




Request your audit at [coinsult.net](https://coinsult.net)

# Advanced Manual Smart Contract Audit

October 26, 2022

 [CoinsultAudits](https://twitter.com/CoinsultAudits)

 [info@coinsult.net](mailto:info@coinsult.net)

 [coinsult.net](https://coinsult.net)

Audit requested by



**ShoshinInu**

0x296090c2544dcc429cda0d6b2869914d0152f9f3

# 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 ShoshinInu

## 7. Contract Snapshot

## 8. Website Review

## 9. Certificate of Proof

# Audit Summary

Project Name	ShoshinInu
Website	<a href="https://shoshininu.com/">https://shoshininu.com/</a>
Blockchain	Ethereum
Smart Contract Language	Solidity
Contract Address	0x296090c2544dcc429cda0d6b2869914d0152f9f3
Audit Method	Static Analysis, Manual Review
Date of Audit	26 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.

# Audit Scope

## Source Code

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

<https://etherscan.io/address/0x296090c2544dcc429cda0d6b2869914d0152f9f3#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.

## Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0x5d5f9d4ee5df08c34b6d167890061593220b8c72	100,000	100.0000%

# 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

# 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
<span style="color: #00A0C0;">●</span> Informational	0	0	0	0
<span style="color: #00C853;">●</span> Low-Risk	6	6	0	0
<span style="color: #C4C400;">●</span> Medium-Risk	1	1	0	0
<span style="color: #C0392B;">●</span> High-Risk	3	3	0	0

## Centralization Risks

Coinsult checked the following privileges:

Contract Privilege	Description
Owner can mint?	<span style="color: #00C853;">●</span> Owner cannot mint new tokens
Owner can blacklist?	<span style="color: #C0392B;">●</span> Owner can blacklist addresses
Owner can set fees > 25%?	<span style="color: #00C853;">●</span> Owner cannot set the sell fee to 25% or higher
Owner can exclude from fees?	<span style="color: #00A0C0;">●</span> Owner can exclude from fees
Owner can pause trading?	<span style="color: #00C853;">●</span> Owner cannot pause the contract
Owner can set Max TX amount?	<span style="color: #00C853;">●</span> Owner cannot set max transaction amount

More owner privileges are listed later in the report.



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 updateMarketingWallet(address newWallet) external onlyOwner {
    marketingWallet = newWallet;
}

function updateDevWallet(address newWallet) external onlyOwner {
    devWallet = newWallet;
}
```

### 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, so Bob 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 rescueETH(uint256 weiAmount) external onlyOwner {  
    payable(devWallet).transfer(weiAmount);  
}
```

## 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-104	CWE-252: Unchecked Return Value

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

## Unchecked transfer

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

```
function rescueERC20(address tokenAdd, uint256 amount) external onlyOwner {
    IERC20(tokenAdd).transfer(devWallet, amount);
}
```

## 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..

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 SetBuyTaxes(
    uint256 _marketing,
    uint256 _liquidity,
    uint256 _dev
) external onlyOwner {
    require((_marketing + _liquidity + _dev) <= 3, "Must keep fees at 3% or less");
    taxes = Taxes(_marketing, _liquidity, _dev);
}
```

## 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
SWC-135	CWE-1164: Irrelevant Code

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

## Code With No Effects

Detect the usage of redundant statements that have no effect.

```
function _msgData() internal view virtual returns (bytes calldata) {
    this; // silence state mutability warning without generating bytecode - see https://github.com/ethers-io/ethers.js/issues/1336
    return msg.data;
}
```

## Recommendation

Remove redundant statements if they congest code but offer no value.

## Exploit scenario

```
contract RedundantStatementsContract {

    constructor() public {
        uint; // Elementary Type Name
        bool; // Elementary Type Name
        RedundantStatementsContract; // Identifier
    }

    function test() public returns (uint) {
        uint; // Elementary Type Name
        assert; // Identifier
        test; // Identifier
        return 777;
    }
}
```

Each commented line references types/identifiers, but performs no action with them, so no code will be generated for such statements and they can be removed.

Error Code	Description
SLT: 076	Costly operations in a loop

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

## Costly operations inside a loop

Costly operations inside a loop might waste gas, so optimizations are justified.

