Coinsult

Advanced Manual
Smart Contract Audit

March 2, 2023

CoinsultAudits

info@coinsult.net

coinsult.net

Audit requested by

🦹 Flokita

0xff107d6b39737828552a4b2f2a7e5cb33a2a8423

Table of Contents

Coinsult

Audit Summary

Project Name	Flokita
Website	https://flokita.online/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0xff107d6b39737828552a4b2f2a7e5cb33a2a8423
Audit Method	Static Analysis, Manual Review
Date of Audit	2 March 2023

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Audit Scope

Coinsult was comissioned by Flokita to perform an audit based on the following code:

https://bscscan.com/address/0xff107d6b39737828552a4b2f2a7e5cb33a2a8423#code

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

Used tools

- Slither: Solidity static analysis framework
- Remix: IDE Developer Tool
- CWE: Common Weakness Enumeration
- SWC: Smart Contract Weakness Classification and Test Cases
- DEX: Testnet Blockchains

Risk Classification

Coinsult uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

Vulnerability Level	Description
🔵 Informational	Does not compromise the functionality of the contract in any way
🟢 Low-Risk	Won't cause any problems, but can be adjusted for improvement
🟡 Medium-Risk	Will likely cause problems and it is recommended to adjust
🔴 High-Risk	Will definitely cause problems, this needs to be adjusted

Coinsult has four statuses that are used for each risk level. Below we explain them briefly.

Risk Status	Description
Total	Total amount of issues within this category
Pending	Risks that have yet to be addressed by the team
Acknowledged	The team is aware of the risks but does not resolve them
Resolved	The team has resolved and remedied the risk

SWC Attack Analysis

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID	Description	Status
SWC-100	Function Default Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Failed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed
SWC-107	Reentrancy	Passed
SWC-108	State Variable Default Visibility	Failed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed

SWC-116	Block values as a proxy for time	Passed
SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed
SWC-123	Requirement Violation	Passed
SWC-124	Write to Arbitrary Storage Location	Passed
SWC-125	Incorrect Inheritance Order	Passed
SWC-126	Insufficient Gas Griefing	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed
SWC-128	DoS With Block Gas Limit	Passed
SWC-129	Typographical Error	Passed
SWC-130	Right-To-Left-Override control character (U+202E)	Passed
SWC-131	Presence of unused variables	Passed
SWC-132	Unexpected Ether balance	Passed
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Passed
SWC-134	Message call with hardcoded gas amount	Passed
SWC-135	Code With No Effects	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed

Global Overview

Manual Code Review

In this audit report we will highlight the following issues:

Vulnerability Level	Total	Pending	Acknowledged	Resolved
🔵 Informational	0	0	0	0
🟢 Low-Risk	11	11	0	0
🟡 Medium-Risk	2	2	0	0
🔴 High-Risk	1	1	0	0

Centralization Risks

Coinsult checked the following privileges:

Contract Privilege	Description
Owner can mint?	🟢 Owner cannot mint new tokens
Owner can blacklist?	🔴 Owner can blacklist addresses
Owner can set fees > 25%?	🔴 Owner can set the sell fee to 25% or higher
Owner can exclude from fees?	🔵 Owner can exclude from fees
Owner can pause trading?	🟢 Owner cannot pause the contract
Owner can set Max TX amount?	🔴 Owner can set max transaction amount

More owner priviliges are listed later in the report.

Error Code	Description
CS-01	wrong require return statement

🟢 **Low-Risk:** Could be fixed, will not bring problems.

wrong require return statement

```
function setTax(uint256 _liquidityFee, uint256 _teamFee, uint256 _marketingFee, uint256 _devFee, uint256
    liquidityFee = _liquidityFee;
    teamFee = _teamFee;
    marketingFee = _marketingFee;
    devFee = _devFee;
    stakingFee = _stakingFee;
    totalFee = _liquidityFee.add(_teamFee).add(_marketingFee).add(_devFee).add(_stakingFee);
    feeDenominator = _feeDenominator;
    require(totalFee &lt; feeDenominator/5, &quot;Buy Fees cannot be more than 20%&quot;);
}
```

Recommendation

Fees are not only applied to buy but also to sell, change the return statement to a valid return statement

Error Code	Description
SLT: 078	Conformance to numeric notation best practices

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Too many digits

Literals with many digits are difficult to read and review.

```
uint256 public swapThreshold = _totalSupply * 20 / 10000;
```

Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

Exploit scenario

```
contract MyContract{
    uint 1_ether = 10000000000000000000;
}
```

While 1_ether looks like 1 ether, it is 10 ether. As a result, it's likely to be used incorrectly.

