Armors Labs

Autumn

Smart Contract Audit

- Autumn Audit Summary
- Autumn Audit
    - Document information
        - Audit results
        - Audited target file
    - Vulnerability analysis
        - Vulnerability distribution
        - Summary of audit results
        - Contract Code
        - Analysis of audit results
            - Re-Entrancy
            - Arithmetic Over/Under Flows
            - Unexpected Ether
            - Delegatecall
            - Default Visibilities
            - Entropy Illusion
            - External Contract Referencing
            - Unsolved TODO comments
            - Short Address/Parameter Attack
            - Unchecked CALL Return Values
            - Race Conditions / Front Running
            - Denial Of Service (DOS)
            - Block Timestamp Manipulation
            - Constructors with Care
            - Unintialised Storage Pointers
            - Floating Points and Numerical Precision
            - tx.origin Authentication

Autumn Audit Summary

Project name : Autumn Contract

Project address: None

Code URL : https://github.com/autumn-finance/aVault.git

Commit: 6059dd29e7421b32ab45da3ef2262ee5ec7ce036

Projct target : Autumn Contract Audit

Test result : PASSED

Audit Info

Audit NO : 0X202103170015

Audit Team : Armors Labs

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

Autumn Audit

The Autumn team asked us to review and audit their Autumn 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
Autumn Audit	Rock ,Hosea, Rushairer	1.0.0	2021-03-14

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Autumn 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 18 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
StrategyV1.sol	aedefcabe031b8effb80be0aaa3c8e6c
aVault.sol	d30430a7b622cbc0f2818f148807bac7
PermissionManager.sol	37b2ddc354fa96d457f27b5f36b12b95
FeeManager.sol	ebeac3026dd2be58a0067af803928124

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 Ether	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
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

Vulnerability	status
tx.origin Authentication	safe

Contract Code

aVault.sol

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

/*
 * This file is part of the Autumn aVault Smart Contract
 */

pragma solidity ^0.7.3;
pragma experimental ABIEncoderV2;

import { FeeManager, PermissionManager } from "./vendor/FeeManager.sol";
import { ERC20, IERC20, SafeMath } from "./vendor/ERC20.sol";
import { SafeERC20Wrapper, SafeERC20 } from "./vendor/SafeERC20Wrapper.sol";
import { IStrategy, IVault } from "./interface/IStrategy.sol";
import { Rescuable } from "./vendor/Rescuable.sol";
import { IWHT } from "./vendor/IWHT.sol";

/**
 * @dev The user interface for deposit and earn by the strategy
 */
contract aVault is IVault, ERC20, FeeManager, Rescuable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    using SafeERC20Wrapper for IERC20;

    bool public override constant IS_VAULT = true;
    IWHT private constant WHT = IWHT(0x5545153CCFcA01fbd7Dd11C0b23ba694D9509A6F);

    /**
     * @dev The underlying asset of vault
     */
    IERC20 private immutable _asset;

    bool public override immutable native;

    constructor (
        address underlyingAsset,
        string memory aAssetName, string memory aAssetSymbol, uint8 aAssetDecimals,
        bool nativeCurrency
    ) ERC20(
        aAssetName, aAssetSymbol, aAssetDecimals
    ) {
        native = nativeCurrency;
        try ERC20(underlyingAsset).decimals() {
            require(
                ERC20(underlyingAsset).decimals() == aAssetDecimals,
                "aVault@constructor: inconsistent asset decimals");
        } catch {}

        require(
            !nativeCurrency || underlyingAsset == address(WHT),
            "aVault@constructor: asset should be WHT for native currency"
        );
        _asset = IERC20(underlyingAsset);
    }
```

```
function balanceOf() public view returns (uint256) {
    return _asset.balanceOf(address(this));
}

// Yield

uint256 public shareToAssetRecord;
uint256 public interestInitialTime;

uint256 private yieldRecord;

