GivingThanks Audit Report

Version 1.0

Trashpirate.io

November 12, 2024

GivingThanks Audit Report

Trashpirate.io

November 12, 2024

Prepared by: Trashpirate Lead Auditors: - Nadina Oates

Table of Contents

Protocol Summary

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.

Disclaimer

The Trashpirate 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
	High	H	H/M	M
Likelihood	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

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

Scope

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

```
#-- src
#   #-- CharityRegistry.sol
#   #-- GivingThanks.sol
```

Roles

- 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.

Executive Summary

This audit report was prepared as part of a training competition **FirstFlight** on CodeHawks.io

Issues found

Severity	Number of issues found
High	2
Medium	1
Low	2
Info	2
Gas	2
Total	9

Findings

High

[H-1] Mixed up use of `CharityRegistry::registeredCharities` and `CharityRegistry::verifiedCharities` returns false verification status for charity

Summary The function `CharityRegistry::isVerified` returning a boolean that indicates whether a charity is verified or not always returns true for any registred charity.

Vulnerability Details Due to mix up of the state variables (mappings) `CharityRegistry::registeredCharities` and`CharityRegistry::verifiedCharities`any registered charity returns **`true`**when `CharityRegistry::isVerified` is called. The test `GivingThanks.t:testCannotDonateToUnverifiedCharity` passes because the `donate` function reverts when called. However, it reverts due to a bug in the constructor, not because the charity is not verified.

Proof of Concept

1. User registers a charity
2. User donates before charity is verified
3. Charity is considered as verified and donation is accepted

Code:

Use explicit revert check for `GivingThanks.t:testCannotDonateToUnverifiedCharity`:

```
    function testCannotDonateToUnverifiedCharity() public {
        address unverifiedCharity = address(0x4);

        // Unverified charity registers but is not verified
        vm.prank(unverifiedCharity);
        registryContract.registerCharity(unverifiedCharity);

        // Fund the donor
        vm.deal(donor, 10 ether);

        // Donor tries to donate to unverified charity
        vm.prank(donor);
+           vm.expectRevert(bytes("Charity not verified"));
-           vm.expectRevert();
        charityContract.donate{value: 1 ether}(unverifiedCharity);
```

```
16      }
```

Test will fail with:

```
1  [FAIL: call reverted as expected, but without data]
```

Impact The check if a charity is verified in `GivingThanks::donate`will always return true for a registered charity. This breaks the verification mechanism by the admin.

Recommendations Fix the mixed up state variables:

```
1   function isVerified(address charity) public view returns (bool) {
2  -        return registeredCharities[charity];
3  +        return verifiedCharities[charity];
4       }
```

[H-2] Wrong use of msg.sender in `GivingThanks:constructor` breaks protocol

Summary In `GivingThanks::constructor` the variable `msg.sender` is used to register the previously deloyed instance of `CharityRegistry`. Instead the constructor argument `_registry` should be used.

Vulnerability Details Using `msg.sender` to register the previously deployed contract instance of `CharityRegistry` means that the contract instance is not registered at the correct address. Because there is no contract instance deployed at the address `msg.sender` this breaks all functionality of `CharityRegistry` which is critical for proper functioning of the protocol.

Proof of Concept A simple way to verify whether the correct registry address is set in `GivingThanks::constructor` is to test the value of `registry` after deployment.

Code:

The following test if placed in `GivingThanks.t.sol` should pass if registry is configured correctly:

```
1  ```solidity
2  function testRegsitryAddress() public {
3      assertEq(address(registryContract), address(charityContract.
           registry()));
4  }
5  ```
```

Impact Protocol is broken as the `CharityRegistry` is not registered at the correct address.

Recommendations Replace `msg.sender` with `_registry` in `GivingThanks::constructor`:

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

Medium

[M-1] Missing Access Control for `GivingThanks::updateRegistry` Allows Anyone to Update the Registry Address

Summary The function `GivingThanks::updateRegistry`as no access control and anyone can call it to update the registry address.

