CertificateRegistry.sol – Smart Contract Audit Report

Audited by: Nandini Pandey

Date: 09/06/2025

Contract: CertificateRegistry.sol

Tools used: Slither v0.11.3

Source code: CertificateRegistry.sol

1. Introduction

This report presents the findings from a security audit of the CertificateRegistry smart contract. The contract was analyzed using the Slither static analysis tool to identify potential vulnerabilities and security issues. The goal was to highlight weaknesses and provide recommendations for improving the contract's security.

2. Summary of Findings

Slither detected several issues in the contract, including dangerous external calls, missing input validation, reentrancy risks, and the use of a broad Solidity version range. The table below summarizes the main findings:

Sl.	Issue Type	Location / Function	Description	Recommendation
1	Dangerous external call	issueCertificate (line 23)	Sends ETH (even 0) to any address via student.call, which can be exploited or cause unexpected behavior.	Remove or restrict external calls.
2	Missing zero address check	issueCertificate (line 23)	No check to ensure student is not the zero address before calling.	Add require(student != address(0)).
3	Missing zero address check	changeCertifier (line 33)	No check to ensure newCertifier is not the zero address.	Add require(newCertifier != address(0)).
4	Reentrancy risk	issueCertificate	External call is made before emitting the event, which could allow reentrancy	Use Checks-Effects-Interactions pattern.

			attacks.	
5	Low-level call	issueCertificate (line 23)	Use of .call is discouraged and can introduce vulnerabilities.	Avoid low-level calls unless absolutely necessary.

3. Detailed Findings

3.1 Dangerous External Call

Location: issueCertificate function, line 23
Details: The contract uses student.call{value: 0}("") to send a zero-value transaction to an arbitrary address. Even though no ETH is sent, this can trigger fallback functions or cause unexpected behavior.
Recommendation: Remove unnecessary external calls or replace with safer alternatives.

3.2 Missing Zero Address Checks

Location: issueCertificate function, line 23
changeCertifier function, line 33
Details: The contract does not check if the student or newCertifier addresses are the zero address, which could lead to unintended consequences.
Recommendation: Add require(student != address(0)) and require(newCertifier != address(0)) to validate input.

3.3 Reentrancy Risk

Location: issueCertificate function
Details: The external call to student is made before emitting the event. If the recipient is a contract, it could re-enter the function and cause issues.
Recommendation: Follow the Checks-Effects-Interactions pattern: update state and emit events before making external calls.

3.4 Use of Low-level Call

Location: issueCertificate function, line 23
Details: The contract uses the low-level .call function, which is generally discouraged due to its complexity and risk.
Recommendation: Avoid low-level calls unless absolutely necessary.

4. Conclusion

The CertificateRegistry contract contains several vulnerabilities that could be exploited or cause unintended behavior. The most critical issues are the lack of access control, unsafe external calls, and missing input validation. It is strongly recommended to address these findings before deploying the contract to a production environment.

5. References

- Slither Detector Documentation
- Solidity Security Best Practices
- 101 Blockchains: Top Smart Contract Auditing Tools

6. Bonus

I used Scribble to annotate the contract with formal properties, such as ensuring only the certifier can issue or revoke certificates and that the certifier address is never zero. I successfully instrumented the contract using Scribble, generating an instrumented Solidity file. I attempted to analyze the instrumented contract with Mythril and MythX, but encountered technical limitations: Mythril has Windows build issues, and MythX has been sunset as a service. On a Linux system or with access to Diligence Fuzzing or similar tools, the instrumented contract could be analyzed for violations of the specified properties. This process demonstrates the application of formal verification principles as recommended by CertiK's methodology.