Error Code	Description
SLT: 056	Missing Zero Address Validation

🟢 **Low-Risk:** Could be fixed, will not bring problems.

No zero address validation for some functions

Detect missing zero address validation.

```
function setTaxReceivers(address _autoLiquidityReceiver, address _marketingFeeReceiver, address _devFee
    autoLiquidityReceiver = _autoLiquidityReceiver;
    marketingFeeReceiver = _marketingFeeReceiver;
    devFeeReceiver = _devFeeReceiver;
    stakingFeeReceiver = _stakingFeeReceiver;
    teamFeeReceiver = _teamFeeReceiver;
}
```

Recommendation

Check that the new address is not zero.

Exploit scenario

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

Bob calls updateOwner without specifying the newOwner, soBob loses ownership of the contract.

Error Code	Description
SLT: 016	Functions that send Ether to arbitrary destinations

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Functions that send Ether to arbitrary destinations

Unprotected call to a function sending Ether to an arbitrary address.

```
function transfer() external {
    payable(devFeeReceiver).transfer(address(this).balance);

}
```

Recommendation

Ensure that an arbitrary user cannot withdraw unauthorized funds.

Exploit scenario

```
contract ArbitrarySend{
    address destination;
    function setDestination(){
        destination = msg.sender;
    }

    function withdraw() public{
        destination.transfer(this.balance);
    }
}
```

Bob calls `setDestination` and `withdraw`. As a result he withdraws the contract's balance.

Error Code	Description
SWC: 103	Floating Pragma

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Floating Pragma

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

```
pragma solidity ^0.8.11;
```

Recommendation

Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

Error Code	Description
SWC: 108	State variable visibility is not set.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

State Variable Default Visibility

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

```
address WBNB;
uint256 _totalSupply = 1 * 10**9 * 10**_decimals;
mapping (address => uint256) _balances;
mapping (address => mapping (address => uint256)) _allowances;
mapping (address => bool) isantiboted;
mapping (address => bool) isFeeExempt;
mapping (address => bool) isTxLimitExempt;
uint256 sellMultiplier = 200;
uint256 buyMultiplier = 100;
uint256 transferMultiplier = 600;
uint256 launchBlock;
bool inSwap;
```

Recommendation

Variables can be specified as being `public`, `internal` or `private`. Explicitly define visibility for all state variables.

Error Code	Description
SLT: 038	Imprecise arithmetic operations order

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Divide before multiply

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

```
uint256 feeAmount = amount.mul(totalFee).mul(multiplier).div(feeDenominator * 100);
```

Recommendation

Consider ordering multiplication before division.

Exploit scenario

```solidity
contract A {
    function f(uint n) public {
        coins = (oldSupply / n) * interest;
    }
}
```

If n is greater than `oldSupply`, `coins` will be zero. For example, with `oldSupply = 5; n = 10`, `interest = 2`, coins will be zero. If (`oldSupply * interest / n`) was used, `coins` would have been 1. In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

Error Code	Description
SLT: 054	Missing Events Arithmetic

● **Low-Risk:** Could be fixed, will not bring problems.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setFeeDistribution(uint256 _buy, uint256 _sell, uint256 _trans) external onlyOwner {
    sellMultiplier = _sell;
    buyMultiplier = _buy;
    transferMultiplier = _trans;

    require(totalFee.mul(buyMultiplier).div(100) &lt; 35, &quot;Tax cannot be more than 35%&quot;);
    require(totalFee.mul(sellMultiplier).div(100) &lt; 35, &quot;Tax cannot be more than 35%&quot;);

}

// switch Trading
function OpenTrading() public onlyOwner {
    tradingOpen = true;
    launchBlock = block.number;
}
```

Recommendation

Emit an event for critical parameter changes.

Exploit scenario

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

`updateOwner()` has no event, so it is difficult to track off-chain changes in the buy price.

Error Code	Description
SLT: 068	Conformity to Solidity naming conventions

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Conformance to Solidity naming conventions

Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

