Giving Thanks First Flight Audit Report

Lead Auditors:

- Aditya Mishra

Table of Contents

About the Project

GivingThanks is a decentralized platform that embodies the spirit of Thanksgiving by enabling donors to contribute Ether to registered and verified charitable causes. Charities can register themselves, and upon verification by the trusted admin, they become eligible to receive donations from generous participants. When donors make a donation, they receive a unique NFT as a donation receipt, commemorating their contribution. The NFT's metadata includes the donor's address, the date of the donation, and the amount donated.

Actors

- Admin (Trusted) - Can verify registered charities.
- Charities - Can register to receive donations once verified.
- Donors - Can donate Ether to verified charities and receive a donation receipt NFT.

Disclaimer

The Aditya Mishra team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

```
All Contracts in `src` are in scope.
```

```
├── src
│   ├── CharityRegistry.sol
│   └── GivingThanks.sol
```

Compatibilities

Compatibilities: Blockchains: - Ethereum/Any EVM Tokens: - ETH - ERC721

Roles

Executive Summary

Issues found

Severity	Number of issues found
High	7
Medium	7
Low	2
Info	3
Total	19

High

[H-1] Logic Error in Verification Check found in `CharityRegistry.sol`

Description: The `CharityRegistry::isVerified()` function returns the registration status instead of the verification status of a charity. This is a critical logical error that undermines the entire verification system.

```
    function isVerified(address charity) public view returns (bool) {
        return registeredCharities[charity];  // Should be
verifiedCharities[charity]
    }
```

Impact:

- Any registered charity appears as verified without admin verification
- Bypasses the entire verification process
- Defeats the purpose of having a two-step verification system
- Could lead to users trusting unverified charities

Proof of Concept:

```
function testLogicErrorInIsVerified() public {
    vm.prank(user1);
    registry.registerCharity(maliciousCharity);
```

```
    // A charity that was never verified returns true
    assertTrue(registry.isVerified(maliciousCharity));
}
```

Recommended Mitigation: Modify the isVerified function to check the verifiedCharities mapping instead of registeredCharities.

Corrected Code:

```
function isVerified(address charity) public view returns (bool) {
    return verifiedCharities[charity];
}
```

This change ensures that the function accurately reflects the verification status of a charity.

[H-2] No Access Control on Registration in `CharityRegistry::registerCharity()` function

Description: The `CharityRegistry::registerCharity()` function lacks access controls, allowing any address to register any other address as a charity. This is a critical security flaw that could lead to widespread abuse.

```
function registerCharity(address charity) public {
    registeredCharities[charity] = true;
}
```

Impact:

- Malicious actors can register unlimited fake charities
- No validation of charity legitimacy
- Could lead to phishing attacks through fake charity registrations
- Undermines the credibility of the entire registry

Proof of Concept:

```
function testUnauthorizedRegistration() public {
    address malicious1 = address(0x1);
    address malicious2 = address(0x2);

    vm.prank(malicious1);
    registry.registerCharity(malicious2);

    assertTrue(registry.registeredCharities(malicious2));
}
```

Recommended Mitigation: Add appropriate access controls or requirements for registration

```
modifier onlyAdmin() {
    require(msg.sender == admin, "Only admin can perform this action");
    _;
}

function registerCharity(address charity) public onlyAdmin {
    require(charity != address(0), "Invalid charity address");
    registeredCharities[charity] = true;
    emit CharityRegistered(charity);
}
```

[H-3] Critical Initializaton Vulnerability found in the `constructor`

Description: The `GivingThanks::constructor()` incorrectly uses `msg.sender` for the registry address instead of the provided `_registry` parameter. This means the registry address will be set to the deployer's address rather than the intended `CharityRegistry` contract.

```
constructor(address _registry) ERC721("DonationReceipt", "DRC") {
    registry = CharityRegistry(msg.sender);  // Uses msg.sender instead of
_registry
    owner = msg.sender;
    tokenCounter = 0;
}
```

Impact:

- Contract will not be properly initialized
- All donation attempts will fail
- System completely unusable

Proof of Concept:

```
function testIncorrectInitialization() public {
    address registryAddr = address(new CharityRegistry());
    vm.prank(user1);
    GivingThanks givingThanks = new GivingThanks(registryAddr);

    // Registry address is set to user1 instead of registryAddr
    assertEq(address(givingThanks.registry()), user1);
}
```

Recommended Mitigation: Check whether the `_registry` address is not same as the address of the user.

```
constructor(address _registry) ERC721("DonationReceipt", "DRC") {
    require(_registry != address(0), "Invalid registry address");
    registry = CharityRegistry(_registry);
    owner = msg.sender;
    tokenCounter = 0;
}
```

