ENTERSOFT

Hedera

Smart Contract Audit

DATE: 8th January 2025 | PREPARED BY: Entersoft Australia

Contents

Revision History & Version Control

Start Date	End Date	Author	Comments/Details
02 Jan 2025	07 Jan 2025	Gurkirat	Interim Release for the Client

Reviewed by	Released by
Nishita Palaksha	Nishita Palaksha

Entersoft was commissioned to perform a source code review on Hedera smart contracts. The review was conducted between January 3, 2025, to January 7, 2025. The report is organized into the following sections.

- Executive Summary: A high-level overview of the security audit findings.
- Technical analysis: Our detailed analysis of the Smart Contract code

The information in this report should be used to understand overall code quality, security, correctness, and meaning that code will work as described in the smart contract.

1.0 Disclaimer

This is a limited audit report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to: (i) smart contract best coding practices and vulnerabilities in the framework and algorithms based on white paper, code, the details of which are set out in this report, (Smart Contract audit). To get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us based on what it says or does not say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Entersoft Australia and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Entersoft) owe no duty of care towards you or any other person, nor does Entersoft make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Entersoft hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Entersoft hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Entersoft, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the Smart contract is purely based on the smart contract code shared with us alone.

2.0 Overview

2.1 Project Overview

During the period of **02 Jan 2025 to 07 Jan 2025**, Entersoft performed smart contract security audits for **Hedera**.

2.2 Scope

The scope of this audit was to analyze and document the smart contract codebase for quality, security, and correctness.

File In Scope:

- AccessControl.sol
- Swap.sol
- MyToken.sol
- IStablecoinStudio.sol

Out of Scope: External contracts, External Oracles, other smart contracts in the repository, or imported smart contracts.

2.3 Project Summary

Project Name	No. of Smart Contract File(s)	Verified	Vulnerabilities
Hedera	4	Yes	As per report. Section 2.6

2.4 Audit Summary

Delivery Date	Method of Audit	Consultants Engaged
08 Jan 2025	Manual and Automated approach	1

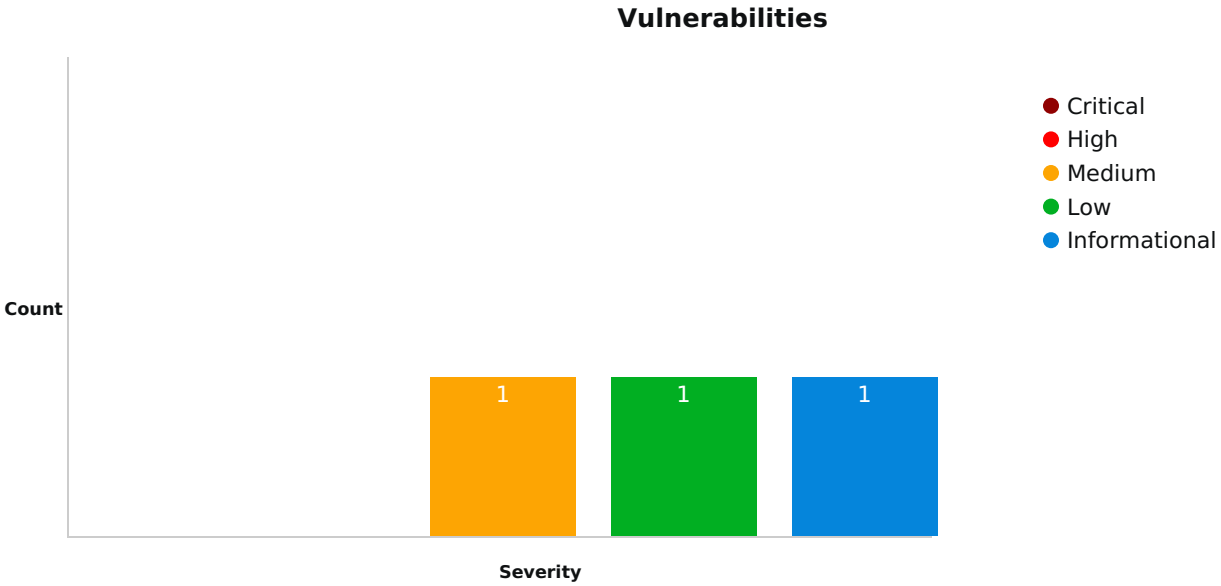
2.5 Security Level References

Every vulnerability in this report was assigned a severity level from the following classification table:

		Impact				
		Minimal	Low	Medium	High	Critical
Likelihood	Critical	Minimal	Low	Medium	High	Critical
	High	Minimal	Low	Medium	High	Critical
	Medium	Minimal	Low	Medium	Medium	High
	Low	Minimal	Low	Low	Low	Medium
	Minimal	Minimal	Minimal	Minimal	Low	

2.6 Vulnerability Summary

Critical	High	Medium	Low	Informational
0	0	1	1	1



3.0 Executive Summary

Entersoft has conducted a comprehensive technical audit of the Hedera protocol through a comprehensive smart contract audit approach. The primary objective was to identify potential vulnerabilities and security risks within the codebase, ensuring adherence to industry-leading standards while prioritizing security, reliability, and performance. Our focus was on prompt and efficient identification and resolution of vulnerabilities to enhance the overall robustness of the solidity smart contract.

Importantly, our audit process intentionally avoided reliance solely on automated tools, emphasizing a more in-depth and nuanced approach to security analysis. Conducted from January 3, 2025, to January 7, 2025, our team diligently assessed and validated the security posture of the solidity smart contract, ultimately finding a number of vulnerabilities as per vulnerability summary table.

Testing Methodology:

We have leveraged static analysis techniques extensively to identify potential vulnerabilities automatically with the aid of cutting-edge tools such as Slither and Aderyn. Apart from this, we carried out extensive manual testing to iron out vulnerabilities that could slip through an automated check. This included a variety of attack vectors like reentrancy attacks, overflow and underflow attacks, timestamp dependency attacks, and more.

While going through the due course of this audit, we also ensured to cover edge cases, and built a combination of scenarios to assess the contracts' resilience. Our attempt to leave no stone unturned involved coming up with both negative and positive test cases for the system, and grace handling of stressed scenarios.

Our testing methodology in Solidity adhered to industry standards and best practices, integrating partially implemented OWASP and NIST SP 800 standards for encryption and signatures. Solidity's renowned security practices were complemented by tools such as Solhint for linting, and the Solidity compiler for code optimization. Sol-profiler, Sol-coverage, and Sol-sec were employed to ensure code readability and eliminate unnecessary dependencies.

Tools Used for Audit:

In the course of our audit, we leveraged a suite of tools to bolster the security and performance of our program. While our team drew on their expertise and industry best practices, we also integrated various tools into our development environment. Noteworthy among them are Slither and Aderyn. This holistic approach ensures a thorough analysis, uncovering potential issues that automated tools alone might overlook. Entersoft takes pride in utilizing these tools, which significantly contribute to the quality, security, and maintainability of our codebase.

Code Review / Manual Analysis:

Our team conducted a manual analysis of the Solidity and solana smart contracts to identify new vulnerabilities or to verify vulnerabilities found during static and manual analysis. We carefully analyzed every line of code and made sure that all instructions provided during the onboarding phase were followed. Through our manual analysis, we were able to identify potential vulnerabilities that may have been missed by automated tools and ensure that the smart contract was secure and reliable.

Auditing Approach and Methodologies Applied:

The solidity smart contract was audited in a comprehensive approach to ensure the highest level of security and reliability. Careful attention was given to the following key areas to ensure the overall quality of code:

- Code quality and structure: We conducted a detailed review of the codebase to identify any potential issues related to code structure, readability, and maintainability. This included analyzing the overall architecture of the solidity smart contract and reviewing the code to ensure it follows best practices and coding standards.