/// @dev should be called only when balance is updated to the actual (latest) value
function updateAssetYield() private {
    if (totalSupply() == 0) { // initialize
        shareToAssetRecord = 1e18;
        interestInitialTime = block.timestamp;
        return;
    }

    uint256 shareToAsset_ = shareToAsset(1e18);
    if (shareToAsset_ > shareToAssetRecord) {
        // shareToAsset will only increase by time in the scenario
        shareToAssetRecord = shareToAsset_;
        yieldRecord = getSecondYieldBy(shareToAsset_);
    }
}

function getSecondYieldBy(uint256 shareToAsset_) public view returns (uint256) {
    uint256 past = block.timestamp.sub(interestInitialTime);
    return shareToAsset_.sub(1e18).div(past);
}

/**
 * @notice Returns the asset yield per second in the share of 1e18
 */
function getSecondYield() external view returns (uint256) {
    // directly use the recorded value
    // in case yield drop by time with an outdated balance
    return yieldRecord;
}

// Strategy

IStrategy public strategy;

function setStrategy(address strategy_, bool update_) onlyOwner() external {
    require(strategy_ != address(0), "aVault@setStrategy: invalid address");
    require(strategy_ != address(strategy), "aVault@setStrategy: already set");

    IStrategy newStrategy = IStrategy(strategy_);
    require(newStrategy.IS_STRATEGY(), "aVault@setStrategy: not a strategy");
    require(newStrategy.asset() == address(_asset), "aVault@setStrategy: variant asset");

    if (address(strategy) != address(0)) {
        if (update_) {
            strategy.update();
        }
        strategy.withdrawAll();
    }

    strategy = newStrategy;
    if (balanceOf() != 0) {
        _approveAndDeposit(balanceOf());
    }
}
```

```
    modifier requireStrategy() {
        require(address(strategy) != address(0), "aVault@requireStrategy: no strategy");
        _;
    }

    // Share

    /**
     * @dev Share value formula:
     *
     * share per token = total share / total supply
     * share = token * share per token
     *
     * token per share = total supply / total share
     * token = share * token per share
     */

    function assetToShare(uint256 amount) external view returns (uint256) {
        return assetToShareBy(balance(), amount);
    }

    function assetToShareBy(uint256 balance_, uint256 amount) public view returns (uint256) {
        if (totalSupply() == 0) {
            return amount;
        }
        return (amount.mul(totalSupply())).div(balance_);
    }

    function shareToAsset(uint256 share) public view returns (uint256) {
        require(totalSupply() != 0, "aVault@shareToAsset: no share minted");
        return (share.mul(balance())).div(totalSupply());
    }

    // Vault Interface

    /**
     * @dev Returns the underlying asset of vault.
     */
    function asset() external override view returns (address) {
        return address(_asset);
    }

    /**
     * @dev Returns the vault balance of underlying asset.
     */
    function balance() requireStrategy() public override view returns (uint256) {
        return strategy.balance().add(balanceOf());
    }

    /**
     * @dev Returns the asset value of vault in stablecoin.
     */
    function value() requireStrategy() external override view returns (uint256) {
        return valueBy(balance());
    }

    function valueBy(uint256 amount) requireStrategy() public override view returns (uint256) {
        return strategy.valueBy(amount); // delegate to strategy
    }

    /**
     * @dev Carry out an update to vault.
     */
    function update() requireStrategy() public override {
        // restrict to off-chain in case price manipulation
        require(_msgSender() == tx.origin, "aVault@update: only callable from off-chain");
        _update();
    }

    function _update() private {
        strategy.update();
        updateAssetYield();
    }

    /**
     * @dev Deposit asset to vault. Returns the share amount.
     */
    function deposit(uint256 amount) requireStrategy() external override payable returns (uint256) {
        require(amount != 0, "aVault@deposit: cannot deposit zero");
        uint256 before = balance(); // use 'current' state to mint share
        if (before == 0) {
            updateAssetYield();
        } else {
            _update();
        }

        if (native) {
            require(msg.value == amount, "aVault@deposit: deposit was inconsistent with amount");
            WHT.deposit{ value: msg.value }();
        } else {
            require(msg.value == 0, "aVault@deposit: deposit was incorrect");
            amount = _asset.safeReceive(_msgSender(), amount);
        }

        amount = _approveAndDeposit(amount);
        uint256 share = assetToShareBy(before, amount);
        _mint(_msgSender(), share);

        if (before == 0) {
            _update();
        }
        return share;
    }

    function _approveAndDeposit(uint256 amount) private returns (uint256) {
        _asset.safeIncreaseAllowance(address(strategy), amount);
        return strategy.deposit(amount);
    }

