

Advanced Manual **Smart Contract Audit**

October 7, 2022

Audit requested by



Swish Coin

0x13AF744AE84040DD979dFDc1351D4daA1bcd151

Table of Contents

1. Audit Summary

- 1.1 Audit scope
- 1.2 Tokenomics
- 1.3 Source Code

2. Disclaimer

3. Global Overview

- 3.1 Informational issues
- 3.2 Low-risk issues
- 3.3 Medium-risk issues
- 3.4 High-risk issues

4. Vulnerabilities Findings

5. Contract Privileges

- 5.1 Maximum Fee Limit Check
- 5.2 Contract Pausability Check
- 5.3 Max Transaction Amount Check
- 5.4 Exclude From Fees Check
- 5.5 Ability to Mint Check
- 5.6 Ability to Blacklist Check
- 5.7 Owner Privileges Check

6. Notes

- 6.1 Notes by Coinsult
- 6.2 Notes by Swish Coin

7. Contract Snapshot

8. Website Review

9. Certificate of Proof

Audit Summary

Audit Scope

Project Name	Swish Coin
Website	https://swishfinance.net/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x13AF744AE84040DD979dFDc1351D4daA1bcd151
Audit Method	Static Analysis, Manual Review
Date of Audit	7 October 2022

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.

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0xae30339ef81829ffd614ab12c4bc5774586f619c	1,000,000,000	100.0000%

Source Code

Coinsult was commissioned by Swish Coin to perform an audit based on the following code:

<https://bscscan.com/address/0x13AF744AE84040DD979dFDc1351D4daA1bcdf151#code>

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.

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	4	4	0	0
● Medium-Risk	1	1	0	0
● High-Risk	1	1	0	0

Privilege Overview

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 can pause the smart contract
Owner can set Max TX amount?	● Owner can set max transaction amount

More owner privileges are listed later in the report.

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

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
if(isSell){
    require(amount<=sellLimit,"Sell amount exceeds the max sell amount that prevents du
    require(!_botWatched.contains(sender) == false, "Seller's address caught by Bot Wa
    if (block.timestamp <= tradeStartedAt + botWatcherTime && botWatcherEnabled == 1
        _botWatched.add(sender);
        emit botWatcher(sender);
    }
    tax=_sTax;

} else if(isBuy){
    require(recipientBalance+amount<=balanceLimit,"Amount would exceed recipients max b
    require(amount <= maxBuyAmount,"Buy amount exceeds max buy amount");
    require(!_botWatched.contains(recipient) == false, "Buyer's address caught by Bot I
    if (block.timestamp <= tradeStartedAt + botWatcherTime && botWatcherEnabled == 1
        _botWatched.add(recipient);
        emit botWatcher(recipient);
    }
    tax=_bTax;

}
```

Recommendation

Do not use block.timestamp, now or blockhash as a source of randomness

Exploit scenario

```
contract Game {

    uint reward_determining_number;

    function guessing() external{
        reward_determining_number = uint256(block.blockhash(10000)) % 10;
    }

}
```

Eve is a miner. Eve calls guessing and re-orders the block containing the transaction. As a result, Eve wins the game.

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

Too many digits

Literals with many digits are difficult to read and review.

```
uint256 public constant InitialSupply= 1000000000 * 10**_decimals;
```

Recommendation

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

Exploit scenario

```
contract MyContract{
    uint 1_ether = 1000000000000000000;
}
```

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

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

Unchecked transfer

The return value of an external transfer/transferFrom call is not checked.

```
function removeDevBNB() external onlyOwner{
    uint256 amount=devAmount;
    devAmount=0;
    payable(devAccount1).transfer((amount*50) / 100);
    payable(devAccount2).transfer((amount-(amount*50) / 100));
}
```

Recommendation

Use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

Exploit scenario

```
contract Token {
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success);
}
contract MyBank{
    mapping(address => uint) balances;
    Token token;
    function deposit(uint amount) public{
        token.transferFrom(msg.sender, address(this), amount);
        balances[msg.sender] += amount;
    }
}
```

Several tokens do not revert in case of failure and return false. If one of these tokens is used in MyBank, deposit will not revert if the transfer fails, and an attacker can call deposit for free..

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

Boolean equality

Detects the comparison to boolean constants.

```
function checkBotWatcher(address submittedAddress) external view returns (bool botWatchTrue) {
    if (_botWatched.contains(submittedAddress) == true) {
        botWatchTrue = true;
        return botWatchTrue;
    }
    if (_botWatched.contains(submittedAddress) == false) {
        botWatchTrue = false;
        return botWatchTrue;
    }
}
```

Recommendation

Remove the equality to the boolean constant.

Exploit scenario

```
contract A {
    function f(bool x) public {
        // ...
        if (x == true) { // bad!
            // ...
        }
        // ...
    }
}
```

Boolean constants can be used directly and do not need to be compare to true or false.

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

Optimize lines of code

```
function checkBotWatcher(address submittedAddress) external view returns (bool botWatchTrue) {  
    if (_botWatched.contains(submittedAddress) == true) {  
        botWatchTrue = true;  
        return botWatchTrue;  
    }  
    if (_botWatched.contains(submittedAddress) == false) {  
        botWatchTrue = false;  
        return botWatchTrue;  
    }  
}
```

Recommendation

You can do: 'return _botWatched.contains(submittedAddress)'

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

Wrong settings in constructor

```
_bTax=5;  
_sTax=10;  
_tTax=10;  
  