Vulnerability Details Even though a owner is specified in the constructor it is not used to restrict access for the `GivingThanks::updateRegistry` function. This means anyone could call `GivingThanks::updateRegistry`and update the registry address that points to a different contract instance which makes the registry vulnerable to manipulation.

Proof of Concept

1. Deploy `GivingThanks` contract with registry
2. User deploys their own registry contract
3. Malicious user calls `GivingThanks::updateRegistry` with manipulated registry address
4. Donor can donate to unverified charity

Code:

The following test demonstrates such a scenario:

```
```solidity
function testRegistryManipulation() public {
    address maliciousUser = makeAddr("maliciousUser");

    // user deploys their own registry contract
```
```

```
6       vm.prank(maliciousUser);
7       CharityRegistry maliciousRegistry = new CharityRegistry();
8
9       // user registrs their own charity
10      vm.prank(maliciousUser);
11      maliciousRegistry.registerCharity(maliciousUser);
12
13      // user verifies their own charity
14      vm.prank(maliciousUser);
15      maliciousRegistry.verifyCharity(maliciousUser);
16
17      // user update registry address in GivingThanks contract
18      vm.prank(maliciousUser);
19      charityContract.updateRegistry(address(maliciousRegistry));
20
21      // donor donates to unverified charity
22      vm.deal(donor, 10 ether);
23      vm.prank(donor);
24      charityContract.donate{value: 1 ether}(maliciousUser);
25
26      // malicious user receives the money
27      assertEq(maliciousUser.balance, 1 ether);
28  }
29  ```
```

Impact The verification process is vulnerable to manipulation. Someone could easily deploy a custom registry contract that contains illegitimate charities without proper verification by the admin.

Recommendations Add check that allows only the owner to update the registry address (see diff below). In addtion, a function to change the owner (access controlled) is recommended. This code could also be written as a modifier. Or as an alternative, use modifier specified in `Ownable.sol`by the OpenZeppelin library via import and inheritance.

```
1  + error GivingThanks__NotAuthorized();
2
3  + if (msg.sender != owner){
4  +  revert GivingThanks__NotAuthorized();
5  + }
```

Low

[L-1] Missing checks for `address(0)` when assigning values to address state variables could break admin functions

Summary Zero address checks for the address state variables `admin` and `registry` are missing. This could lead to a situation where admin functions cannot be accessed anymore.

3 Found Instances:

- Found in src/CharityRegistry.sol Line: 29

```
1       function changeAdmin(address newAdmin) public {
2           require(msg.sender == admin, "Only admin can change admin"
                );
3   @>      admin = newAdmin;
4       }
```

- Found in src/GivingThanks.sol Line: 16

```
1       constructor(address _registry) ERC721("DonationReceipt", "DRC"
            ) {
2   @>      registry = CharityRegistry(msg.sender);
3           owner = msg.sender;
4           tokenCounter = 0;
5       }
```

- Found in src/GivingThanks.sol Line: 56

```
1       function updateRegistry(address _registry) public {
2   @>      registry = CharityRegistry(_registry);
3       }
```

Vulnerability Details The address state variables for `admin` and `registry` are not checked for zero address in the functions `CharityRegistry::verifyCharity` and `CharityRegistry::changeAdmin` allowing the admin/user to accidentally set the state variables to the zero address.

Proof of Concept

1. Admin accidentally sets the admin address to zero address
2. Unverified charity registeredCharities
3. Admin tries to verify charity but fails
4. Admin tries to change admin address but fails

Code:

The following test demonstrates that after setting the admin to the zero address the `CharityRegistry::verifyCharity` function and `CharityRegistry::changeAdmin` function are not accessible anymore.

```
    function testAdminSetToZero() public {
        // Admin sets admin address to zero
        vm.prank(admin);
        registryContract.changeAdmin(address(0x0));

        // Unverified charity registers but is not verified
        address unverifiedCharity = address(0x4);
        vm.prank(unverifiedCharity);
        registryContract.registerCharity(unverifiedCharity);

        // Admin tries to verify charity
        vm.prank(admin);
        vm.expectRevert(bytes("Only admin can verify"));
        registryContract.verifyCharity(charity);

        // Admin tries to change admin address
        vm.prank(admin);
        vm.expectRevert(bytes("Only admin can change admin"));
        registryContract.changeAdmin(admin);
    }
```

Impact If not checked for zero address, the `admin` address or `registry` address could accidentally be set to the zero address. In case for the `registry`, this might not be critical as the `registry` can be updated afterwards. However, for the `admin` address, this could lead to a situation where admin functions are not accessible anymore without the ability to fix it by calling the `CharityRegistry::changeAdmin`.