    /**
     * @dev Withdraw share from vault. Returns the asset amount.
     */
    function withdraw(uint256 share) external override returns (uint256) {
        _update();
        return _withdraw(share);
    }

    /// @dev alternative in case update fails
    function _withdraw(uint256 share) requireStrategy() public returns (uint256) {
        require(share != 0, "aVault@withdraw: cannot withdraw zero");

        uint256 amount = shareToAsset(share);
        _burn(_msgSender(), share);
        require(amount != 0, "aVault@withdraw: withdraw amount is too small");

        uint256 balance_ = balanceOf();
        if (balance_ < amount) {
            uint256 expection = amount.sub(balance_);
            uint256 withdrawn = strategy.withdraw(expection);
            if (withdrawn < expection) {
                amount = balance_.add(withdrawn);
            }
        }

        amount = FeeManager.chargeFeeWith("withdraw", address(_asset), amount);

        if (native) {
            WHT.withdraw(amount);
            _msgSender().transfer(amount);
        } else {
            _asset.safeTransfer(_msgSender(), amount);
        }

        return amount;
    }

    receive() payable external {}
}
```

StrategyV1.sol

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

/*
 * This file is part of the Autumn aVault Smart Contract
 */

pragma solidity ^0.7.3;
pragma experimental ABIEncoderV2;

import { IProvider } from "./interface/IProvider.sol";
import { IVault, IStrategy } from "./interface/IStrategy.sol";
import { PermissionManager } from "./vendor/PermissionManager.sol";
import { IMdexRouter, IMdexFactory, IMdexPair } from "./vendor/IMdex.sol";
import { SafeERC20Wrapper, SafeERC20, SafeMath, IERC20 } from "./vendor/SafeERC20Wrapper.sol";

/**
 * @title Strategy for the vaults to earn by multiple providers
 */
contract StrategyV1 is IStrategy, PermissionManager {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    using SafeERC20Wrapper for IERC20;

    bool public override constant IS_VAULT = true;
    bool public override constant IS_STRATEGY = true;

    IMdexRouter private constant MDEX = IMdexRouter(0xED7d5F38C79115ca12fe6C0041abb22F0A06C300);

    /**
     * @dev The underlying asset of strategy
     */
    IERC20 private immutable _asset;

    address[] public valuePath;

    constructor(address vault, address[] memory valuePath_) {
        _asset = IERC20(IVault(vault).asset());
        valuePath = valuePath_;

        // Prepare permissions
        PermissionManager.addRole("governor", 0);
        PermissionManager.setPermission("governor", _msgSender(), true);
        PermissionManager.addRole("strategist", 0);
        PermissionManager.setPermission("strategist", _msgSender(), true);
        PermissionManager.setSingletonPermission("vault", vault);
    }

    /// @dev updates the path of swapping to stablecoin, only used for stats
    function setValuePath(address[] memory valuePath_) onlyOwner() external {
        valuePath = valuePath_;
    }

    /*
     * Strategy Interface
     */

    // Viewers

    function native() external override view returns (bool) {
        return provider().native();
    }

    function asset() external override view returns (address) {
        return address(_asset);
    }

    function balance() public override view returns (uint256 sum) {
        for (uint256 i = 0; i < providerNames.length; i++) {
            sum = sum.add(getProvider(i).balance());
        }
        return balanceOf().add(sum);
    }

    /**
     * @dev Returns the asset value of vault in stablecoin.
     */
    function value() external override view returns (uint256 sum) {
        return valueBy(balance());
    }

    function valueProvider(string memory name) external view returns (uint256) {
        return valueBy(providers[name].balance());
    }

    /// @dev simulates a swap in Mdex without applying fee
    function valueBy(uint256 assetAmount) public override view returns (uint256) {
        if (valuePath.length <= 1) { // for stablecoin
            return assetAmount;
        }
        IMdexFactory factory = IMdexFactory(MDEX.factory());
        uint256 lastPathAmountOut = assetAmount;
        for (uint256 i = 0; i < valuePath.length; i++) {
            if (i == valuePath.length - 1) {
                break;
            }