[H-4] Unprotected Regiistry Update in `GivingThanks.sol`

Description: The `GivingThanks::updateRegistry` function has no access control, allowing anyone to change the registry address to any value.

```
function updateRegistry(address _registry) public {
    registry = CharityRegistry(_registry);
}
```

Impact:

- Complete system compromise possible
- Attacker can redirect donations to unverified addresses
- Loss of funds for donors

Proof of Concept:

```
function testRegistryHijack() public {
    address attacker = address(0x1);
    address maliciousRegistry = address(new FakeRegistry());

    vm.prank(attacker);
    givingThanks.updateRegistry(maliciousRegistry);

    // Registry successfully changed to attacker's contract
    assertEq(address(givingThanks.registry()), maliciousRegistry);
}
```

Recommended Mitigation:

```
modifier onlyOwner() {
    require(msg.sender == owner, "Not owner");
    _;
}

function updateRegistry(address _registry) public onlyOwner {
    require(_registry != address(0), "Invalid registry address");
    registry = CharityRegistry(_registry);
```

```
        emit RegistryUpdated(_registry);
    }
```

[H-5] Reentrancy Vulnerability found in `GivingThanks::donate()`

Description: The `GivingThanks::donate` function is vulnerable to reentrancy attacks due to state changes after external calls.

```
function donate(address charity) public payable {
    require(registry.isVerified(charity), "Charity not verified");
    (bool sent,) = charity.call{value: msg.value}("");
    require(sent, "Failed to send Ether");

    _mint(msg.sender, tokenCounter);
    // ... rest of the function
}
```

Impact:

- Multiple NFTs could be minted for a single donation
- Token counter could be manipulated
- Potential drainage of contract funds

Proof of Concept:

```
contract AttackingCharity {
    GivingThanks public target;

    receive() external payable {
        if (address(target).balance >= 1 ether) {
            target.donate{value: 1 ether}(address(this));
        }
    }
}
```

Recommended Mitigation: The `donate()` function should be like this:

```
function donate(address charity) public payable {
    require(registry.isVerified(charity), "Charity not verified");

    uint256 currentToken = tokenCounter++;
    _mint(msg.sender, currentToken);

    string memory uri = _createTokenURI(msg.sender, block.timestamp, msg.value);
    _setTokenURI(currentToken, uri);

    (bool sent,) = charity.call{value: msg.value}("");
```

```
    require(sent, "Failed to send Ether");
}
```

[H-6] Missing Input Validation for Donation Amount in `GivingThanks.sol`

Description: The `GivingThanks::donate` function accepts any value for donation without minimum or maximum limits.

```
function donate(address charity) public payable {
    require(registry.isVerified(charity), "Charity not verified");

    uint256 currentToken = tokenCounter++;
    _mint(msg.sender, currentToken);

    string memory uri = _createTokenURI(msg.sender, block.timestamp, msg.value);
    _setTokenURI(currentToken, uri);

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

Impact:

- Possible dust attacks with minimal donations
- Network spam through many small transactions
- Storage bloat from unnecessary NFT mints

Recommended Mitigation:

```
uint256 public constant MIN_DONATION = 0.01 ether;
uint256 public constant MAX_DONATION = 100 ether;

function donate(address charity) public payable {
    require(msg.value >= MIN_DONATION, "Donation too small");
    require(msg.value <= MAX_DONATION, "Donation too large");
    // ... rest of the function
}
```

[H-7] Centralization Risk in Registry Verification

Description: Both contracts `CharityRegistry` and `GivingThanks` rely on a single admin for charity verification, creating a central point of failure.

Impact:

- System vulnerability to admin key compromise
- Single point of failure for verification

- Potential system deadlock if admin is unavailable

Recommended Mitigation:

```
contract MultiSigCharityRegistry {
    uint256 public constant MIN_SIGNATURES = 2;
    mapping(address => bool) public isAdmin;
    mapping(address => mapping(bytes32 => bool)) public hasSignedVerification;

    function verifyCharity(address charity) public {
        bytes32 verificationHash = keccak256(abi.encodePacked(charity));
        require(isAdmin[msg.sender], "Not admin");
        hasSignedVerification[msg.sender][verificationHash] = true;

        uint256 signatures = 0;
        for (uint i = 0; i < admins.length; i++) {
            if (hasSignedVerification[admins[i]][verificationHash]) {
                signatures++;
            }
        }

        if (signatures >= MIN_SIGNATURES) {
            verifiedCharities[charity] = true;
            emit CharityVerified(charity);
        }
    }
}
```

Medium

[M-1] Missing Zero Address Validation in `CharityRegistry.sol`

Description: The contract lacks zero address validation in critical functions such as `CharityRegistry::changeAdmin()` and `CharityRegistry::registerCharity()` where addresses are used as parameters. This could lead to irrecoverable states if zero addresses are accidentally input.

