Armors Labs

Oh ! Finance

Smart Contract Audit

Oh！Finance Audit Summary

Project name : Oh！Finance Contract

Project address: https://oh.finance

Code URL : https://github.com/OhFinance/oh-contracts

Commit : db8eda7caf4e7076d9c23ad633ba57d27d4b5a03

Project target : Oh！Finance Contract Audit

Blockchain : Ethereum

Test result : PASSED

Audit Info

Audit NO : 0X202106090006

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

Oh！Finance Audit

The Oh！Finance team asked us to review and audit their Oh！Finance contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
Oh！Finance Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-06-09

Audit results

Note that: This audit includes OhToken.sol file and OhTimeLock.sol file.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Oh！Finance contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

(Statement: Armors Labs reports only on facts that have occurred or existed before this report is issued and assumes corresponding responsibilities. Armors Labs is not able to determine the security of its smart contracts and is not responsible for any subsequent or existing facts after this report is issued. The security audit analysis and other content of this report are only based on the documents and information provided by the information provider to Armors Labs at the time of issuance of this report (" information provided " for short). Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused.)

Audited target file

file	md5
./OhToken.sol	631442a69eeb296e634a0c0ce7e16a31
./libraries/TransferHelper.sol	2895a3c9122539f2d05075f6c90554f3
./OhTimelock.sol	240c71fed31aa28adfefe63332da0e9c
./registry/OhSubscriber.sol	8c4bfd7808b919cb7a0a8ea1830f2a4b
./registry/OhRegistry.sol	038fc6456eb93bc93b660743c9395cab
./interfaces/ISubscriber.sol	49ab2f1989883f4fe22a5cde72ad173b
./interfaces/IRegistry.sol	05673fdeea05bf3162bf4cb67d471dee
./interfaces/ITimelock.sol	f5f88105d5716bf82cdd3d635d09d702
./interfaces/IToken.sol	8d3da343072234b9ba5bf5f10e122cb5