            IMdexPair pair = IMdexPair(factory.pairFor(valuePath[i], valuePath[i + 1]));
            (uint112 reserve0, uint112 reserve1,) = pair.getReserves();
            lastPathAmountOut = MDEX.quote(lastPathAmountOut, reserve0, reserve1);
        }
        return lastPathAmountOut;
    }

    // Externals

    function update() requirePermission("vault") external override {
        for (uint256 i = 0; i < providerNames.length; i++) {
            getProvider(i).update();
        }
    }
```

```
function deposit(uint256 amount) requirePermission("vault") external override payable returns (ui
    require(amount != 0, "StrategyV1@deposit: cannot deposit zero");

    amount = _asset.safeReceive(_msgSender(), amount);
    return _approveAndDeposit(amount);
}

function _approveAndDeposit(uint256 amount) private returns (uint256) {
    IProvider p = provider();
    _asset.safeApprove(address(p), amount);
    return p.deposit(amount);
}

function withdraw(uint256 amount) requirePermission("vault") public override returns (uint256) {
    require(amount != 0, "StrategyV1@withdraw: cannot withdraw zero");

    uint256 balance_ = balanceOf();
    if (balance_ < amount) {
        uint256 expection = amount.sub(balance_);
        uint256 withdrawn = withdrawFromProviders(expection);
        if (withdrawn < expection) {
            amount = balance_.add(withdrawn);
        }
    }

    _asset.safeTransfer(_msgSender(), amount);
    return amount;
}

function withdrawFromProviders(uint256 amount) private returns (uint256 withdrawn) {
    IProvider p = provider();
    if (amount < p.balance()) {
        return p.withdraw(amount);
    }

    uint256 remaining = amount;
    for (uint256 i = 0; i < providerNames.length; i++) {
        p = getProvider(i);

        if (remaining < p.balance()) {
            return withdrawn.add(p.withdraw(remaining));
        } else {
            remaining = remaining.sub(p.balance());
            withdrawn = withdrawn.add(p.withdrawAll());
        }
    }
}

function withdrawAll() requirePermission("vault") external override returns (uint256) {
    uint256 balance_ = balance();
    return balance_ == 0 ? 0 : withdraw(balance());
}

/*
 * Provider
 */

/**
 * @notice All providers in strategy by their names
 */
mapping (string => IProvider) public providers;

/**
 * @notice Array of all provider names
 */
```

```
    string[] public providerNames;

    /**
     * @notice The default provider for deposit
     */
    IProvider public defaultProvider;

    /**
     * @notice List names of all providers
     */
    function listProviders() external view returns (string[] memory) {
        return providerNames;
    }

    /**
     * @notice Suggest a provider by strategy
     */
    function provider() public view returns (IProvider) {
        if (address(defaultProvider) != address(0)) {
            return defaultProvider;
        } else {
            require(providerNames.length != 0, "StrategyV1@provider: no provider yet");
            return getProvider(0);
        }
    }

    /**
     * @notice Get a provider instance by its index
     */
    function getProvider(uint256 index) public view returns (IProvider) {
        require(providerNames.length > index, "StrategyV1@getProvider: provider at index not exists")
        return providers[providerNames[index]];
    }

    /**
     * @notice Query the balance of a provider by its name
     */
    function getProviderBalance(string memory name) public view returns (uint256) {
        return providers[name].balance();
    }

    /**
     * @notice Search for the index of a provider by its name
     */
    function searchProviderIndex(string memory name) public view returns (bool found, uint256 at) {
        for (uint256 i = 0; i < providerNames.length; i++) {
            if (compareString(providerNames[i], name)) {
                return (true, i);
            }
        }
        return (false, 0);
    }

    // Governance functions

    /**
     * @notice Add or replace a provider to strategy
     */
    function addProvider(string memory name, address provider_) requirePermission("governor") externa
        IProvider p = IProvider(provider_);
        require(p.IS_PROVIDER(), "StrategyV1@addProvider: not a provider");
        require(p.asset() == address(_asset), "StrategyV1@addProvider: variant asset");