```solidity
function updateMax (uint256 _MaxGas) public onlyOwner {
        MaxGas = _MaxGas * 1 gwei;

}
```

Recommendation

Follow the Solidity naming convention.

Rule exceptions

- Allow constant variable name/symbol/decimals to be lowercase (ERC20).
- Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

Error Code	Description
CS: 071	Using safemath in Solidity 0.8.0+

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Using safemath in Solidity 0.8.0+

SafeMath is generally not needed starting with Solidity 0.8, since the compiler now has built in overflow checking.

```solidity
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.
```

Recommendation

Check if you really need SafeMath and consider removing it.

Error Code	Description
CS: 016	Initial Supply

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Initial Supply

When the contract is deployed, the contract deployer receives all of the initially created assets. Since the deployer and/or contract owner can distribute tokens without consulting the community, this could be a problem.

Recommendation

Private keys belonging to the employer and/or contract owner should be stored properly. The initial asset allocation procedure should involve consultation with the community.

Error Code	Description
CS: 017	Reliance on third-parties

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Reliance on third-parties

Interaction between smart contracts with third-party protocols like Uniswap and Pancakeswap. The audit's scope presupposes that third party entities will perform as intended and treats them as if they were black boxes. In the real world, third parties can be hacked and used against you. Additionally, improvements made by third parties may have negative effects, such as higher transaction costs or the deprecation of older routers.

Recommendation

Regularly check third-party dependencies, and when required, reduce severe effects.

Error Code	Description
CSM-01	two functions with the same function name

🟡 **Medium-Risk:** Should be fixed, could bring problems.

two functions with the same function name

```
function transfer() external {
        payable(devFeeReceiver).transfer(address(this).balance);

    }

function transfer(address recipient, uint256 amount) external override returns (bool) {
      return _transferFrom(msg.sender, recipient, amount);
    }
```

Recommendation

Change the name of one of the function to prevent wrong function use

Error Code	Description
CSM-02	Wrong require return statement

🟡 **Medium-Risk:** Should be fixed, could bring problems.

Wrong require return statement

```solidity
function setFeeDistribution(uint256 _buy, uint256 _sell, uint256 _trans) external onlyOwner {
    sellMultiplier = _sell;
    buyMultiplier = _buy;
    transferMultiplier = _trans;

    require(totalFee.mul(buyMultiplier).div(100) < 35, "Tax cannot be more than 35%");
    require(totalFee.mul(sellMultiplier).div(100) < 35, "Tax cannot be more than 35%");

}
```

Recommendation

Transfer tax can be more than 35%, change the requirement return statement to a correct return statement

Error Code	Description
CSH-01	Addresses can be blacklisted if MaxGas is set arbitrary low

🔴 **High-Risk:** Must be fixed, will bring problems.

Addresses can be blacklisted if MaxGas is set arbitrary low

```
function updateMax (uint256 _MaxGas) public onlyOwner {
            MaxGas = _MaxGas * 1 gwei;

    }

if (tx.gasprice &gt;= MaxGas &amp;&amp; recipient != pair) {
          isblacklisted[recipient] = true;
        }
```

Recommendation

Require this MaxGas fee to prevent misuse

Maximum Fee Limit Check

Error Code	Description
CEN-01	Centralization: Operator Fee Manipulation

Coinsult tests if the owner of the smart contract can set the transfer, buy or sell fee to 25% or more. It is bad practice to set the fees to 25% or more, because owners can prevent healthy trading or even stop trading when the fees are set too high.

Type of fee	Description
Transfer fee	🔴 Owner can set the transfer fee to 25% or higher
Buy fee	🔴 Owner can set the buy fee to 25% or higher
Sell fee	🔴 Owner can set the sell fee to 25% or higher

Type of fee	Description
Max transfer fee	100%
Max buy fee	35%
Max sell fee	35%

Coinsult

Contract Pausability Check

Error Code	Description
CEN-02	Centralization: Operator Pausability

Coinsult tests if the owner of the smart contract has the ability to pause the contract. If this is the case, users can no longer interact with the smart contract; users can no longer trade the token.

Privilege Check	Description
Can owner pause the contract?	🟢 Owner cannot pause the contract

Max Transaction Amount Check

Error Code	Description
CEN-03	Centralization: Operator Transaction Manipulation

Coinsult tests if the owner of the smart contract can set the maximum amount of a transaction. If the transaction exceeds this limit, the transaction will revert. Owners could prevent normal transactions to take place if they abuse this function.

Privilege Check	Description
Can owner set max tx amount?	🔴 Owner can set max transaction amount

Function