Recommendations Check for `address(0)` when assigning values to address state variables. For example:

```
    function changeAdmin(address newAdmin) public {
        require(msg.sender == admin, "Only admin can change admin");
+       require(newAdmin != address(0), "CharityRegistry: new admin is
    the zero address");
        admin = newAdmin;
    }
```

[L-2] Using `ERC721::_mint()` allows minting receipts to addresses which don't support ERC721 tokens

Summary Using `ERC721::_mint()` can mint ERC721 tokens to addresses which don't support ERC721 tokens. This means if a donor is calling `GivingThanks::donate` from an address which does not support the ERC721 standard, they will not receive a receipt.

Vulnerability Details `ERC721::_mint()` does not perform any checks to ensure that the recipient address can handle ERC721 tokens. This is fine in most cases, but issues can arise if the `GivingThanks::donate` functions is called from a contract that is not prepared to handle ERC721 tokens. Thus, the receipt issued by `GivingThanks` will be lost because it cannot be retreived from the contract that received the token.

Proof of Concept

1. Donor has a contract that does not support ERC721 tokens
2. Donor donates to the charity via contract
3. Charity contract mints a token using `ERC721::_mint()`
4. Transaction does not revert even though the ERC721 token receipt cannot be retreived from the contract
5. Donating to the charity via contract using `ERC721::_safeMint()` reverts when donor donates via contract

Code:

The following test demonstrates how using `ERC721::_mint()` does not revert if the recipient address does not support ERC721 tokens, but using `ERC721::_safeMint()` does.

```
function testMintVsSafeMint() public {
    uint256 donationAmount = 1 ether;

    // donor has dumb contract
    DumbContract dumbContract = new DumbContract();

    // Fund the donor
    vm.deal(address(dumbContract), 10 ether);

    // Donor donates to the charity via contract
    vm.prank(address(dumbContract));
    charityContract.donate{value: donationAmount}(charity);

    // Setup charity contract with safeMint
    GivingThanksSafeMint charityContractSafeMint = new
        GivingThanksSafeMint(address(registryContract));
```

```
16
17      // Donor donates to the charity via contract
18      vm.prank(address(dumbContract));
19      vm.expectRevert(abi.encodeWithSelector(IERC721Errors.
            ERC721InvalidReceiver.selector, address(dumbContract)));
20      charityContractSafeMint.donate{value: donationAmount}(charity);
21  }
```

An adapted version of `GivingThanks.sol` was used for this test to implement the `_safeMint` function:

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3
4  import "src/CharityRegistry.sol";
5  import "@openzeppelin/contracts/token/ERC721/extensions/
       ERC721URIStorage.sol";
6  import "@openzeppelin/contracts/access/Ownable.sol";
7  import "@openzeppelin/contracts/utils/Strings.sol";
8  import "@openzeppelin/contracts/utils/Base64.sol";
9
10 contract GivingThanksSafeMint is ERC721URIStorage {
11     CharityRegistry public registry;
12     uint256 public tokenCounter;
13     address public owner;
14
15     constructor(address _registry) ERC721("DonationReceipt", "DRC") {
16         registry = CharityRegistry(_registry);
17         owner = msg.sender;
18         tokenCounter = 0;
19     }
20
21     function donate(address charity) public payable {
22         require(registry.isVerified(charity), "Charity not verified");
23         (bool sent,) = charity.call{value: msg.value}("");
24         require(sent, "Failed to send Ether");
25
26         _safeMint(msg.sender, tokenCounter);
27
28         // Create metadata for the tokenURI
29         string memory uri = _createTokenURI(msg.sender, block.timestamp
               , msg.value);
30         _setTokenURI(tokenCounter, uri);
31
32         tokenCounter += 1;
33     }
34
35     function _createTokenURI(address donor, uint256 date, uint256
           amount) internal pure returns (string memory) {
36         // Create JSON metadata
37         string memory json = string(
```