```
function changeAdmin(address newAdmin) public
function registerCharity(address charity) public
```

Impact:

- Admin role could be permanently lost if set to zero address
- Charities could be registered with zero address
- No way to recover from these states due to lack of validation

Proof of Concept:

```
function testZeroAddressVulnerability() public {
    vm.prank(admin);
    registry.changeAdmin(address(0));

    // Contract now has zero address admin
    assertEq(registry.admin(), address(0));
}
```

Recommended Mitigation: Add zero address validation checks such as given below:

```
modifier nonZeroAddress(address _address) {
    require(_address != address(0), "Zero address not allowed");
    _;
}

function changeAdmin(address newAdmin) public onlyAdmin nonZeroAddress(newAdmin) {
    admin = newAdmin;
    emit AdminChanged(newAdmin);
}
```

[M-2] Lack of Event Emissions in `CharityRegistry.sol`

Description: The contract does not emit events for important state changes, making it difficult to track changes off-chain and limiting transparency.

Impact:

- Difficult to track contract state changes
- Limited transparency for users and front-end applications
- Challenging to build meaningful monitoring systems
- Poor auditability of contract usage

Recommended Mitigation: Add events for registration, verification, and admin changes such as given below:

```
// Define events
event CharityRegistered(address indexed charity);
event CharityVerified(address indexed charity);
event AdminChanged(address indexed newAdmin);

// Implement in functions
function registerCharity(address charity) public onlyAdmin nonZeroAddress(charity)
{
    registeredCharities[charity] = true;
    emit CharityRegistered(charity);
}
```

[M-3] Missing Registry Removal Functionality in `CharityRegistry.sol`

Description: The contract lacks functionality to remove registered or verified charities, making it impossible to handle compromised or defunct charities.

Impact:

- No way to remove malicious charities
- Cannot update registry for defunct organizations
- Permanent storage bloat
- No ability to handle compromised charity addresses

Recommended Mitigation: Add functions to remove charities with appropriate access controls.

```
event CharityRemoved(address indexed charity);

function removeCharity(address charity) public onlyAdmin {
    require(registeredCharities[charity], "Charity not registered");
    registeredCharities[charity] = false;
    verifiedCharities[charity] = false;
    emit CharityRemoved(charity);
}
```

[M-4] Insufficient Event Emission in `GivingThanks.sol`

Description: Critical state changes lack event emissions, making it difficult to track system activity off-chain.

Impact:

- Limited ability to monitor donations
- Difficult to track charity verification status
- Poor system auditability

Recommended Mitigation:

```
contract GivingThanks is ERC721URIStorage {
    event DonationMade(address indexed donor, address indexed charity, uint256
amount, uint256 tokenId);
    event RegistryUpdated(address indexed newRegistry);

    function donate(address charity) public payable {
        // ... existing code ...
        emit DonationMade(msg.sender, charity, msg.value, tokenCounter);
    }
}
```

[M-5] Inadequate Token URI Security

Description: The token URI generation includes sensitive data (donor address and amount) without encryption or privacy controls.

```
function _createTokenURI(address donor, uint256 date, uint256 amount) internal
pure returns (string memory)
```

Impact:

- Donor privacy concerns
- Public visibility of donation amounts
- Possible donor targeting

Recommended Mitigation: Implement optional privacy controls:

```
struct DonationMetadata {
    address donor;
    uint256 date;
    uint256 amount;
    bool isPrivate;
}

mapping(uint256 => DonationMetadata) private donationMetadata;

function _createTokenURI(uint256 tokenId) internal view returns (string memory) {
    DonationMetadata memory metadata = donationMetadata[tokenId];
    if (metadata.isPrivate) {
        return _createPrivateURI(tokenId);
    }
    return _createPublicURI(metadata);
}
```

[M-6] Lack of Donation Receipt Validation in `GivingThanks.sol`

Description: No verification mechanism exists to ensure donations were actually received by charities.

Recommended Mitigation:

```
contract GivingThanks is ERC721URIStorage {
    mapping(address => uint256) public charityDonations;

    function donate(address charity) public payable {
        // ... existing code ...
        charityDonations[charity] += msg.value;
        emit DonationReceived(charity, msg.value);
    }

    function verifyDonation(uint256 tokenId) public view returns (bool) {
        // Verification logic
    }
}
```

[M-7] Missing Contract Upgradability

Description: Neither contract implements upgrade mechanisms, making bug fixes and improvements difficult.

Recommended Mitigation: Implement upgradeable contract pattern:

```
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";

contract GivingThanksUpgradeable is Initializable, ERC721URIStorageUpgradeable {
    function initialize(address _registry) public initializer {
        __ERC721_init("DonationReceipt", "DRC");
        registry = CharityRegistry(_registry);
        owner = msg.sender;
    }
}
```

Low

[L-1] No Two-Step Admin Transfer in `CharityRegistry.sol`

Description: Admin transfer happens in a single step, which could lead to permanent loss of admin access if wrong address is provided.

Impact:

- Risk of losing admin access through typos
- No confirmation from new admin
- No way to recover from mistakes

Recommended Mitigation: Implement a two-step transfer pattern like the given below:

```
address public pendingAdmin;

function initiateAdminTransfer(address newAdmin) public onlyAdmin
nonZeroAddress(newAdmin) {
    pendingAdmin = newAdmin;
    emit AdminTransferInitiated(admin, newAdmin);
}

function acceptAdminTransfer() public {
    require(msg.sender == pendingAdmin, "Only pending admin can accept");
    admin = pendingAdmin;
    pendingAdmin = address(0);
    emit AdminTransferCompleted(admin);
}
```

[L-2] Missing Contract Documentation of `CharityRegistry.sol`

Description: The contract lacks comprehensive documentation and NatSpec comments, reducing maintainability and auditability.

Impact:

- Reduced code maintainability
- Difficulty in understanding contract functionality
- Increased risk of misuse
- Challenges in future audits

Proof of Concept:

Recommended Mitigation: Add comprehensive NatSpec documentation

```
/// @title CharityRegistry
/// @notice A registry for managing and verifying charitable organizations
/// @dev Implements a two-step verification process for charities
contract CharityRegistry {
    /// @notice The address of the contract administrator
    /// @dev Has exclusive rights to verify charities and change admin
    address public admin;
```

Informational

[I-1] Boolean Optimization in Mappings of `CharityRegistry.sol`

Description: Using boolean values in mappings is inefficient from a gas perspective. Each slot in storage is 32 bytes, but we only need 1 bit to store a boolean.

```
mapping(address => bool) public verifiedCharities;
mapping(address => bool) public registeredCharities;
```

Impact:

- Unnecessary gas consumption during storage operations
- Higher deployment costs
- Increased transaction costs for users

Gas Savings:

- Storage Write (Original): ~20,000 gas
- Storage Write (Optimized): ~5,000 gas
- Potential savings: ~15,000 gas per storage operation

Recommended Mitigation:

```
// Use uint256 for packing multiple booleans
mapping(address => uint256) public charityStatus;

// Constants for bit positions
uint256 constant REGISTERED_BIT = 1;
uint256 constant VERIFIED_BIT = 2;

function registerCharity(address charity) public onlyAdmin {
    charityStatus[charity] |= REGISTERED_BIT;
    emit CharityRegistered(charity);
}

function isRegistered(address charity) public view returns (bool) {
    return charityStatus[charity] & REGISTERED_BIT != 0;
}
```

[I-2] Public functions not called in the `CharityRegistry.sol` contract.

Description: Functions marked as `public` that are never called internally by the contract should be marked as `external` to save gas.

Impact:

- Extra gas cost due to copying function parameters to memory
- Unnecessary overhead in function calls

Gas Savings:

- External vs Public call: ~100-200 gas per call
- Affects all read operations on the contract

Recommended Mitigation:

```
function isVerified(address charity) external view returns (bool)
```

[I-3] Redundant State Checks in `CharityRegistry.sol`

Description: Multiple require statements with state checks can be combined to save gas.

```
function verifyCharity(address charity) public {
    require(msg.sender == admin, "Only admin can verify");
    require(registeredCharities[charity], "Charity not registered");
    verifiedCharities[charity] = true;
}
```

Impact:

- Multiple state accesses
- Redundant SLOAD operations

Gas Savings: Combining checks: ~2,000 gas per transaction

Recommended Mitigation:

```
function verifyCharity(address charity) public {
    if (msg.sender != admin || !registeredCharities[charity]) {
        revert("Unauthorized or not registered");
    }
    verifiedCharities[charity] = true;
}
```