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe

Vulnerability	status
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

```
// SPDX-License-Identifier: MIT

pragma solidity 0.7.6;

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import {SafeMath} from "@openzeppelin/contracts/math/SafeMath.sol";
import {IToken} from "./interfaces/IToken.sol";
import {OhSubscriber} from "./registry/OhSubscriber.sol";

/// @title Oh! Finance Token
/// @notice Protocol Governance and Profit-Share ERC-20 Token
contract OhToken is ERC20("Oh! Finance", "OH"), OhSubscriber, IToken {
    using SafeMath for uint256;

    /// @notice A checkpoint for marking number of votes from a given block
    struct Checkpoint {
        uint32 fromBlock;
        uint256 votes;
    }

    /// @notice The max token supply, minted on initialization. 100m tokens.
    uint256 public constant MAX_SUPPLY = 100000000e18;

    /// @notice The EIP-712 typehash for the delegation struct used by the contract
    bytes32 public constant DELEGATION_TYPEHASH =
        keccak256("Delegation(address delegator,address delegatee,uint256 nonce,uint256 deadline)");

    /// @notice the EIP-712 typehash for approving token transfers via signature
    bytes32 public constant PERMIT_TYPEHASH =
        keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)

    /// @notice The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH =
        keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)

    /// @notice The EIP-712 typehash used for replay protection, set at deployment
    // solhint-disable-next-line
    bytes32 public immutable DOMAIN_SEPARATOR;
    /// @notice Delegate votes from `msg.sender` to `delegatee`
    mapping(address => address) public delegates;

    /// @notice A record of votes checkpoints for each account, by index
    mapping(address => mapping(uint32 => Checkpoint)) public checkpoints;

    /// @notice A record of states for signing / validating signatures
    mapping(address => uint256) public nonces;

    /// @notice The number of checkpoints for each account
    mapping(address => uint32) public numCheckpoints;

    /// @notice An event thats emitted when an account changes its delegate
    event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed to

    /// @notice An event thats emitted when a delegate account's vote balance changes
    event DelegateVotesChanged(address indexed delegate, uint256 previousBalance, uint256 newBalance)

    constructor(address registry_) OhSubscriber(registry_) {
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name())), keccak256(bytes("1")), getChainId()
        );

        _mint(msg.sender, MAX_SUPPLY);
    }

    /// @notice Delegate votes from `msg.sender` to `delegatee`
    /// @param delegatee The address to delegate votes to
    function delegate(address delegatee) external override {
        return _delegate(msg.sender, delegatee);
    }

    /// @notice Delegates votes from `delegator` to `delegatee`
    /// @param delegator the address holding tokens
    /// @param delegatee The address to delegate votes to
    /// @param deadline The time at which to expire the signature
    /// @param v The recovery byte of the signature
    /// @param r Half of the ECDSA signature pair
    /// @param s Half of the ECDSA signature pair
    function delegateBySig(
        address delegator,
        address delegatee,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external override {
        // solhint-disable-next-line
        require(block.timestamp <= deadline, "Delegate: Invalid Expiration");
        require(delegator != address(0), "Delegate: Invalid Delegator");

        uint256 currentValidNonce = nonces[delegator];
        bytes32 digest =
            keccak256(
                abi.encodePacked(
                    "\x19\x01",
                    DOMAIN_SEPARATOR,
                    keccak256(abi.encode(DELEGATION_TYPEHASH, delegator, delegatee, currentValidNonce
                )
            );

        require(delegator == ecrecover(digest, v, r, s), "Delegate: Invalid Signature");
        nonces[delegator] = currentValidNonce.add(1);
        return _delegate(delegator, delegatee);
    }
```

```
/// @dev implements the permit function per EIP-712
/// @param owner the owner of the funds
/// @param spender the spender
/// @param value the amount
/// @param deadline the deadline timestamp, type(uint256).max for max deadline
/// @param v the recovery byte of the signature
/// @param r half of the ECDSA signature pair
/// @param s half of the ECDSA signature pair
function permit(
    address owner,
    address spender,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external override {
    require(block.timestamp <= deadline, "Permit: Invalid Deadline");
    require(owner != address(0), "Permit: Invalid Owner");

    uint256 currentValidNonce = nonces[owner];
    bytes32 digest =
        keccak256(
            abi.encodePacked(
                "\x19\x01",