- Security vulnerabilities: Our team used manual techniques to identify any potential security vulnerabilities that could be exploited by attackers. This involved a thorough analysis of the code to identify any potential weaknesses, such as buffer overflows, injection vulnerabilities, Signatures, and deprecated functions.

Documentation Overview:
Documentation is crucial for any project as it ensures clarity and understanding, helping developers and users grasp how the code and protocol function. It aids in new contributors, auditors facilitating maintenance and troubleshooting by providing insights into design decisions.
For blockchain protocols, thorough documentation is particularly important due to the complexity and need for security and transparency.
Below is the required documentation, rated according to the quality as provided by the protocol.

Protocol Documentation:

- Purpose: To describe the overall protocol, its goals, architecture, and how it operates.

- Quality Provided: N/A

Technical Documentation:

- Purpose: To provide detailed information about the technical aspects of the protocol, including smart contracts, APIs, and their interactions.

- Quality Provided: N/A

Inline Comments:

- Purpose: To explain specific parts of the code, such as the purpose of functions, variables, and logic.

- Quality Provided: N/A

3.1 Findings

Vulnerability ID	Contract Name	Severity	Status
1	AccessControl, Swap	Medium	Pending
2	AccessControl, Swap	Low	Pending
3	AccessControl, Swap, MyToken	Informational	Pending

3.2 Recommendations

Overall, the smart contracts were well-written, adhering to industry best practices and security standards. However, the audit of Hedera identified a few vulnerabilities. The process involved a thorough validation of each function, testing of edge case scenarios and execution of test scripts to ensure every functionality was meticulously evaluated.

4.0 Technical Analysis

4.1 Zero address check missing

Severity	Status	Type of Analysis
Medium	Identified	Dynamic

Contract Name:

AccessControl, Swap

Description:

The AccessControl and Swap contract constructors do not validate whether the _contractAddress, _audd_address, and _audr_address parameters are non-zero addresses. Zero addresses are often used as placeholders or default values, and failing to properly validate them can result in unintended behavior.

Locations:

AccessControl - 48
Swap - 28,29

Remediation:

Add checks in the constructor to ensure zero address is not passed in the constructor. Implement the following checks:

```
constructor(address _contractAddress, Policy memory _masterPolicy) {
require(_contractAddress!=address(0),"Address zero not allowed");
 masterPolicy = _masterPolicy; contractAddress = _contractAddress;
require(_masterPolicy.signatories.length >= 1, "must be at least one signatory");
require(_masterPolicy.minSignatories >= 1, "minimum signatories must be at least 1");
require(areAddressesUnique(_masterPolicy.signatories), "signatories must be unique");
}

constructor(address _audd_address, address _audr_address) { require(_audd_address!=0,"Address zero not allowed");
require(_audr_address!=0,"Address zero not allowed");
audd_address = _audd_address;
audr_address = _audr_address;
}
```

Impact:

Deploying the contract with zero addresses for critical roles could lead to misconfiguration,rendering the contract unusable or

insecure.

Code Snippet:

```
    AccessControl

constructor(address _contractAddress, Policy memory _masterPolicy) {

masterPolicy = _masterPolicy;

contractAddress = _contractAddress;

require(_masterPolicy.signatories.length >= 1, "must be at least one signatory");

require(_masterPolicy.minSignatories >= 1, "minimum signatories must be at least 1");

require(areAddressesUnique(_masterPolicy.signatories), "signatories must be unique");

}


Swap


constructor(address _audd_address, address _audr_address) {

audd_address = _audd_address;

audr_address = _audr_address;

}
```

Reference:

Proof of Vulnerability:

N.A.

4.2 State variables that could be declared immutable

Severity	Status	Type of Analysis
Low	Identified	Dynamic

Contract Name:

AccessControl, Swap

Description:

State variables that are not updated following deployment should be declared immutable to save gas.

