AUDIT REPORT

Jul 2024

Audit conducted by

RICARDO PONTES

Table Of Contents

Audit Scope & Methodology

The scope of this report is to audit the above smart contract source code and CryptoHub has scanned the contract and reviewed the project for common vulnerabilities, exploits, hacks, and back-doors. Below is the list of commonly known smart contract vulnerabilities, exploits, and hacks:

Smart Contract Vulnerabilities

- □ Re-entrancy
- □ Unhandled Exceptions
- □ Transaction Order Dependency
- □ Integer Overflow
- □ Unrestricted Action
- □ Incorrect Inheritance Order
- □ Typographical Errors
- □ Requirement Violation

Source Code Review

- □ Ownership Takeover
- □ Gas Limit and Loops
- □ Deployment Consistency
- □ Repository Consistency
- □ Data Consistency
- □ Token Supply Manipulation

Functional Assessment

- □ Access Control and Authorization
- □ Operations Trail and Event Generation
- □ Assets Manipulation
- □ Liquidity Access

CryptoHub Audit Standard

The aim of CryptoHub standard is to analyze the smart contract and identify the vulnerabilities and the hacks in the smart contract. Mentioned are the steps used by CryptoHub to assess the smart contract:

1. Solidity smart contract source code reviewal:

- Review of the specifications, sources, and instructions provided to CryptoHub to make sure we understand the size, scope, and functionality of the smart contract.
- Manual review of code, which is the process of reading source code line-by-line to identify potential vulnerabilities.

2. Static, Manual, and Software analysis:

- Test coverage analysis, which is the process of determining whether the test cases are covering the code and how much code is exercised when we run those test cases.
- Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.

3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts

Automated 3P frameworks used to assess the smart contract vulnerabilities

- Slither
- Consensys MythX
- Consensus Surya
- Open Zeppelin Code Analyzer
- Solidity Code Complier

Smart Contract Risk Assessment

SWC Attacks

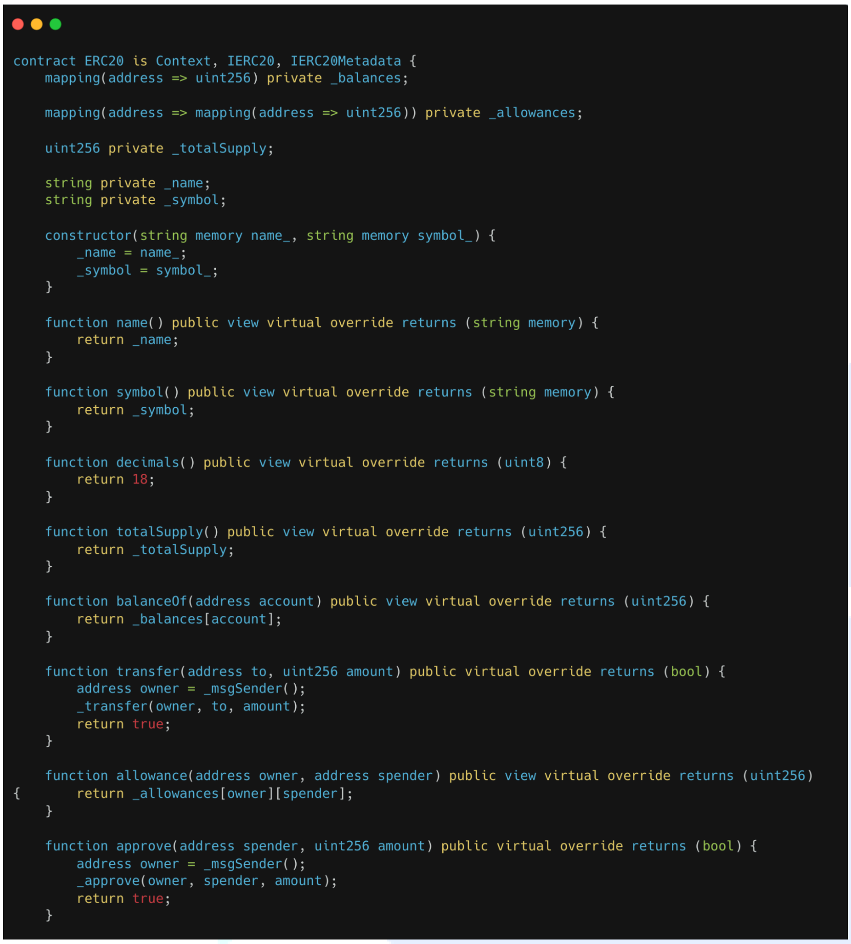
The following table contains an overview of the SWC registry. Each row consists of an SWC identifier (ID), weakness title, CWE parent and list of related code samples.

