4th05

1WProject

Security Review Report

13 November 2024

Security Review Report

4th05

November 13, 2024

Table of Contents

Protocol Summary

The One World Project leverages blockchain technology to revolutionize global capital markets. By democratizing access to funding and resources, we empower communities, foster economic growth, and create sustainable financial opportunities worldwide.

One World Project is a dynamic DAO marketplace where users can actively participate in decentralized organizations. This repository contains smart contracts for creating and managing DAO memberships using ERC1155 tokens. The contracts facilitate the creation and management of DAO memberships. Each DAO membership is represented by an ERC1155 token with different tiers.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Risk Classification

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

Overview

Contest platform	Codehawks
LOC	541
Language	Solidity
Commit	1e872c7ab393c380010a507398d4b4caca1ae32b
Previous audits	Cyfrin Private Audit, LightChaser

Scope

MembershipFactory

- Create New DAO Membership: Deploys a new proxy contract for the DAO membership.
- Update DAO Membership: Updates the tier configurations for a specific DAO.

- Join DAO: Allows a user to join a DAO by purchasing a membership NFT at a specific tier.
- Upgrade Tier: Allows users to upgrade their tier within a sponsored DAO.
- Set Currency Manager: Updates the currency manager contract.
- Call External Contract: Allows the factory to perform external calls to other contracts.

MembershipERC1155

- Initialize: Initializes the contract with the name, symbol, URI, and creator address.
- Mint Tokens: Mints new tokens.
- Burn Tokens: Burns existing tokens.
- Claim Profit: Allows users to claim profits from the profit pool.
- Send Profit: Distributes profits to token holders.
- Call External Contract: Allows the contract to perform external calls to other contracts.

```
#-- OWPIdentity.sol
#-- dao
#   #-- CurrencyManager.sol
#   #-- MembershipFactory.sol
#   #-- interfaces
#   #   #-- ICurrencyManager.sol
#   #   #-- IERC1155Mintable.sol
#   #-- libraries
#   #  #-- MembershipDAOStructs.sol
#   #-- tokens
#       #-- MembershipERC1155.sol
#-- meta-transaction
      #-- EIP712Base.sol
      #-- NativeMetaTransaction.sol
```

Issues found

Severity	Number of issues found
High	0
Medium	0
Low	1
Info	0

Findings

[L1] Missing currency check when joining a DAO leading to potential loss of profits and protocol fees

Relevant GitHub Links https://github.com/Cyfrin/2024-11-one-world/blob/main/contracts/dao/MembershipFactory.s
L147

Summary The protocol has implemented some allowed currencies to use its functions to ensure payments are received. However, using `MembershipFactory::joinDAO` no check is made to ensure payments are all made with allowed currencies. (As done using the `MembershipFactory::createNewDAOMembership`)

Vulnerability Details No check made through the `currencyManager::isCurrencyWhitelisted` function when joining a DAO using `MembershipFactory::joinDAO`. This because the OG currency could have been removed from the list because the protocol cannot accept that currency anymore.

Impact As not stated otherwise in the documentation provided it is reasonable to assume that payments made with no `viewWhitelistedCurrencies` could not be received by the contract/wallet causing this way a loss of all the funds.

POC

```solidity
// SPDX-License-Identifier: MIT
pragma solidity 0.8.22;

import { IMembershipERC1155 } from "../contracts/dao/interfaces/IERC1155Mintable.sol";
import { ICurrencyManager } from "../contracts/dao/interfaces/ICurrencyManager.sol";
import { CurrencyManager } from "../contracts/dao/CurrencyManager.sol";
import { MembershipERC1155 } from "../contracts/dao/tokens/MembershipERC1155.sol";
import { MembershipFactory } from "../contracts/dao/MembershipFactory.sol";
import { DAOConfig, DAOInputConfig, TierConfig, DAOType, TIER_MAX } from "../contracts/dao/libraries/MembershipDAOStructs.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/proxy/transparent/TransparentUpgradeableProxy.sol";
import "@openzeppelin/contracts/proxy/transparent/ProxyAdmin.sol";
import { NativeMetaTransaction } from "../contracts/meta-transaction/NativeMetaTransaction.sol";
import {Test, console} from "../forge-std/src/Test.sol";
```

```


contract OWPTest is Test {

string baseURI = "test_trial";
address public owpWallet = address(1);
address public user = makeAddr("user");
event firstslotevent (uint256);
MembershipFactory membershipfactory;
MembershipERC1155 membershipImplementation;
CurrencyManager currencymanager;

function setUp () public {

    currencymanager = new CurrencyManager();
    membershipImplementation = new MembershipERC1155();
    membershipfactory = new MembershipFactory(address(currencymanager),
        owpWallet, baseURI, address(membershipImplementation));

}

function test_paymentWithNotAllowedCurrency () public {

Currency currency1 = new Currency("currency1", "CUR1");
Currency currency2 = new Currency("currency2", "CUR2");
Currency currency3 = new Currency("currency3", "CUR3");

currencymanager.addCurrency(address(currency1));
currencymanager.addCurrency(address(currency2));
currencymanager.addCurrency(address(currency3));

uint256 numtiers = 6;
TierConfig[] memory tierconf = new TierConfig[](numtiers);
    for (uint256 i=0; i<numtiers; ++i) {
        tierconf[i] = TierConfig({amount:10, price:10, power:1, minted
            :0});
    }

DAOInputConfig memory conf = DAOInputConfig({ensname: "BestDAO",
    daoType: DAOType.PUBLIC, currency:address(currency1), maxMembers
    :200, noOfTiers:6});

address newdao = membershipfactory.createNewDAOMembership(conf,
    tierconf);

currencymanager.removeCurrency(address(currency1));

vm.deal(user, 1 ether);
vm.startPrank(user);
currency1.mint(user);
currency1.approve(address(membershipfactory), type(uint256).max);

vm.expectRevert();
membershipfactory.joinDAO(newdao, 4);
}



contract Currency is ERC20 {

constructor(string memory _name, string memory _symbol) ERC20 (_name, _symbol){}

function mint(address _user) public {
_mint(_user, 1 ether);
}

}
```

Tools Used

Manual review

Recommendations

```
function joinDAO(address daoMembershipAddress, uint256 tierIndex) external {
        require(daos[daoMembershipAddress].noOfTiers > tierIndex, "Invalid tier.");
        require(daos[daoMembershipAddress].tiers[tierIndex].amount > daos[daoMembershipAddress].tiers[tierIndex].minted, "Tier full.");
+        require(currencyManager.isCurrencyWhitelisted(daos[daoMembershipAddress].currency), "Currency not accepted.");
        uint256 tierPrice = daos[daoMembershipAddress].tiers[tierIndex].price;
        uint256 platformFees = (20 * tierPrice) / 100;
        daos[daoMembershipAddress].tiers[tierIndex].minted += 1;
        IERC20(daos[daoMembershipAddress].currency).transferFrom(_msgSender(), owpWallet, platformFees);
        IERC20(daos[daoMembershipAddress].currency).transferFrom(_msgSender(), daoMembershipAddress, tierPrice - platformFees);
        IMembershipERC1155(daoMembershipAddress).mint(_msgSender(), tierIndex, 1);
        emit UserJoinedDAO(_msgSender(), daoMembershipAddress, tierIndex);
    }
```