```
function airdropTokens(address[] memory accounts, uint256[] memory amounts) external onlyOwner{
    require(accounts.length == amounts.length, "Arrays must have same size");
    for(uint256 i; i<&lt; accounts.length; i++){
        super._transfer(msg.sender, accounts[i], amounts[i]);
    }
}
```

## Recommendation

Use a local variable to hold the loop computation result.

## Exploit scenario

```
contract CostlyOperationsInLoop{

    function bad() external{
        for (uint i=0; i < loop_count; i++){
            state_variable++;
        }
    }

    function good() external{
        uint local_variable = state_variable;
        for (uint i=0; i < loop_count; i++){
            local_variable++;
        }
        state_variable = local_variable;
    }
}
```

Incrementing `state_variable` in a loop incurs a lot of gas because of expensive `SSTOREs`, which might lead to an out-of-gas.

Error Code	Description
CSM-01	Max Sell limit also triggered when transferring from wallet to wallet

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

### Max Sell limit also triggered when transferring from wallet to wallet

```
if (
  sender != pair &&& !exemptFee[recipient] &&& !exemptFee[sender] &&& !_liqui
) {
  require(amount <= maxSellLimit, "You are exceeding maxSellLimit");
  if (recipient != pair) {
    require(
      balanceOf(recipient) + amount = coolDownTime, "Cooldown enabled");
    _lastSell[sender] = block.timestamp;
  }
}
```

### Recommendation

Add an if statement to check for wallet to wallet transfers.

Error Code	Description
CSH-01	Owner can bulk blacklist addresses

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

### Owner can bulk blacklist addresses

```
function bulkIsBlacklisted(address[] memory accounts, bool state) external onlyOwner {
    for (uint256 i = 0; i < accounts.length; i++) {
        isBlacklisted[accounts[i]] = state;
    }
}
```

### Recommendation

Using this, the owner can blacklist a lot of addresses in 1 time, potentially all presale buyers and preventing them to trade.



Error Code	Description
CSH-02	Owner can set sell cooldown to infinity

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

### Owner can set sell cooldown to infinity

```
function updateCooldown(bool state, uint256 time) external onlyOwner {
    coolDownTime = time * 1 seconds;
    coolDownEnabled = state;
}
```

### Recommendation

Add a require statement to prevent setting the cooldown to arbitrary high amounts

Error Code	Description
CSH-03	Launchtax is set to 99

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

### Launchtax is set to 99

```
uint256 private launchtax = 99;

else if (useLaunchFee) {
    feeswap = launchtax;
    feesum = launchtax;
}
```

### Recommendation

When `block.number < genesis_block + deadline`, the contract will use `launchtax`, which is 99%. Be careful when trading

## 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 cannot set the transfer fee to 25% or higher
Buy fee	● Owner cannot set the buy fee to 25% or higher
Sell fee	● Owner cannot set the sell fee to 25% or higher

Type of fee	Description
Max transfer fee	3%
Max buy fee	3%
Max sell fee	3%

## Function

```
function SetBuyTaxes(
    uint256 _marketing,
    uint256 _liquidity,
    uint256 _dev
) external onlyOwner {
    require((_marketing + _liquidity + _dev) <= 3, "Must keep fees at 3% or less");
    taxes = Taxes(_marketing, _liquidity, _dev);
}

function SetSellTaxes(
    uint256 _marketing,
    uint256 _liquidity,
    uint256 _dev
) external onlyOwner {
    require((_marketing + _liquidity + _dev) <= 3, "Must keep fees at 3% or less");
```

## 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?	<span style="color: green;">●</span> 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?	<span style="color: green;">●</span> Owner cannot set max transaction amount

## 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?	<input checked="" type="radio"/> Owner can exclude from fees

## Function

