Christmas Dinner Audit Report

Prepared by: HEXXA Protocol

Christmas Dinner Audit Report

- Title: Christmas Dinner Audit Report
- author: Gaurang Bharadava
- date: December 27, 2024

Prepared by: HEXXA Protocol

- Lead Auditors: Gaurang Bharadava

Assisting Auditors:

- None

Table of contents

▶ Details
See table

About Gaurang Bharadava

Gaurang Bharadava is experienced smart contract engineer and security researcher. Building HEXXA Protocol, will provide Smart contract development and security audit.

Disclaimer

The HEXXA Protocol 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

Audit Details

Scope

```
./src/
-- ChristmasDinner.sol
```

Compatibilities

```
Compatibilities:
  Blockchains:
      - Ethereum
  Tokens:
```

```
        - ETH
        - WETH
        - WBTC
        - USDC
```

Protocol Summary

This contract is designed as a modified fund me. It is supposed to sign up participants for a social christmas dinner (or any other dinner), while collecting payments for signing up.

Roles

- `Host`: The person doing the organization of the event. Receiver of the funds by the end of `deadline`. Privilegded Role, which can be handed over to any `Participant` by the current `host`
- `Participant`: Attendees of the event which provided some sort of funding. `Participant` can become new `Host`, can continue sending money as Generous Donation, can sign up friends and can become `Funder`.
- `Funder`: Former Participants which left their funds in the contract as donation, but can not attend the event. `Funder` can become `Participant` again BEFORE deadline ends.

Executive Summary

Issues found

Severity	Number of issues found
High	3
Medium	2
Low	2
Info	2
Gas	1
Total	0

Findings

High

[H-1] Reentrancy is possible due to wrong `nonReentrant` modifier which lead to drain all the funds.

Description:

```
    modifier nonReentrant() {
          require(!locked, "No re-entrancy");
          _;
          locked = false;
    }
```

The `nonReentrant` modifier is not written properly as it is not locking the user as per docuumnetation provided. Here `nonReentrant` modifier checks weather the locked variable is true or false, If it is true then and only then the function able to go for further execution. Also here `locked` variable is only set to false. it has not set to true for preventing reentrancy attack.

Impact: A malicious user or a contract can drain all the ETH.

Recommended Mitigation: To mitigate reentrancy attack we have to set the `locked` variable to true so that user can not reenter into the function before function execution complate.

1. Suggested `nonReentrant` modifier:

```
    modifier nonReentrant() {
        require(!locked, "No re-entrancy");
+       locked = true;
        _;
        locked = false;
    }
```

2. `openzeppelin reentrancyguard` is suggested to mitigate the risk of reetrancy attack.

[H-2] Missing check for `zero` address in `ChristmasDinner:recieve` function.

Description: `ChistmasDinner:recieve` function is not checking that who is the participant, Weather it is real person or zero address, can participate.

```
    receive() external payable {
@>      etherBalance[msg.sender] += msg.value;
        emit NewSignup(msg.sender, msg.value, true);
    }
```

By exploiting this missing check malicious user can become host and lock all the funds.

Impact: `Zero address` or `malicious contract` can participate and become host, lead to lock all the ETH funds in contract.

Proof of Concept: Add this to `ChristmasDinnerTest.t.sol`

Code:

```
```

```
    function testZeroAddressCanparticipate() public {
       address payable hacker = payable(address(0));
        vm.deal(hacker, 10e18);
        vm.prank(hacker);
        (bool sent,) = address(cd).call{value: 1e18}("");
        require(sent, "transfer failed");
        assertEq(hacker.balance, 9e18);
        assertEq(address(cd).balance, 1e18);
    }
```

Recommended Mitigation: The Zero address check can be added to `recieve` function.

```
    receive() external payable {
+     if(msg.sender == address(0)) {
+      revert();
+     }
      etherBalance[msg.sender] += msg.value;
      emit NewSignup(msg.sender, msg.value, true);
    }
```

[H-3] Host unable withdraw ETH balance from contract.

Description: `ChistmasDinner::withdraw` frunction is sending all the allowed tokens to the host. but `withdraw` function is not sending ETH balance to the host. which lead to stuck all the ETH funds in contract.

Impact: ETH balace will stuck into the contract, unable to withdraw by host.

Proof of Concept: Add this to `ChristmasDinnerTest.t.sol`.