```
38              abi.encodePacked(
39                  '{"donor":"',
40                  Strings.toHexString(uint160(donor), 20),
41                  '","date":"',
42                  Strings.toString(date),
43                  '","amount":"',
44                  Strings.toString(amount),
45                  '"}'
46              )
47          );
48
49          // Encode in base64 using OpenZeppelin's Base64 library
50          string memory base64Json = Base64.encode(bytes(json));
51
52          // Return the data URL
53          return string(abi.encodePacked("data:application/json;base64,",
                base64Json));
54      }
55
56      function updateRegistry(address _registry) public {
57          registry = CharityRegistry(_registry);
58      }
59  }
```

Impact When a donor donates via contract that does not support ERC721 tokens, the receipt will be lost and cannot be retreived from the contract. If a receipt is needed, this is an issue as receipts cannot be issued after the donation completed. However, this does not cause any loss of funds and is not a security issue.

Recommendations Use `_safeMint()` instead of `_mint()` for ERC721.

```
1       function donate(address charity) public payable {
2           require(registry.isVerified(charity), "Charity not verified");
3           (bool sent,) = charity.call{value: msg.value}("");
4           require(sent, "Failed to send Ether");
5
6   +       _safeMint(msg.sender, tokenCounter);
7   -       _mint(msg.sender, tokenCounter);
8
9           // Create metadata for the tokenURI
10          string memory uri = _createTokenURI(msg.sender, block.timestamp
                , msg.value);
11          _setTokenURI(tokenCounter, uri);
12
13          tokenCounter += 1;
14      }
```

Informational

[L-1] Differente and wide versions of solidity are used

Summary The use of caret in the pragma statements allows different versions of Solidity to be used to compile the contracts. This can lead to deployment instability, unexpected behavior and compilation issues.

2 Found Instances:

- Found in src/CharityRegistry.sol Line: 2

```
1 pragma solidity ^0.8.0;
```

- Found in src/GivingThanks.sol Line: 2

```
1 pragma solidity ^0.8.0;
```

Vulnerability Details Using a caret version for the pragma statement allows different versions to be used to compile the contracts. Solc compiler version `0.8.20`, which is used for the library Openzeppelin, switches the default target EVM version to Shanghai, which means that the generated bytecode will include `PUSH0` opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support `PUSH0`, otherwise deployment of your contracts will fail.

Slither output:

```
1  Different versions of Solidity are used:
2          - Version used: ['^0.8.0', '^0.8.20']
3          - ^0.8.0 (src/CharityRegistry.sol#3)
4          - ^0.8.0 (src/GivingThanks.sol#3)
5          - ^0.8.20 (lib/openzeppelin-contracts/contracts/access/Ownable.
               sol#4)
6          - ^0.8.20 (lib/openzeppelin-contracts/contracts/interfaces/
               IERC165.sol#4)
7          - ^0.8.20 (lib/openzeppelin-contracts/contracts/interfaces/
               IERC4906.sol#4)
8          - ^0.8.20 (lib/openzeppelin-contracts/contracts/interfaces/
               IERC721.sol#4)
9          - ^0.8.20 (lib/openzeppelin-contracts/contracts/interfaces/
               draft-IERC6093.sol#3)
10         - ^0.8.20 (lib/openzeppelin-contracts/contracts/token/ERC721/
               ERC721.sol#4)
11         - ^0.8.20 (lib/openzeppelin-contracts/contracts/token/ERC721/
               IERC721.sol#4)
```