        (bool exists, uint256 index) = searchProviderIndex(name);
        if (exists) {
            removeProviderByIndex(index, true);
        }

        providers[name] = p;
        providerNames.push(name);
        emit ProviderAdded(name, provider_);
    }

    /**
     * @notice Withdraw all from and remove a provider by its name
     */
    function removeProvider(string memory name, bool withdraw_) requirePermission("governor") externa
        (bool has, uint256 index) = searchProviderIndex(name);
        require(has, "StrategyV1@removeProvider: provider not exists");
        removeProviderByIndex(index, withdraw_);
    }

    function removeProviderByIndex(uint256 index, bool withdraw_) private {
        string memory name = providerNames[index];
        if (withdraw_) {
            _adjustWithdraw(name, 0);
        }
        emit ProviderRemoved(name, deleteProviderData(name, index));
    }

    /**
     * @dev Delete a provider permanently from storage
     */
    function deleteProviderData(string memory name, uint256 index) private returns (address) {
        address provider_ = address(providers[name]);
        if (address(defaultProvider) == provider_) {
            defaultProvider = IProvider(address(0));
        }
        providerNames[index] = providerNames[providerNames.length - 1];
        providerNames.pop();
        delete providers[name];
        return provider_;
    }

    event ProviderAdded(string indexed providerName, address indexed providerAddress);
    event ProviderRemoved(string indexed providerName, address indexed providerAddress);

    // Strategist functions

    /**
     * @notice Set the default provider for strategy
     */
    function setDefaultProvider(string memory name) requirePermission("strategist") external {
        IProvider p = providers[name];
        require(p.IS_PROVIDER(), "StrategyV1@setDefaultProvider: provider not exists");
        defaultProvider = p;
        emit DefaultProviderUpdated(name, address(p));
    }

    /**
     * @notice Adjust strategy by deposit balance to a provider
     *  Use `0` for deposit all available balance
     */
    function adjustDeposit(string memory name, uint256 amount) requirePermission("strategist") public
        IProvider p = providers[name];
        require(p.IS_PROVIDER(), "StrategyV1@adjustDeposit: provider not exists");

        uint256 balance_ = balanceOf();
        amount = amount == 0 ? balance_ : amount;
        require(balance_ >= amount, "StrategyV1@adjustDeposit: amount exceeds balance");
        amount = _approveAndDeposit(amount);
        emit StrategyAdjusted(_msgSender(), address(p), 0, amount);
        return amount;
    }

    /**
     * @notice Adjust strategy by withdraw balance from a provider
     *  Use `0` for withdraw all provider balance
     */
    function adjustWithdraw(string memory name, uint256 amount) requirePermission("strategist") publi
        return _adjustWithdraw(name, amount);
    }

    /// @dev perform permission check before calling this
    function _adjustWithdraw(string memory name, uint256 amount) private returns (uint256) {
        IProvider p = providers[name];
        require(p.IS_PROVIDER(), "StrategyV1@adjustWithdraw: provider not exists");

        amount = amount == 0 ? p.withdrawAll() : p.withdraw(amount);
        emit StrategyAdjusted(_msgSender(), address(p), 1, amount);
        return amount;
    }

    /**
     * @notice Adjust strategy by move balance from one provider to another
     *  Use `0` for move all provider balance
     */
    function adjustMove(string memory from, string memory to, uint256 amount) requirePermission("stra
        return adjustDeposit(to, adjustWithdraw(from, amount));
    }

    event DefaultProviderUpdated(string indexed providerName, address indexed providerAddress);
    event StrategyAdjusted(address indexed strategist, address indexed provider, uint256 opcode, uint

    // Misc

    /**
     * @notice Returns asset balance of current contract
     */
    function balanceOf() public view returns (uint256) {
        return _asset.balanceOf(address(this));
    }

    /**
     * @dev Compares the equity of two strings
     */
    function compareString(string memory a, string memory b) private pure returns (bool) {
        return keccak256(abi.encodePacked(a)) == keccak256(abi.encodePacked(b));
    }
}
```

