Coinsult

Request your audit at coinsult.net

Advanced Manual **Smart Contract Audit**

May 12, 2023

CoinsultAudits

info@coinsult.net

coinsult.net

Audit requested by

ChitCAT app

No contract address

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 ChitCAT app
7. Contract Snapshot
8. Website Review
9. Certificate of Proof

Audit Summary

Project Name	ChitCAT app
Website	https://chitcat.io/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	
Audit Method	Static Analysis, Manual Review
Date of Audit	12 May 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 ChitCAT app to perform an audit based on the following code:

Not available

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.

This app audit was based solely on Solidity code file, not on the security of the dApp.

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	4	4	0	0
● Medium-Risk	3	3	0	0
● High-Risk	0	0	0	0

Error Code	Description
CS-01	Etherscan will not display friendships and messages

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

Etherscan will not display friendships and messages

```
// returns friend array
function getFriendList() external view returns (Friend[] memory) {
    return userList[msg.sender].friendlist;
}
```

Recommendation

Etherscan does not support these datatypes, make sure to query messages via ether. Js or web3. Js

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 ^0.8.7;
```

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.

```
AllUsersStruct[] getAllUsers;

mapping(address =&gt; User) userList;
mapping(bytes32 =&gt; Message[]) allMessages;
```

Recommendation

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

Error Code	Description
SLT: 062	Comparison to boolean constant

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

Boolean equality

Detects the comparison to boolean constants.

```
require(checkUserExists(msg.sender) == false, "User already exists");

require(
      checkAlreadyFriends(msg.sender, friend_key) == false,
      "Alredy friends"
  );
```

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`.

Error Code	Description
CSM-01	Different users can create accounts with the same name

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

Different users can create accounts with the same name

```
//create Account
function createAccount(string calldata name) external {
    require(checkUserExists(msg.sender) == false, "User already exists");
    require(bytes(name).length &gt; 0, "User name cannot be empty");

    userList[msg.sender].name = name;
    getAllUsers.push(AllUsersStruct(name, msg.sender));
}
```

Recommendation

Make sure names are unique, if it is intentional that people can have the same name, then this issue can be disregarded.

Error Code	Description
CSM-02	User can add anyone to friend list, friend does not have to accept any request

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

User can add anyone to friend list, friend does not have to accept any request

```
//Add Friend
function addFriend(address friend_key, string calldata name) external {
    require(checkUserExists(friend_key), "User does not exist");
    require(checkUserExists(msg.sender), "create an account first");
    require(msg.sender != friend_key, "User cant add themselves as friend");
    require(
        checkAlreadyFriends(msg.sender, friend_key) == false,
        "Alredy friends"
    );

    _addFriend(msg.sender, friend_key, name);
    _addFriend(friend_key, msg.sender, userList[msg.sender].name);
}
```

Recommendation

Make sure the user can accept or deny the friend request, if it is intentional, then this issue can be disregarded.

Error Code	Description
CSM-03	Add friend uses own name, not name of the user itself

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

Add friend uses own name, not name of the user itself