```
function updateMaxTx(uint256 maxTXPercentage_base1000) external onlyOwner {
    require(maxTXPercentage_base1000 &gt;= 5,"Cannot set max transaction less than 0.5%");
    _maxTxAmount = (_totalSupply * maxTXPercentage_base1000 ) / 1000;
}
```

Exclude From Fees Check

Error Code	Description
CEN-04	Centralization: Operator Exclusion

Coinsult tests if the owner of the smart contract can exclude addresses from paying tax fees. If the owner of the smart contract can exclude from fees, they could set high tax fees and exclude themselves from fees and benefit from 0% trading fees. However, some smart contracts require this function to exclude routers, dex, cex or other contracts / wallets from fees.

Privilege Check	Description
Can owner exclude from fees?	🟢 Owner can exclude from fees

Function

```
function setIsFeeExempt(address[] calldata addresses, bool status) external onlyOwner {
    require(addresses.length < 501,"GAS Error: max limit is 500 addresses");
    for (uint256 i; i < addresses.length; ++i) {
        isFeeExempt[addresses[i]] = status;
    }
}
```

Ability To Mint Check

Error Code	Description
CEN-05	Centralization: Operator Increase Supply

Coinsult tests if the owner of the smart contract can mint new tokens. If the contract contains a mint function, we refer to the token's total supply as non-fixed, allowing the token owner to "mint" more tokens whenever they want.

A mint function in the smart contract allows minting tokens at a later stage. A method to disable minting can also be added to stop the minting process irreversibly.

Minting tokens is done by sending a transaction that creates new tokens inside of the token smart contract. With the help of the smart contract function, an unlimited number of tokens can be created without spending additional energy or money.

Privilege Check	Description
Can owner mint?	🟢 Owner cannot mint new tokens

Coinsult

Ability To Blacklist Check

Error Code	Description
CEN-06	Centralization: Operator Dissalows Wallets

Coinsult tests if the owner of the smart contract can blacklist accounts from interacting with the smart contract. Blacklisting methods allow the contract owner to enter wallet addresses which are not allowed to interact with the smart contract.

This method can be abused by token owners to prevent certain / all holders from trading the token. However, blacklists might be good for tokens that want to rule out certain addresses from interacting with a smart contract.

Privilege Check	Description
Can owner blacklist?	🔴 Owner can blacklist addresses

Function

```
function addtoblacklist(address[] calldata addresses, bool status) public onlyOwner {
    for (uint256 i; i &lt; addresses.length; ++i) {
        isblacklisted[addresses[i]] = status;
    }
}

function addtoantibot(address[] calldata addresses, bool status) public onlyOwner {
    for (uint256 i; i &lt; addresses.length; ++i) {
        isantiboted[addresses[i]] = status;
    }
}
```

Other Owner Privileges Check

Error Code	Description
CEN-100	Centralization: Operator Priviliges

Coinsult lists all important contract methods which the owner can interact with.

⚠ Owner can open trading whenever he wants

⚠ Owner can withdraw tokens from the contract balance

⚠ Owner can set max holding balance

⚠ Owner can exclude addresses from max holding limit

⚠ Owner can exclude addresses from max transaction amount

⚠ Owner can set targetliquidity level

⚠ Authorized wallets can trade before trading is opened

Notes

Notes by Flokita

No notes provided by the team.

Notes by Coinsult

No notes provided by Coinsult

Contract Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