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, currentValidNonce, d
            )
        );

    require(owner == ecrecover(digest, v, r, s), "Permit: Invalid Signature");
    nonces[owner] = currentValidNonce.add(1);
    return _approve(owner, spender, value);
}

/// @notice Gets the current votes balance for `account`
/// @param account The address to get votes balance
/// @return The number of current votes for `account`
function getCurrentVotes(address account) external view override returns (uint256) {
    uint32 nCheckpoints = numCheckpoints[account];
    return nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0;
}

/// @notice Determine the prior number of votes for an account as of a block number
/// @dev Block number must be a finalized block or else this function will revert to prevent misi
/// @param account The address of the account to check
/// @param blockNumber The block number to get the vote balance at
/// @return The number of votes the account had as of the given block
function getPriorVotes(address account, uint256 blockNumber) external view override returns (uint
    require(blockNumber < block.number, "GetPriorVotes: Invalid Block");

    uint32 nCheckpoints = numCheckpoints[account];
    if (nCheckpoints == 0) {
        return 0;
    }

    // First check most recent balance
    if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
        return checkpoints[account][nCheckpoints - 1].votes;
    }

    // Next check implicit zero balance
    if (checkpoints[account][0].fromBlock > blockNumber) {
        return 0;
    }
```

```
        uint32 lower = 0;
        uint32 upper = nCheckpoints - 1;
        while (upper > lower) {
            uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
            Checkpoint memory cp = checkpoints[account][center];
            if (cp.fromBlock == blockNumber) {
                return cp.votes;
            } else if (cp.fromBlock < blockNumber) {
                lower = center;
            } else {
                upper = center - 1;
            }
        }
        return checkpoints[account][lower].votes;
    }

    /// @notice Destroys an amount of tokens from the caller
    /// @param amount The amount of tokens to burn
    function burn(uint256 amount) public override {
        _burn(msg.sender, amount);
    }

    /// @notice Creates an amount of tokens on a recipient address
    /// @param recipient The receiver of the tokens
    /// @param amount The amount of tokens to mint
    /// @dev callable by governance only
    function mint(address recipient, uint256 amount) public override onlyGovernance {
        _mint(recipient, amount);
    }

    function _burn(address from, uint256 amount) internal override {
        super._burn(from, amount);
        _moveDelegates(delegates[from], address(0), amount);
    }

    function _mint(address to, uint256 amount) internal override {
        require(totalSupply().add(amount) <= MAX_SUPPLY, "Token: Max Supply Exceeded");
        super._mint(to, amount);
        _moveDelegates(address(0), delegates[to], amount);
    }

    function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal override {
        super._transfer(from, to, amount);
        _moveDelegates(delegates[from], delegates[to], amount);
    }

    function _delegate(address delegator, address delegatee) internal {
        address currentDelegate = delegates[delegator];
        uint256 delegatorBalance = balanceOf(delegator); // balance of underlying CAKEs (not scaled);
        delegates[delegator] = delegatee;

        emit DelegateChanged(delegator, currentDelegate, delegatee);

        _moveDelegates(currentDelegate, delegatee, delegatorBalance);
    }

    // move an amount of delegates from srcRep to dstRep
    function _moveDelegates(
        address srcRep,
        address dstRep,
        uint256 amount
    ) internal {
        if (srcRep != dstRep && amount > 0) {
            if (srcRep != address(0)) {
                // decrease old representative
                uint32 srcRepNum = numCheckpoints[srcRep];
                uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
                uint256 srcRepNew = srcRepOld.sub(amount);
                _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
            }

            if (dstRep != address(0)) {
                // increase new representative
                uint32 dstRepNum = numCheckpoints[dstRep];
                uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
                uint256 dstRepNew = dstRepOld.add(amount);
                _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
            }
        }
    }

    function _writeCheckpoint(
        address delegatee,
        uint32 nCheckpoints,
        uint256 oldVotes,
        uint256 newVotes
    ) internal {
        uint32 blockNumber = uint32(block.number);

        // if the user has already been delegated to this block, update vote count
        if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
            checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
        } else {
            // else write a new checkpoint with updated vote count
            checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
            numCheckpoints[delegatee] = nCheckpoints + 1;
        }

        emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
    }

    function getChainId() internal pure returns (uint256 chainId) {
        // solhint-disable-next-line no-inline-assembly
        assembly {
            chainId := chainid()
        }
    }
}

// SPDX-License-Identifier: MIT

pragma solidity 0.7.6;

interface IToken {
    function delegate(address delegatee) external;

    function delegateBySig(
        address delegator,
        address delegatee,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;

    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;

    function burn(uint256 amount) external;

    function mint(address recipient, uint256 amount) external;

    function getCurrentVotes(address account) external view returns (uint256);

    function getPriorVotes(address account, uint256 blockNumber) external view returns (uint256);
}

// SPDX-License-Identifier: MIT

pragma solidity 0.7.6;

interface ITimelock {}

// SPDX-License-Identifier: MIT

pragma solidity 0.7.6;

interface ISubscriber {
    function registry() external view returns (address);

    function governance() external view returns (address);

    function manager() external view returns (address);
}

// SPDX-License-Identifier: MIT

pragma solidity 0.7.6;

interface IRegistry {
    function governance() external view returns (address);

    function manager() external view returns (address);
}

// SPDX-License-Identifier: MIT

pragma solidity 0.7.6;

import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import {SafeERC20} from "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";

library TransferHelper {
    using SafeERC20 for IERC20;

    // safely transfer tokens without underflowing
    function safeTokenTransfer(
        address recipient,
        address token,
        uint256 amount
    ) internal returns (uint256) {
        if (amount == 0) {
            return 0;
        }

        uint256 balance = IERC20(token).balanceOf(address(this));
        if (balance < amount) {
            IERC20(token).safeTransfer(recipient, balance);
            return balance;
        } else {
            IERC20(token).safeTransfer(recipient, amount);
            return amount;
        }
    }
}


// SPDX-License-Identifier: MIT

pragma solidity 0.7.6;

import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import {SafeMath} from "@openzeppelin/contracts/math/SafeMath.sol";
import {ReentrancyGuard} from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
import {TransferHelper} from "./libraries/TransferHelper.sol";
import {ITimelock} from "./interfaces/ITimelock.sol";
import {IToken} from "./interfaces/IToken.sol";
import {OhSubscriber} from "./registry/OhSubscriber.sol";

/// @title Oh! Finance Token Timelock
/// @notice Contract to manage linear token vesting over a given time period
/// @notice Users accrue vested tokens as soon as the timelock starts, every second
contract OhTimelock is ReentrancyGuard, OhSubscriber, ITimelock {
    using SafeMath for uint256;

    /// @notice The total vested balance of tokens a user can claim
    mapping(address => uint256) public balances;

    /// @notice The total amount of tokens a user has already claimed
    mapping(address => uint256) public claimed;

    /// @notice The Oh! Finance Token address
    address public token;

    /// @notice The UNIX timestamp that the timelock starts at
    uint256 public timelockStart;

    /// @notice The length in seconds of the timelock
    uint256 public timelockLength;

    /// @notice Emitted when a user is added to the timelock
    event Add(address indexed user, uint256 amount);

    /// @notice Emitted every time a user claims tokens
    event Claim(address indexed user, uint256 amount);

    /// @notice Timelock constructor
    /// @param registry_ The address of the Registry
    /// @param _token The address of the Oh! Finance Token
    /// @param _timelockDelay Seconds to delay the timelock from starting
    /// @param _timelockLength The length of the timelock in seconds
    constructor(
        address registry_,
        address _token,
        uint256 _timelockDelay,
        uint256 _timelockLength
    ) OhSubscriber(registry_) {
        token = _token;
        timelockStart = block.timestamp + _timelockDelay;
        timelockLength = _timelockLength;
    }
    /// @notice Add a set of users to the vesting contract with a set amount
    /// @dev Only callable by Governance, delegates token votes to msg.sender until they are claimed
    /// @param users The array of users to be added to the vesting contract
    /// @param amounts The array of amounts of tokens to add to each users vesting schedule