```
// adds freind address to friendList
function _addFriend(
    address user,
    address friend,
    string memory friendName
) internal {
    Friend memory newFriend = Friend(friend, friendName);
    userList[user].friendlist.push(newFriend);
}
```

Recommendation

Remove the name parameter and use the name entered during account creation.

Simulated transaction

Test Code	Description
SIM-01	Testing a normal transfer

https://testnet.bscscan.com/address/0xda07e74ca081db2d5d4b9b0c533268535ca9fcee#readContract

Other Owner Privileges Check

Error Code	Description
CEN-100	Centralization: Operator Priviliges

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

Encryption happens server-side, and the owner has access to the encryption key. Which means he can read all conversations of every person.

Notes

Notes by ChitCAT app

No notes provided by the team.

Notes by Coinsult

This app audit was based solely on Solidity code file, not on the security of the dApp infrastructure such as connecting wallets etc.

Make sure to check very well to what you are approving when you connect your wallet to any website.

This contract enables a basic decentralized messaging system on the Binance network. It’s important to note that the messages sent through this system are public and visible to anyone on the network. This contract does not provide privacy or encryption for the messages. If privacy is a requirement, additional measures such as message encryption would need to be implemented.

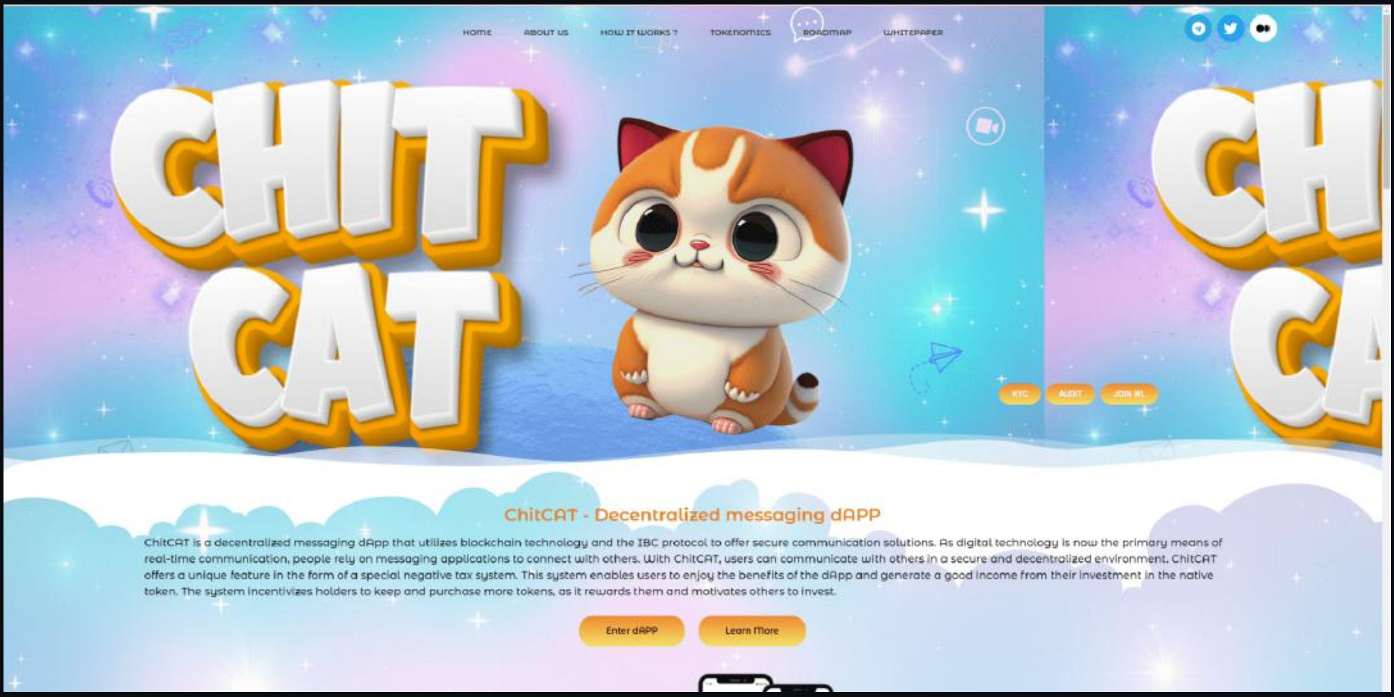
Contract Snapshot

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

```
pragma solidity ^0.8.7;

contract Chitcatapp {
    struct User {
        string name;
        Friend[] friendlist;
    }

    struct Friend {
        address pubKey;
        string name;
    }

    struct AllUsersStruct {
        string name;
        address account;
    }

    struct Message {
        address sender;
        uint256 timeStamp;
        string message;
    }