Code:

```
    function testHostCanNotWithdrawETH() public {
        _makeParticipants();
        vm.deal(user2, 10e18);
        vm.prank(user2);
        (bool sent,) = address(cd).call{value: 1e18}("");
        require(sent);
        assertEq(address(cd).balance, 1e18);
        assertEq(address(deployer).balance, 0);
        vm.prank(deployer);
        cd.withdraw();
        assertEq(address(cd).balance, 1e18);
        assertEq(deployer.balance, 0);
    }
```

Recommended Mitigation: The function shoild lool like as recommended below.

```
    function withdraw() external onlyHost {
        address _host = getHost();
        i_WETH.safeTransfer(_host, i_WETH.balanceOf(address(this)));
        i_WBTC.safeTransfer(_host, i_WBTC.balanceOf(address(this)));
        i_USDC.safeTransfer(_host, i_USDC.balanceOf(address(this)));
+       _refundETH(_host)
    }
```

Medium

[M-1] ETH payer can not be participant. As recieve function is not updating the participants list.

Description: `ChristmasDinner::recieve` function is not updating participants list for that users who pays ETH to participate in dinner. The Reecieve function is not following doccumentation, as it is not handling user signup properly.

```
  receive() external payable {
@>      etherBalance[msg.sender] += msg.value;
        emit NewSignup(msg.sender, msg.value, true);
  }
```

Impact: ETH payer will assume that thay are not participants, will pay ETh again and again to participate.

Proof of Concept: Add this to `ChristmasDinnerTest.t.sol`.

Code:

```
  function testRecieveFunctionDoesNotUpdatingParticipants() public {
        address participant = makeAddr("User");
        vm.deal(participant, 10e18);
        vm.prank(participant);
        (bool sent, ) = address(cd).call{value: 1e18}("");
        require(sent);
        assertEq(cd.getParticipationStatus(participant), false);
  }
```

Recommended Mitigation: The `recieve` function shold be like mentioned below.

```
  receive() external payable {
+   participant[msg.sender] = true;
    etherBalance[msg.sender] += msg.value;
    emit NewSignup(msg.sender, msg.value, true);
  }
```

[M-2] Host can withdraw all the amount of token before deadline from the contract.

Description: In `ChristmasDinner::withdraw` function there is not check for deadline before withdrawinng all the funds for the contract to the host, Allows host to entirly sweep the contract funds before deadline end.

```
@>  function withdraw() external onlyHost {
        address _host = getHost();
        i_WETH.safeTransfer(_host, i_WETH.balanceOf(address(this)));
        i_WBTC.safeTransfer(_host, i_WBTC.balanceOf(address(this)));
        i_USDC.safeTransfer(_host, i_USDC.balanceOf(address(this)));
    }
```

Impact: Users who want refund unable to get their deposited amount.

Proof of Concept: Add this to `ChristmasDinnerTest.t.sol`.

Code:

```
    function testUserCannotGetTheirAmoutBackCollectedByHost() public {
        vm.warp(1 + 3 days);
        vm.startPrank(user1);
        cd.deposit(address(wbtc), 2e18);
        assertEq(wbtc.balanceOf(address(cd)), 2e18);
        vm.stopPrank();
        vm.prank(deployer);
        cd.withdraw();
        vm.startPrank(user1);
        vm.warp(1 + 3 days);
        vm.expectRevert();
        cd.refund();
        vm.stopPrank();
        assertEq(wbtc.balanceOf(user1), 0);
    }
```

Recommended Mitigation: Add the check that does not allow host to withdraw funds from the contract before deadline. Add this to `ChristmasDinner::withdraw` function.

```
-   function withdraw() external onlyHost {
+   function withdraw() external beforeDeadline onlyHost {
        address _host = getHost();
        i_WETH.safeTransfer(_host, i_WETH.balanceOf(address(this)));
        i_WBTC.safeTransfer(_host, i_WBTC.balanceOf(address(this)));
        i_USDC.safeTransfer(_host, i_USDC.balanceOf(address(this)));
    }
```

Low

[L-1] Missing `zero` address check in `ChristmasDinner:changeHost` function.

Description: `ChristmasDinner::changeHost` function does not check weather the address is selected to become host is `zero` address ot not.

```
@>  function changeHost(address _newHost) external onlyHost {
        if(!participant[_newHost]) {
            revert OnlyParticipantsCanBeHost();
        }
        host = _newHost;
        emit NewHost(host);
    }
```

Impact: All funds will stuck in contract.

