Coinsult

Request your audit at coinsult.net

Advanced Manual
Smart Contract Audit

February 20, 2024

CoinsultAudits

t.me/coinsult_tg

coinsult.net

Audit requested by

WinnersCoin Presale

0x01832626D80394CBDB0C70e6278D3D586d1DB45E

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	3	3	0	0
Medium-Risk	2	2	0	0
High-Risk	0	0	0	0

Table of Contents

Audit Summary

Project Name	WinnersCoin Presale
Website	https://winnerscoin.io
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x01832626D80394CBDB0C70e6278D3D586d1DB45E
Audit Method	Static Analysis, Manual Review
Date of Audit	20 February 2024

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 WinnersCoin Presale to perform an audit based on the following code:

https://bscscan.com/address/0x01832626D80394CBDB0C70e6278D3D586d1DB45E#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	Passed
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	Passed
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

Error Code	Description
CS-01	Dead code

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

Dead code

```
if (!buyersAmount[msg.sender].exists) {
    buyers.push(msg.sender);
    buyersAmount[msg.sender].exists = true;
} // useless code

uint256 tokensforWithdraw = buyersAmount[msg.sender].amount;
buyersAmount[msg.sender].amount = 0;
// not removing from exists list

        );
        buyersAmount[buyers[i]].amount = 0;
        // not removing from exists list
    }
```

Recommendation

buyersAmount[msg.sender].exists is never used, and also not implemented correctly because you are not removing it when the user has claimed tokens.

Error Code	Description
CWE-841	Improper Enforcement of Behavioral Workflow

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

Contract does not use a ReEntrancyGuard

One of the major dangers of calling external contracts is that they can take over the control flow. In the reentrancy attack (a.k.a. recursive call attack), a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways.

```
function buyToken(
    address _token,
    uint256 _amount
) external payable saleEnabled {
    uint256 saleTokenAmt;
    if (_token != address(0)) {
        require(_amount &gt; 0);
        require(
            payableTokens[_token] == true,
            "Presale: Token not allowed"
        );

        saleTokenAmt = getTokenAmount(_token, _amount);
        require(
            (totalTokensSold + saleTokenAmt) &lt; totalTokensforSale,
            &quot;Presale: Not enough tokens to be sale&quot;
```

Recommendation

The best practices to avoid Reentrancy weaknesses are: Make sure all internal state changes are performed before the call is executed. This is known as the Checks-Effects-Interactions pattern, or use a reentrancy lock (ie. OpenZeppelin's ReentrancyGuard.

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.

```
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
CSM-01	Prices and rates are entered manually

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

Prices and rates are entered manually

```
function updateTokenRate(
    address[] memory _tokens,
    uint256[] memory _prices,
    uint256 _rate
) external onlyOwner {
    require(
        _tokens.length == _prices.length,
        "Presale: tokens &amp;amp; prices arrays length mismatch"
    );

    if (_rate != 0) {
        rate = _rate;
    }

    for (uint256 i = 0; i &amp;lt; _tokens.length; i += 1) {
        require(payableTokens[_tokens[i]] == true);
        require(_prices[i] != 0);
        tokenPrices[_tokens[i]] = _prices[i];
    }
}
```

Recommendation

The contract heavily depends on the capability of the owner to setup the right values for the presale. If the owner uses a wrong value for 1 token, the investors can get wrong output amounts.

Error Code	Description
CSM-02	Owner gets all funds

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

Owner gets all funds