_burnTax=0;  
_lpTax=2;  
_devTax=8;
```

Recommendation

$_burnTax + _lpTax + _devTax$ should be 100, instead of 10

Contract Privileges

Maximum Fee Limit Check

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	100%
Max sell fee	100%

Contract Pausability Check

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 can pause the smart contract

Max Transaction Amount Check

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

Exclude From Fees Check

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

Ability To Mint Check

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

Ability To Blacklist Check

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?	<div></div> Owner can blacklist addresses

Other Owner Privileges Check

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

- ⚠ Trading is disabled by default, owner can enable it anytime
- ⚠ Owner can change PancakeSwap Router
- ⚠ Owner can set balance limit
- ⚠ Owner can set Liquidity Lock Time
- ⚠ Owner can remove Liquidity after Liquidity Timer is finished

Notes

Notes by Swish Coin

No notes provided by the team.

Notes by Coinconsult

 No notes provided by Coinconsult

Contract Snapshot

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

```
contract SwishCoin is IBEP20, Ownable
{
    using Address for address;
    using EnumerableSet for EnumerableSet.AddressSet;

    event contractHadAChange(uint256 indexed value);
    event contractBoolChanged(bool indexed value);
    event contractAddressChanged(address indexed value);
    event botWatcher(address indexed value);

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

    EnumerableSet.AddressSet private _excluded;
    EnumerableSet.AddressSet private _botWatched;

    string private constant _name = 'Swish Coin';
    string private constant _symbol = '$SC';
    uint8 private constant _decimals = 9;
    uint256 public constant InitialSupply= 1000000000 * 10**_decimals;

    address private PancakeRouter=0x10ED43C718714eb63d5aA57B78B54704E256024E;
    //address private PancakeRouter=0x9Ac64Cc6e4415144C455BD8E4837Fea55603e5c3;

    uint256 contractTokensToSell = 1000000000 * 10**_decimals;

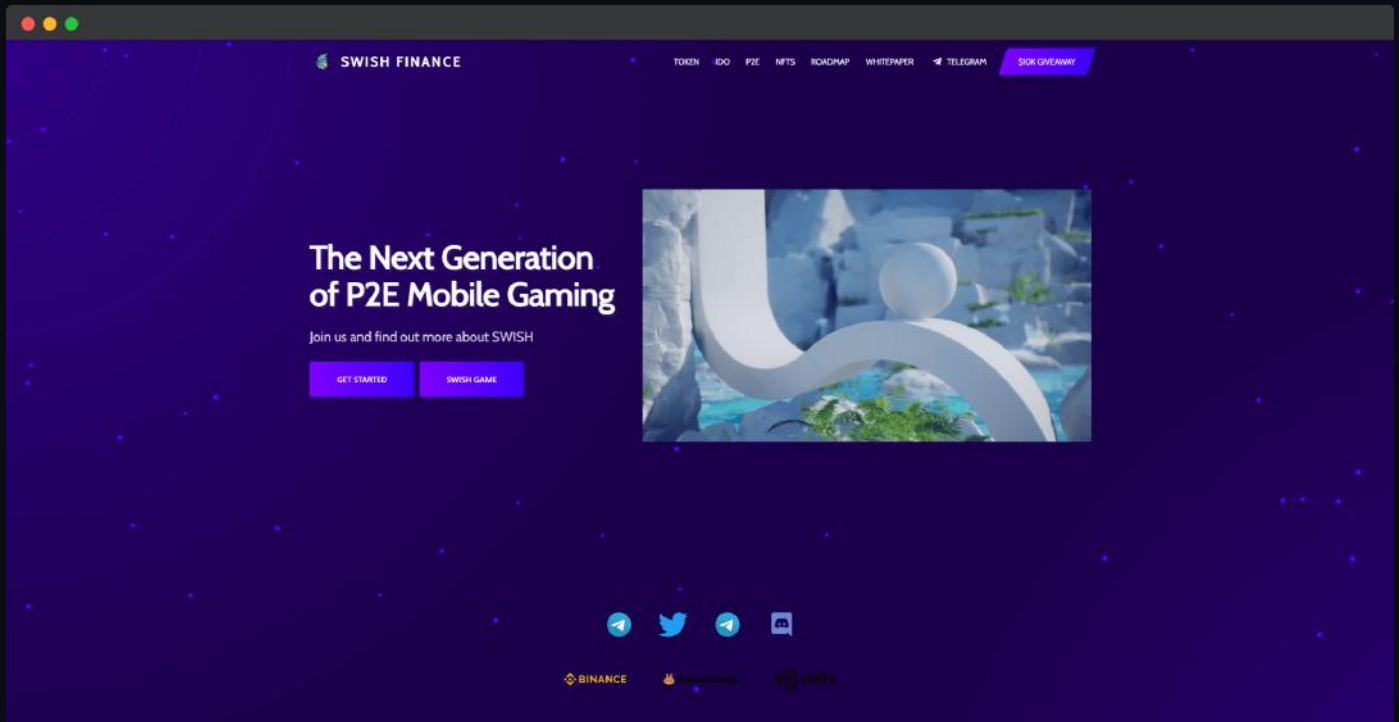
    address public devAccount1=payable(0xaE30339eF81829Ffd614aB12c4bc5774586f619c);
    address public devAccount2=payable(0x23D0715fF7E6668ef3f579eaed8076C9E03dF251);

    uint256 private tradeStartedAt;
    uint256 private _circulatingSupply =InitialSupply;
    uint256 public balanceLimit = _circulatingSupply;
    uint256 public sellLimit = _circulatingSupply;
    uint256 private maxBuyAmount = 20000000 * 10**_decimals;

    uint16 public constant maxBuyTimeLock= 9 seconds;
```

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

Certificate of Proof

● Not KYC verified by Coinsult

Swish Coin

Audited by Coinsult.net



Date: 7 October 2022

✓ Advanced Manual Smart Contract Audit

End of report
Smart Contract Audit

Request your smart contract audit / KYC

t.me/coinsult_tg