Recommended Mitigation: The check for zero address have to be added before changing the host.

```
    function changeHost(address _newHost) external onlyHost {
+       if(_newHost == address(0)) {
+           revert();
+       }
        if(!participant[_newHost]) {
            revert OnlyParticipantsCanBeHost();
        }
        host = _newHost;
        emit NewHost(host);
    }
```

[L-2] Missing `deadline` check in `receive` function

Description: `ChristmasDinner::receive` function does not checks the deadline for the user who are participating by depositiong ETH.

Impact: Anyone can participate after deadline.

Proof of Concept:

Add this into `ChristmasDinnerTest.t.sol`

Code:

```
    function testUserCanDepositeAfterDeadline() public {
        vm.deal(user1, 10e18);
        vm.warp(1 + 8 days);
        vm.startPrank(user1);
        (bool ok, ) = address(cd).call{value: 1e18}("");
        vm.stopPrank();
```

```
        assert(ok);
    }
```

Recommended Mitigation: The `receive` frunction should be like

```
    receive() external payable {
+     if(block.timestamp > deadline) {
+       revert BeyondDeadline()
+     }
      etherBalance[msg.sender] += msg.value;
      emit NewSignup(msg.sender, msg.value, true);
    }
```

Informational

[I-1] Defining the state variables.

The state variables in `ChristmasDinner` shold be as suggested below.

```
    address public s_host;
    uint256 public s_deadline;
    bool public s_deadlineSet = false;
    bool private s_locked = false;
    mapping (address user => bool) s_participant;
    mapping (address user => mapping (address token => uint256 balance ))
s_balances;
    mapping (address user => uint256 amount) s_etherBalance;
    mapping (address token => bool ) s_whitelisted;
```

[I-2] `ChristmasDinner::_refundETH` function does not follow CEI.

The function should look like as mentioned.

```
    function _refundETH(address payable _to) internal {
        uint256 refundValue = etherBalance[_to];
+       etherBalance[_to] = 0;
        _to.transfer(refundValue);
-       etherBalance[_to] = 0;
    }
```

[I-3] Event is encountered after making external call in `ChristmasDinner::refund` function.

The `ChristmasDinner::refund::Refunded` event should emmit before making exteranl calls.

```
    function refund() external nonReentrant beforeDeadline {
        address payable _to = payable(msg.sender);
+       emit Refunded(msg.sender);
        _refundERC20(_to);
        _refundETH(_to);
-       emit Refunded(msg.sender);
    }
```

Gas

[G-1] Missing check for participant in `ChristmasDinner:refund` function.

Description: `ChristmasDinner::refund` function does not checks for participant before refund the funds to the user.

```
    function refund() external nonReentrant beforeDeadline {
@>    address payable _to = payable(msg.sender);
      _refundERC20(_to);
      _refundETH(_to);
      emit Refunded(msg.sender);
    }
```

Impact: Unnessesery gas is used to perform the refund task if the caller is not the participant.

Proof of Concept: Add these to `ChristmasDinnerTest.t.sol`

code:

```
    function testlossGas() public {
        vm.txGasPrice(1);
        address user = makeAddr("user");
        uint256 gasBefore = gasleft();
        vm.prank(user);
        cd.refund();
        uint256 gasAfter = gasleft();
        assert(gasBefore > gasAfter);
        console.log("Gas before: ", gasBefore);
        console.log("Gas after: ", gasAfter);
        console.log("Used gas:, ", gasBefore - gasAfter);
    }
```

If `refund` function does not check participant before doing any stuff, The output of this test is:

```
gasBefore = 1073716296
gasAfter = 1073661511
gasDiffrence = 54785
```

After adding missing check the output will be :

```
gasBefore = 1073716296
gasAfter = 1073703811
gasDiffrence = 12485
```

Check the output by adding reccomended mitigation by this test

Code:

```
    function testlossGas() public {
        vm.txGasPrice(1);
        address user = makeAddr("user");
        uint256 gasBefore = gasleft();
        vm.prank(user);
        vm.expectRevert();
        cd.refund();
        uint256 gasAfter = gasleft();
        assert(gasBefore > gasAfter);
        console.log("Gas before: ", gasBefore);
        console.log("Gas after: ", gasAfter);
        console.log("Used gas:, ", gasBefore - gasAfter);
    }
```

Recommended Mitigation: The function should be like this.

```
    function refund() external nonReentrant beforeDeadline {
+       if(participant[msg.sender] == false) {
+           revert();
+       }
        address payable _to = payable(msg.sender);
        _refundERC20(_to);
        _refundETH(_to);
        emit Refunded(msg.sender);
    }
```