    function add(address[] memory users, uint256[] memory amounts) external onlyGovernance {
        require(users.length == amounts.length, "Timelock: Arrity mismatch");

        // find total, add to user balances
        uint256 totalAmount = 0;
        uint256 length = users.length;
        for (uint256 i = 0; i < length; i++) {
            // get user and amount
            address user = users[i];
            uint256 amount = amounts[i];

            // update state and total, emit add
            balances[user] = amount;
            totalAmount = totalAmount.add(amount);
            emit Add(user, amount);
        }

        // transfer from msg.sender, delegate votes back to msg.sender
        IERC20(token).transferFrom(msg.sender, address(this), totalAmount);
    }

    /// @notice Claim all available tokens for the msg.sender, if any
    /// @dev Reentrancy guard to prevent double claims
    function claim() external nonReentrant {
        require(block.timestamp > timelockStart, "Timelock: Lock not started");

        // check for available claims
        address user = msg.sender;
        uint256 amount = claimable(user);
        require(amount > 0, "Timelock: No Tokens");

        // update user claimed variables
        claimed[user] = claimed[user].add(amount);

        // transfer to user
        TransferHelper.safeTokenTransfer(user, token, amount);
        emit Claim(user, amount);
    }

    /// @notice Available tokens available for a user to claim
    /// @dev Available = ((Balances[user] * Time_Passed) / Total_Time) - Claimed[user]
    /// @param user The user address to check
    /// @return amount The amount of tokens available to claim
    function claimable(address user) public view returns (uint256 amount) {
        // save state variable to memory
        uint256 userClaimed = claimed[user];

        // if timelock hasn't started yet
        if (block.timestamp < timelockStart) {
            // return entire balance
            amount = balances[user];
        }
        // else if timelock has expired
        else if (block.timestamp > timelockStart.add(timelockLength)) {
            // return total remaining balance
            amount = balances[user].sub(userClaimed);
        }
        // else we are currently in the vesting phase
        else {
            // find the time passed since timelock start
            uint256 delta = block.timestamp.sub(timelockStart);
            // find the total vested amount of tokens available
            uint256 totalVested = balances[user].mul(delta).div(timelockLength);

            // return vested - claimed
            amount = totalVested.sub(userClaimed);
        }
    }
}


// SPDX-License-Identifier: MIT

pragma solidity 0.7.6;

import {Address} from "@openzeppelin/contracts/utils/Address.sol";
import {IRegistry} from "../interfaces/IRegistry.sol";

/// @title Oh! Finance Registry
/// @dev Contract that contains references to the all core contracts for Oh! Finance
/// @dev Ideally, we should never need to replace this contract. Only update references.
contract OhRegistry is IRegistry {
    using Address for address;

    /// @notice address of governance contract
    address public override governance;
    /// @notice address of the management contract
    address public override manager;

    event GovernanceUpdated(address indexed oldGovernance, address indexed newGovernance);
    event ManagerUpdated(address indexed oldManager, address indexed newManager);

    modifier onlyGovernance {
        require(msg.sender == governance, "Registry: Only Governance");
        _;
    }

    constructor() {
        governance = msg.sender;
    }

    /// @notice Sets the Governance address
    /// @param _governance the new governance address
    /// @dev Only Governance can call this function
    function setGovernance(address _governance) external onlyGovernance {
        require(_governance.isContract(), "Registry: Invalid Governance");
        emit GovernanceUpdated(governance, _governance);
        governance = _governance;
    }

    /// @notice Sets the Manager address
    /// @param _manager the new manager address
    /// @dev Only Governance can call this function
    function setManager(address _manager) external onlyGovernance {
        require(_manager.isContract(), "Registry: Invalid Manager");
        emit ManagerUpdated(manager, _manager);
        manager = _manager;
    }
}

// SPDX-License-Identifier: MIT

pragma solidity 0.7.6;

import {Address} from "@openzeppelin/contracts/utils/Address.sol";
import {ISubscriber} from "../interfaces/ISubscriber.sol";
import {IRegistry} from "../interfaces/IRegistry.sol";
```

```
/// @title Oh! Finance Subscriber
/// @notice Base Oh! Finance contract used to control access throughout the protocol
abstract contract OhSubscriber is ISubscriber {
    address internal _registry;

    /// @notice Only allow authorized addresses (governance or manager) to execute a function
    modifier onlyAuthorized {