The auditor used a MythX tool, A static analyzer that parses the Soldity AST, a symbolic analyzer that detects possible vulnerable states, and a greybox fuzzer that detects vulnerable execution paths.

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	✓ **Passed**
SWC - 108	State Variable Default Visibility	✓ **Passed**
SWC - 109	Uninitialized Storage Pointer	✓ **Passed**
SWC - 110	Assert Violation Passed	✓ **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**

ID	Description	Status
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	✓ Passed
SWC - 124	Write to Arbitrary Storage Location	✓ Passed
SWC - 125	Incorrect Inheritance Order Passed	✓ 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 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

Our Findings:



```
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

    function name() public view virtual override returns (string memory) {
        return _name;
    }

    function symbol() public view virtual override returns (string memory) {
        return _symbol;
    }

    function decimals() public view virtual override returns (uint8) {
        return 18;
    }

    function totalSupply() public view virtual override returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view virtual override returns (uint256) {
        return _balances[account];
    }

    function transfer(address to, uint256 amount) public virtual override returns (bool) {
        address owner = _msgSender();
        _transfer(owner, to, amount);
        return true;
    }

    function allowance(address owner, address spender) public view virtual override returns (uint256)
{       return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, amount);
        return true;
    }
```

Low Severity Issues:

1 | Redundant checks

Type: gas/code quality
Level: Low
File: `contracts/libraries/LibEnsureSafeTransfer.sol`
Functions: `safeTransferFromEnsureExactAmount`, `transferEnsureExactAmount`, `safeTransferFrom`, `safeTransfer`

Description: all these functions use a check for address zero on the token,recipient and sender address which is redundant as the `SafeERC20` library and the `ERC20` already make both those checks.

Recommendation: Remove those checks.

Legal Advisory

Important Disclaimer

CryptoHub provides contract auditing and project verification services for blockchain projects. The purpose of the audit is to analyze the on-chain smart contract source code, and to provide a basic overview of the project. This report should not be transmitted, disclosed, referred to, or relied upon by any person for any purposes without CryptoHub prior written consent.

CryptoHub provides the easy-to-understand assessment of the project, and the smart contract (otherwise known as the source code). The audit makes no statements or warranties on the security of the code. It also cannot be considered as an adequate assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have used all the data at our disposal to provide the transparent analysis, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Be aware that smart contracts deployed on a blockchain aren't resistant from external vulnerability, or a hack. Be aware that active smart contract owner privileges constitute an elevated impact to smart contract's safety and security. Therefore, CryptoHub does not guarantee the explicit security of the audited smart contract.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

This report should not be considered as an endorsement or disapproval of any project or team. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. Do conduct your own due diligence and consult your financial advisor before making any investment decisions.

About CryptoHub

CryptoHub provides intelligent blockchain solutions. CryptoHub is developing an ecosystem that is seamless and responsive. Some of our services: Blockchain Security, Token Launchpad, NFT Marketplace, etc. CryptoHub's mission is to interconnect multiple services like Blockchain Security, DeFi, Gaming, and Marketplace under one ecosystem that is seamless, multi-chain compatible, scalable, secure, fast, responsive, and easy-to-use.

CryptoHub is built by a decentralized team of UI experts, contributors, engineers, and enthusiasts from all over the world. Our team currently consists of 3+ core team members, and 6+ casual contributors. CryptoHub provides manual, static, and automatic smart contract analysis, to ensure that the project is checked against known attacks and potential vulnerabilities.