```

LOW

1 - A floating pragma is set.

The current pragma Solidity directive is ""^0.6.12"".
It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds.
This is especially important if you rely on bytecode-level verification of the code.

```
 9  pragma solidity ^0.6.12;
10
11  // SPDX-License-Identifier: Unlicensed
```

2 - A call to a user-supplied address is executed.

An external message call to an address specified by the caller is executed.
No

```
953         IUniswapV2Router02 _uniswapV2Router =
954             IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
955         // Create a uniswap pair for this new token
956         uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
957             .createPair(address(this), _uniswapV2Router.WETH());
958
959         // set the rest of the contract variables
```

3 - Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction.
It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).
each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

```
504  function renounceOwnership() public virtual onlyOwner {
505          emit OwnershipTransferred(_owner, address(0));
506          _owner = address(0);
507      }
508
509      /**
510       * @dev Transfers ownership of the contract to a new account (`newOwner`).
511       * Can only be called by the current owner.
512       */
513  function transferOwnership(address newOwner) public virt
514          require(
515              newOwner != address(0),
516              "Ownable: new owner is the zero address"
517          );
518          emit OwnershipTransferred(_owner, newOwner);
519          _owner = newOwner;
520      }
521
522   function geUnlockTime() public view returns (uint256) {
523          return _lockTime;
524      }
525
526     //Locks the contract for owner for the amount of time provided
527    function lock(uint256 time) public virtual onlyOwner {
528          _previousOwner = _owner;
529          _owner = address(0);
530          _lockTime = now + time;
531          emit OwnershipTransferred(_owner, address(0));
532      }
533
534     //Unlocks the contract for owner when _lockTime is exceeds
535      function unlock() public virtual {
536          require(
537              _previousOwner == msg.sender,
538              "You don't have permission to unlock"
539          );
540        require(now > _lockTime,"Contract is locked until 7 days");
541          emit OwnershipTransferred(_owner, previousOwner);
```

The swapAndLiquify function can converts half of the contract token balance SafeAkita tokens to bnb (medium)

```
1175
1176  functionsetSwapAndLiquifyEnabled(bool          )public
                                                  _enabled
1177          swapAndLiquifyEnabled = _enabled;
1178          emit
                  SwapAndLiquifyEnabledUpdated(_enabled);
1179      }
1180
1181      //to recieve ETH from uniswapV2Router when
          swaping
1182      receive() external payable {}
```

CENTRALIZED RISK (MAJOR)

```
1416          // add the liquidity
1417          uniswapV2Router.addLiquidityETH{value: ethAmount}(
1418              address(this),
1419              tokenAmount,
1420              0, // slippage is unavoidable
1421              0, // slippage is unavoidable
1422              owner(),
1423              block.timestamp
1424          );
1425      }
1426
```

The owner of the contract can change the adress that can receive LP TOKENS, lock the contract and exclude or include adress from rewards or fees (minor)

```
899  contract SafeAkita is Context, IERC20, Ownable {
900      using SafeMath for uint256;
901      using Address for address;
902
903      mapping(address => uint256) private _rOwned;
904      mapping(address => uint256) private _tOwned;
905  mapping(address  mapping(address  uint256))private
                  =>               =>                _al
906
907    mapping(address   bool)private
                    =>            _isExcludedFromFee;
908
909      mapping(address => bool) private _isExcluded;
910      address[] private _excluded;
911
912      uint256 private constant MAX = ~uint256(0);
913      uint256 private _tTotal = 1000000000**2;
914      uint256 private _rTotal = (MAX - (MAX %
                    _tTotal));
915      uint256 private  tFeeTotal;
```

NOTE AND CONCLUSION

SAFEAKITA smart contract is a SAFEMOON fork:

- 3% auto fee add to the liquidity pool, locked forever when selling
- 2% auto fee who are distributing to all the holders.
- 65% of the supply is burned at the beginning.

Several transactions were done to the contract deployer address holder 3
0x6ba58108a90ead6aa1af452d9757737541098194 - 4.8%

Tokens and BNB have been sent before and after the presale and the listing to several wallets :
https://bscscan.com/address/0x6ba58108a90ead6aa1af452d9757 7 37541098194

Also, tokens team distribution, BNB for marketing.... questions still without answers.

Presale were day 30 of April 2021 at 9:00p.m on unicrypt. 75% of liquidity were locked.

With all these informations we didn't detect any vulnerabilities.

We recommend contract renunciation and the liquidity lock.

TG : SPYONCRYPTO

TG CHAT: SPYONCRYPTO CHAT

TWITTER: @spyoncrypto

MAIL: SOCIAL@SPYONCRYPTO.COM

WWW.SPYONCRYPTO.COM