```
12          - ^0.8.20 (lib/openzeppelin-contracts/contracts/token/ERC721/
                IERC721Receiver.sol#4)
13          - ^0.8.20 (lib/openzeppelin-contracts/contracts/token/ERC721/
                extensions/ERC721URIStorage.sol#4)
14          - ^0.8.20 (lib/openzeppelin-contracts/contracts/token/ERC721/
                extensions/IERC721Metadata.sol#4)
15          - ^0.8.20 (lib/openzeppelin-contracts/contracts/token/ERC721/
                utils/ERC721Utils.sol#4)
16          - ^0.8.20 (lib/openzeppelin-contracts/contracts/utils/Base64.
                sol#4)
17          - ^0.8.20 (lib/openzeppelin-contracts/contracts/utils/Context.
                sol#4)
18          - ^0.8.20 (lib/openzeppelin-contracts/contracts/utils/Panic.sol
                #4)
19          - ^0.8.20 (lib/openzeppelin-contracts/contracts/utils/Strings.
                sol#4)
20          - ^0.8.20 (lib/openzeppelin-contracts/contracts/utils/
                introspection/ERC165.sol#4)
21          - ^0.8.20 (lib/openzeppelin-contracts/contracts/utils/
                introspection/IERC165.sol#4)
22          - ^0.8.20 (lib/openzeppelin-contracts/contracts/utils/math/Math
                .sol#4)
23          - ^0.8.20 (lib/openzeppelin-contracts/contracts/utils/math/
                SafeCast.sol#5)
24          - ^0.8.20 (lib/openzeppelin-contracts/contracts/utils/math/
                SignedMath.sol#4)
```

Impact Unspecified solidity version can lead to deployment instability, unexpected behavior and compilation issues. It can also lead to deployment issues on chains that do not support PUSH0 which is included in solc version `0.8.20`.

Recommendations Specify the exact version of Solidity in the pragma statement that is compatible with the codebase and used dependencies. For example, use `pragma solidity 0.8.0;` instead of `pragma solidity ^0.8.0;`. Make sure the used solidity version is compatible with the dependencies. Also make sure correct EVM version is selected in case of deployment on L2 chains.

[I-2] State variable changes but no event is emitted.

State variable changes in this function but no event is emitted.

5 Found Instances

- Found in src/CharityRegistry.sol Line: 21

```
1       function registerCharity(address charity) public {
```

- Found in src/CharityRegistry.sol Line: 28

```
1       function verifyCharity(address charity) public {
```

- Found in src/CharityRegistry.sol Line: 43

```
1       function changeAdmin(address newAdmin) public {
```

- Found in src/GivingThanks.sol Line: 31

```
1       function donate(address charity) public payable {
```

- Found in src/GivingThanks.sol Line: 71

```
1       function updateRegistry(address _registry) public {
```

Gas

[G-1] `public` functions not used internally are marked `external` resulting in more gas ususage

Summary Several functions are marked as `public` but are not used internally, which uses more gas.

Vulnerability Details Functions should be marked as external whenever possible, as external functions are more gas-efficient than public ones. This is because external functions expect arguments to be passed from the external call, which can save gas. See Function Optimization.

6 Found Instances

- Found in src/CharityRegistry.sol Line: 21

```
1       function registerCharity(address charity) public {
```

- Found in src/CharityRegistry.sol Line: 28

```
1       function verifyCharity(address charity) public {
```

- Found in src/CharityRegistry.sol Line: 38

```
1       function isVerified(address charity) public view returns (bool
            ) {
```

- Found in src/CharityRegistry.sol Line: 43

```
1      function changeAdmin(address newAdmin) public {
```

- Found in src/GivingThanks.sol Line: 31

```
1      function donate(address charity) public payable {
```

- Found in src/GivingThanks.sol Line: 71

```
1      function updateRegistry(address _registry) public {
```

Impact Marking the functions as `public` uses more gas.

Recommendation Mark the functions as `external` instead of `public`. For example:

```
+   function donate(address charity) external payable {
-   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);

        // Create metadata for the tokenURI
        string memory uri = _createTokenURI(msg.sender, block.timestamp
            , msg.value);
        _setTokenURI(tokenCounter, uri);

        tokenCounter += 1;
    }
```

[G-2] State variable could be declared immutable

State variables that are should be declared immutable to save gas. Add the `immutable` attribute to state variables that are only changed in the constructor

1 Found Instances

- Found in src/GivingThanks.sol Line: 20

```
1      address public owner;
```