```
function transferETH() private {
    payable(LPAddress).transfer(msg.value);
}

function transferToken(address _token, uint256 _amount) private {
    IERC20(_token).safeTransferFrom(msg.sender, LPAddress, _amount);
}
```

Recommendation

The owner of this contract gets all the funds, even when he has not distributed the tokens yet. So there is a high centralization risk here.

Other Owner Privileges Check

Error Code	Description
CEN-100	Centralization: Operator Priviliges

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

✅ No other important owner privileges to mention.

Notes

Notes by WinnersCoin Presale

No notes provided by the team.

Notes by Coinsult

No notes provided by Coinsult

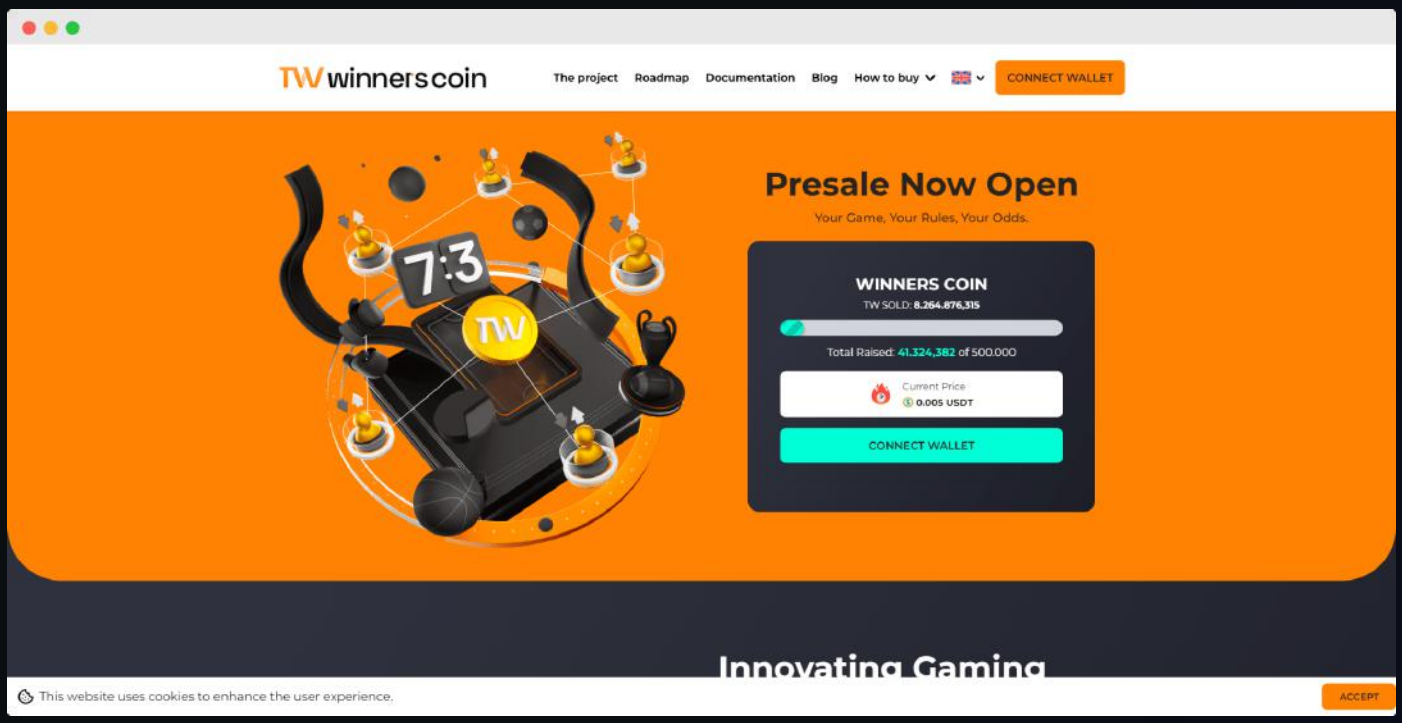
Contract Snapshot

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

```
contract WinnersCoinPresale is Ownable {
using SafeMath for uint256;
using SafeERC20 for IERC20;
using SafeERC20 for IERC20Metadata;

address public LPAddress;

uint256 public rate;

address public saleToken;

uint public saleTokenDec;

uint256 public totalTokensforSale;

mapping(address =&gt; bool) public payableTokens;
```

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

WinnersCoin Presale

Audited by Coinsult.net

Date: 20 February 2024

✔ Advanced Manual Smart Contract Audit

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

coinsult.net

End of report
Smart Contract Audit

CoinsultAudits

info@coinsult.net

coinsult.net