        require(msg.sender == governance() || msg.sender == manager(), "Subscriber: Only Authorized")
        _;
    }

    /// @notice Only allow the governance address to execute a function
    modifier onlyGovernance {
        require(msg.sender == governance(), "Subscriber: Only Governance");
        _;
    }

    /// @notice Construct contract with the Registry
    /// @param registry_ The address of the Registry
    constructor(address registry_) {
        require(Address.isContract(registry_), "Subscriber: Invalid Registry");
        _registry = registry_;
    }

    /// @notice Get the Governance address
    /// @return The current Governance address
    function governance() public view override returns (address) {
        return IRegistry(registry()).governance();
    }

    /// @notice Get the Manager address
    /// @return The current Manager address
    function manager() public view override returns (address) {
        return IRegistry(registry()).manager();
    }

    /// @notice Get the Registry address
    /// @return The current Registry address
    function registry() public view override returns (address) {
        return _registry;
    }

    /// @notice Set the Registry for the contract. Only callable by Governance.
    /// @param registry_ The new registry
    /// @dev Requires sender to be Governance of the new Registry to avoid bricking.
    /// @dev Ideally should not be used
    function setRegistry(address registry_) external onlyGovernance {
        require(Address.isContract(registry_), "Subscriber: Invalid Registry");

        _registry = registry_;
        require(msg.sender == governance(), "Subscriber: Bad Governance");
    }
}
```

```
{
  "name": "@ohfinance/contracts",
  "version": "1.0.0",
  "description": "Oh! Finance Ethereum Smart Contracts",
  "homepage": "https://oh.finance",
  "repository": "https://github.com/OhFinance/contracts",
  "author": "OhFinance <hello@oh.finance>",
  "license": "MIT",
  "types": "types/index.ts",
  "files": [
    "abi",
    "artifacts",
    "contracts",
    "types"
  ],
  "scripts": {
    "bump:minor": "yarn version --minor",
    "bump:major": "yarn version --major",
    "clean": "hardhat clean",
    "release": "yarn publish",
    "lint": "yarn prettier && solhint -c .solhint.json contracts/**/*.sol",
    "build": "hardhat compile && tsc",
    "compile": "hardhat compile",
    "docs:build": "rm -rf docs && yarn run hardhat docgen",
    "docs:serve": "serve -s docs",
    "dev": "hardhat node --watch",
    "test": "hardhat test",
    "test:gas": "cross-env REPORT_GAS=1 hardhat test",
    "test:fast": "cross-env TS_NODE_TRANSPILE_ONLY=1 hardhat test",
    "rinkeby:deploy": "hardhat --network rinkeby deploy",
    "rinkeby:run": "hardhat --network rinkeby run",
    "rinkeby:sourcify": "hardhat --network rinkeby sourcify",
    "rinkeby:verify": "hardhat --network rinkeby etherscan-verify",
    "mainnet:deploy": "hardhat --network mainnet deploy",
    "mainnet:run": "hardhat --network mainnet run",
    "mainnet:sourcify": "hardhat --network mainnet sourcify",
    "mainnet:verify": "hardhat --network mainnet etherscan-verify"
  },
  "devDependencies": {
    "@digix/doxity": "^0.5.2",
    "@ethereum-waffle/chai": "^3.2.2",
    "@nomiclabs/hardhat-ethers": "npm:hardhat-deploy-ethers",
    "@nomiclabs/hardhat-etherscan": "^2.1.2",
    "@nomiclabs/hardhat-waffle": "^2.0.1",
    "@openzeppelin/contracts": "3.4.1",
    "@openzeppelin/contracts-upgradeable": "3.4.1",
    "@studydefi/money-legos": "^2.4.1",
    "@typechain/ethers-v5": "^7.0.0",
    "@typechain/hardhat": "^2.0.1",
    "@types/chai": "^4.2.14",
    "@types/mocha": "^8.2.0",
    "@types/node": "^14.14.22",
    "@uniswap/v2-core": "^1.0.1",
    "@uniswap/v2-periphery": "^1.1.0-beta.0",
    "chai": "^4.2.0",
    "cross-env": "^7.0.3",
    "dotenv": "^8.2.0",
    "ethereum-waffle": "^3.3.0",
    "ethers": "^5.3.0",
    "hardhat": "^2.0.6",
    "hardhat-abi-exporter": "^2.0.7",
    "hardhat-deploy": "^0.7.10",
    "hardhat-docgen": "^1.1.1",
    "hardhat-gas-reporter": "^1.0.4",
    "hardhat-spdx-license-identifier": "^2.0.3",
    "hardhat-typechain": "^0.3.4",
    "prettier": "^2.2.1",
    "prettier-plugin-solidity": "^1.0.0-beta.9",
    "solc": "0.7.6",
    "ts-generator": "^0.1.1",
    "ts-node": "^9.1.1",
    "tsconfig-paths": "^3.9.0",
    "typechain": "^5.0.0",
    "typescript": "^4.1.3"
  }
```

```
}
```

Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already

deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Unsolved TODO comments

- **Description:**
Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Short Address/Parameter Attack

- **Description:**
This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Unchecked CALL Return Values

- **Description:**
There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Permission restrictions

- **Description:**
  Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other

users.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

armors.io

contact@armors.io