```
function updateExemptFee(address _address, bool state) external onlyOwner {
    exemptFee[_address] = state;
}

function bulkExemptFee(address[] memory accounts, bool state) external onlyOwner {
    for (uint256 i = 0; i < accounts.length; i++) {
        exemptFee[accounts[i]] = state;
    }
}
```

## 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

## 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?	<span style="color: red;">●</span> Owner can blacklist addresses

## Function

```
function updateIsBlacklisted(address account, bool state) external onlyOwner {
    isBlacklisted[account] = state;
}

function bulkIsBlacklisted(address[] memory accounts, bool state) external onlyOwner {
    for (uint256 i = 0; i < accounts.length; i++) {
        isBlacklisted[accounts[i]] = state;
    }
}
```



## 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 use the deadline function before launch
- ⚠ Owner can update cooldown period arbitrary long
- ⚠ Owner can update max wallet holding balance
- ⚠ Owner can update max transaction amount but not less than 0.1% of the total supply

# Notes

## Notes by ShoshinInu

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.

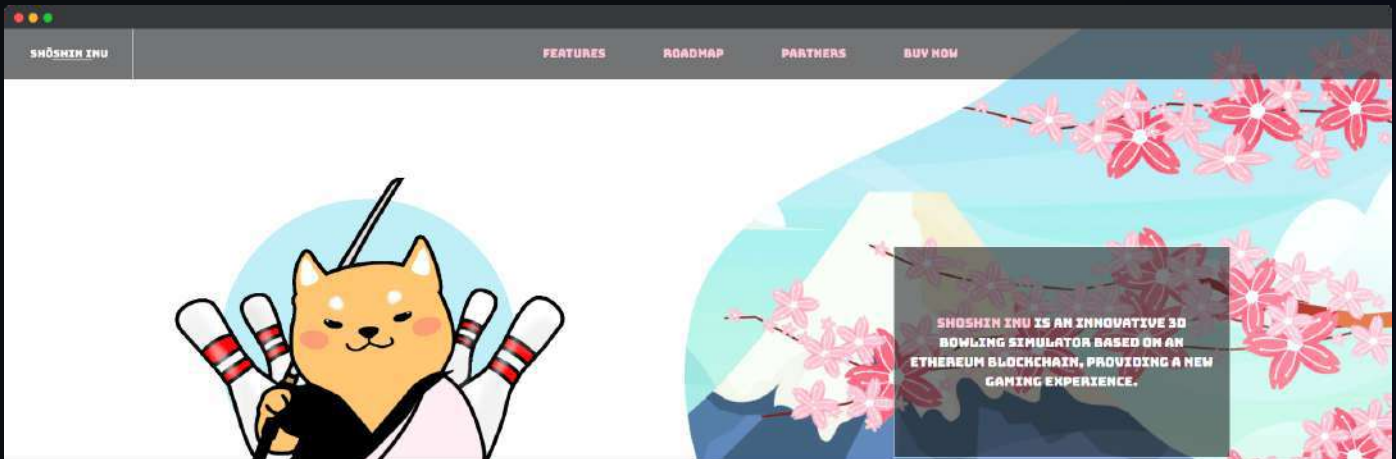
```
contract ShoshinInu is ERC20, Ownable {
    using Address for address payable;

    IRouter public router;
    address public pair;

    bool private _liquidityMutex = false;
    bool public providingLiquidity = false;
    bool public tradingEnabled = false;
```

# 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

## ShoshinInu

Audited by Coinsult.net




Date: 26 October 2022

✓ Advanced Manual Smart Contract Audit

# End of report

## **Smart Contract Audit**

 [CoinsultAudits](#)

 [info@coinsult.net](mailto:info@coinsult.net)

 [coinsult.net](https://coinsult.net)

Request your smart contract audit / KYC

**[t.me/coinsult\\_tg](https://t.me/coinsult_tg)**