```
contract Flokita is Ownable, IBEP20 {
using SafeMath for uint256;

address WBNB;
address constant DEAD = 0x000000000000000000000000000000000000dEaD;
address constant ZERO = 0x0000000000000000000000000000000000000000;

string constant _name = "Flokita";
string constant _symbol = "Flokita";
uint8 constant _decimals = 9;
```

Coinsult

Website Review

Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.
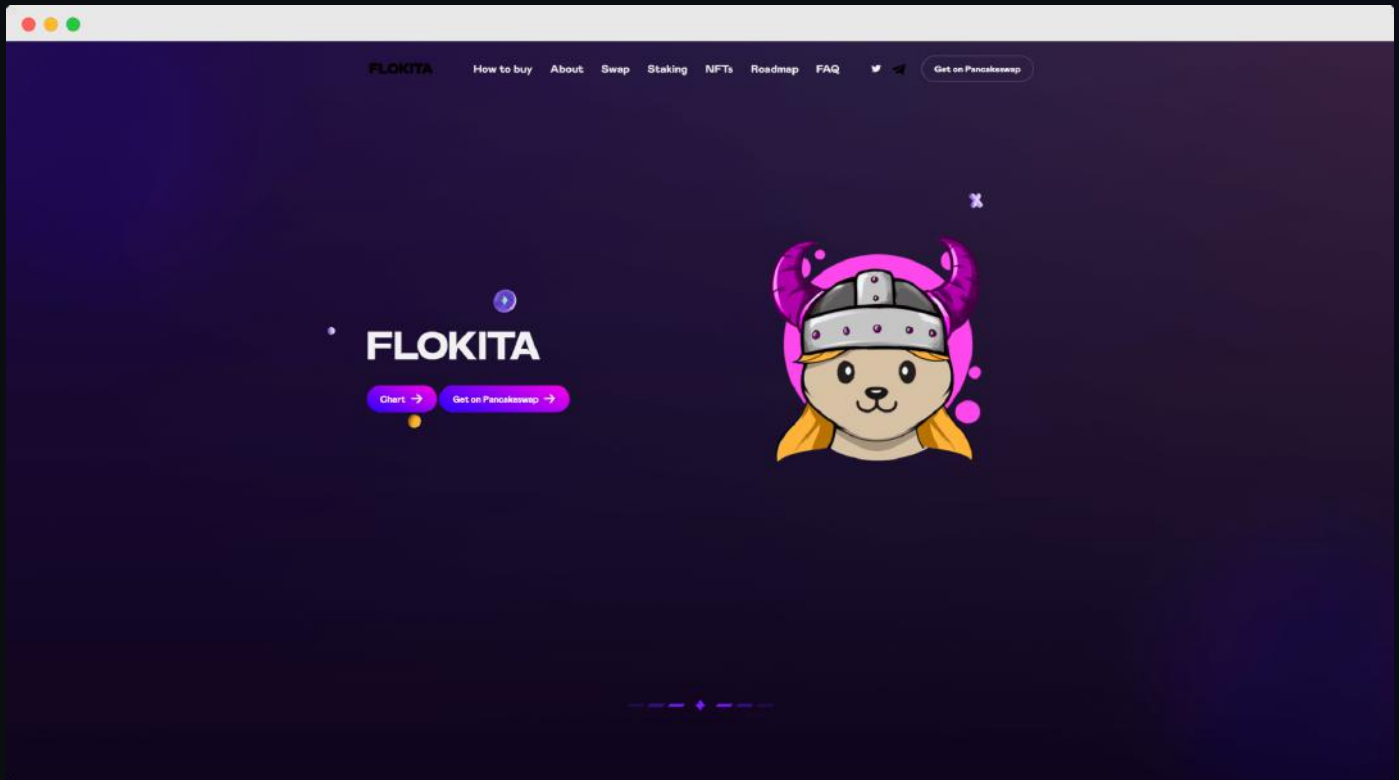
Type of check	Description
Mobile friendly?	🟢 The website is mobile friendly
Contains jQuery errors?	🟢 The website does not contain jQuery errors
Is SSL secured?	🟢 The website is SSL secured
Contains spelling errors?	🟢 The website does not contain spelling errors

Coinsult

Certificate of Proof

🟡 Not KYC verified by Coinsult

Flokita

Audited by Coinsult.net

Date: 2 March 2023

✔ Advanced Manual Smart Contract Audit

Coinsult

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Coinsult

End of report
Smart Contract Audit

🐦 CoinsultAudits

✉ info@coinsult.net

🌐 coinsult.net

Request your smart contract audit / KYC

t.me/coinsult_tg