Locations:

AccessControl.sol - 41
Swap.sol - 24,25

Remediation:

Add the immutable attribute to state variables that never change or are set only in the constructor.

Add:

AccessControl

```
address public immutable contractAddress;
```

Swap

```
address immutable audd_address;
address immutable audr_address;
```

Impact:

While this issue doesn't directly impact the functionality of the contract, the contract can benefit from the use of constant and immutable keywords for variables that do not change after deployment. This can save gas costs by storing the variables directly in the bytecode.

Code Snippet:

```
    AccessControl

address public contractAddress;


constructor(address _contractAddress, Policy memory _masterPolicy) {

masterPolicy = _masterPolicy;

contractAddress = _contractAddress;

require(_masterPolicy.signatories.length >= 1, "must be at least one signatory");

require(_masterPolicy.minSignatories >= 1, "minimum signatories must be at least 1");

require(areAddressesUnique(_masterPolicy.signatories), "signatories must be unique");

}


Swap


address audd_address;

address audr_address;


constructor(address _audd_address, address _audr_address) {

audd_address = _audd_address;

audr_address = _audr_address;

}
```

Reference:

Proof of Vulnerability:

N.A.

4.3 Solidity Pragma Should Be Specific, Not Wide

Severity	Status	Type of Analysis
Informational	Identified	Static

Contract Name:

AccessControl, Swap, MyToken

Description:

In the smart contract, the pragma directive pragma solidity `^0.8.27;` is used to specify the compiler version. This directive allows any compiler version greater than or equal to `0.8.27;`.

Locations:

AccessControl - 16
Swap - 16
MyToken - 3

Remediation:

Update the pragma statements in the contracts to specify a particular version of Solidity. For example, replace pragma solidity `^0.8.27;` with pragma solidity `0.8.27;`.

Impact:

This directive allows any compiler version greater than or equal to`0.8.27;.` Failure to specify a specific version may lead to compatibility issues or unexpected behaviour in future compiler versions. It's important to follow best practices to ensure the stability and security of the contracts.

Code Snippet:

```
pragma solidity ^0.8.27;
```

Reference:

Proof of Vulnerability:

N.A.

5.0 Auditing Approach and Methodologies applied

Throughout the audit of the smart contract, care was taken to ensure:

- Overall quality of code
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Mathematical calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of token standards.
- Efficient use of gas.
- Code is safe from Re-entrancy and other vulnerabilities.

A combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

5.1 Structural Analysis

In this step we have analysed the design patterns and structure of all smart contracts. A thorough check was completed to ensure all Smart contracts are structured in a way that will not result in future problems.

5.2 Static Analysis

Static Analysis of smart contracts was undertaken to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

5.3 Code Review / Manual Analysis

Manual Analysis or review of done to identify new vulnerabilities or to verify the vulnerabilities found during the Static Analysis. The contracts were completely manually analysed, and their logic was checked and compared with the one described in the whitepaper. It should also be noted that the results of the automated analysis were verified manually.

5.4 Gas Consumption

In this step, we checked the behaviour of all smart contracts in production. Checks were completed to understand how much gas gets consumed, along with the possibilities of optimisation of code to reduce gas consumption.

5.5 Tools & Platforms Used For Audit

Slither,Aderyn

5.6 Checked Vulnerabilities

We have scanned Hedera smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC-20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

6.0 Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of Hedera and methods for exploiting them. Entersoft recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while Entersoft considers the major security vulnerabilities of the analyzed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the Smart Contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Entersoft makes no undertaking to supplement or update this report based on changed circumstances or facts of which Entersoft becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that Entersoft use certain software or hardware products manufactured or maintained by other vendors. Entersoft bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, Entersoft does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by Entersoft for the exclusive benefit of Hedera and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between Entersoft and Hedera governs the disclosure of this report to all other parties including product vendors and suppliers.