    AllUsersStruct[] getAllUsers;

    mapping(address =&gt; User) userList;
    mapping(bytes32 =&gt; Message[]) allMessages;
```

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.



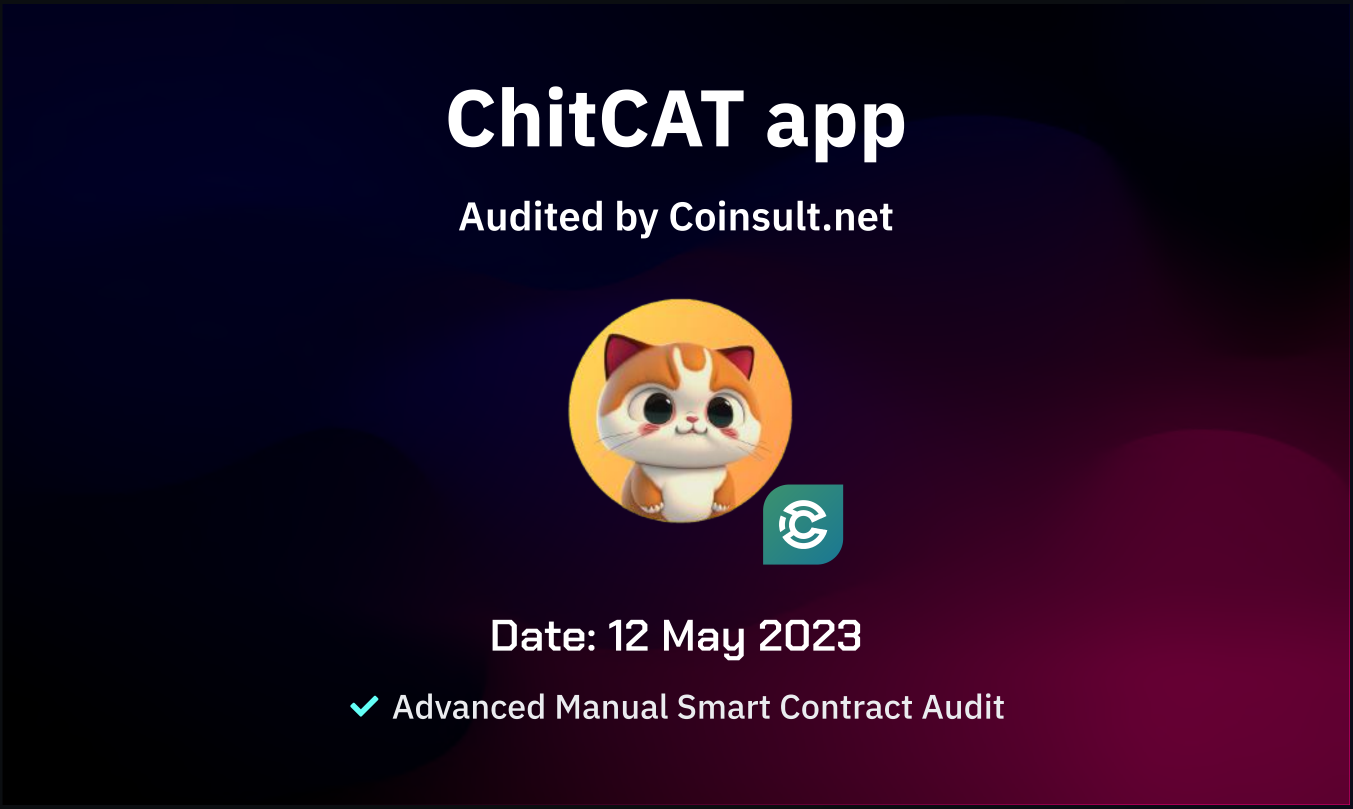
ChitCAT - Decentralized messaging dAPP

ChitCAT is a decentralized messaging dApp that utilizes blockchain technology and the IBC protocol to offer secure communication solutions. As digital technology is now the primary means of real-time communication, people rely on messaging applications to connect with others. With ChitCAT, users can communicate with others in a secure and decentralized environment. ChitCAT offers a unique feature in the form of a special negative tax system. This system enables users to enjoy the benefits of the dApp and generate a good income from their investment in the native token. The system incentivizes holders to keep and purchase more tokens, as it rewards them and motivates others to invest.

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



ChitCAT app

Audited by Coinsult.net

Date: 12 May 2023

✓ 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.