PermissionManager.sol

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

/*
 * This file is part of the Autumn aVault Smart Contract
 */

pragma solidity ^0.7.3;
pragma experimental ABIEncoderV2;

import "./Ownable.sol";
```

```
/**
 * @title Manage and restrict permission by roles
 */
contract PermissionManager is Ownable {

    // Roles

    struct RoleInfo {
        bool exists;
        uint256 limit;
        uint256 index;
    }

    mapping (string => RoleInfo) roles;

    string[] public roleList;

    function listRoles() external view returns (string[] memory) {
        return roleList;
    }

    function addRole(string memory role, uint256 limit) onlyOwner() public {
        require(!roles[role].exists, "PermissionManager@addRole: already added");
        roles[role] = RoleInfo(true, limit, roleList.length);
        roleList.push(role);

        emit RoleAdded(role);
    }

    function removeRole(string memory role) onlyOwner() public {
        require(roles[role].exists, "PermissionManager@addRole: role not added");
        uint256 atArray = roles[role].index;
        roleList[atArray] = roleList[roleList.length - 1];
        roleList.pop();
        delete roles[role];

        emit RoleRemoved(role);
    }

    event RoleAdded(string indexed role);
    event RoleRemoved(string indexed role);

    // Permissions

    struct PermissonInfo {
        bool has;
        uint256 index;
    }

    mapping (string => mapping (address => PermissonInfo)) permissions;

    mapping (string => address[]) permissionList;

    function listPermissions(string memory role) external view returns (address[] memory) {
        return permissionList[role];
    }

    modifier requirePermission(string memory role) {
        require(
            permissions[role][_msgSender()].has,
            string(abi.encodePacked("PermissionManager@requirePermission: no permission ", role)));
        _;
    }

    function setPermission(string memory role, address user, bool permit) onlyOwner() public {
        RoleInfo memory roleInfo = roles[role];
        require(roleInfo.exists, "PermissionManager@setPermission: role not exists");
        require(permissions[role][user].has != permit, "PermissionManager@setPermission: already set"

        uint256 permissionListSize = permissionList[role].length;

        if (permit) {
            require(roleInfo.limit == 0 || roleInfo.limit > permissionListSize, "PermissionManager@se
            permissions[role][user] = PermissonInfo(true, permissionListSize);
            permissionList[role].push(user);

        } else {
            uint256 atList = permissions[role][user].index;
            permissionList[role][atList] = permissionList[role][permissionListSize - 1];
            permissionList[role].pop();
            delete permissions[role][user];
        }

        emit PermissionUpdated(role, user, permit);
    }

    event PermissionUpdated(string indexed role, address indexed user, bool permission);

    // Wrappers

    function setSingletonPermission(string memory role, address user) onlyOwner() public {
        PermissionManager.addRole(role, 1);
        PermissionManager.setPermission(role, user, true);
    }
}
```

FeeManager.sol

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

/*
 * This file is part of the Autumn aVault Smart Contract
 */

pragma solidity ^0.7.3;
pragma experimental ABIEncoderV2;

import { SafeERC20, SafeMath, IERC20 } from "./SafeERC20.sol";
import { PermissionManager } from "./PermissionManager.sol";

abstract contract FeeManager is PermissionManager {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    uint256 private constant FEE_DENOMINATOR = 1e8;

    mapping (string => uint256) public fees;

    address public feeRecipient;

    constructor() {
        feeRecipient = _msgSender();
        PermissionManager.addRole("fee", 0);
    }

    /**
     * @dev Charges fee on the amount and allocated for the associated token (and transfer if exceeds
     */
    function chargeFeeWith(string memory name, address token, uint256 amount) internal returns (uint2
        uint256 fee = fees[name];
        if (fee == 0) {
            return amount;
        }

        uint256 fee_ = amount.mul(fee).div(FEE_DENOMINATOR);
        IERC20(token).safeTransfer(feeRecipient, fee_);
        return amount.sub(fee_);
    }

    // Governance

    function setFeeFor(string memory name, uint256 fee) requirePermission("fee") external {
        fees[name] = fee;
    }

    function setFeeRecipient(address recipient) onlyOwner() external {
        feeRecipient = recipient;
    }
}
```

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 ether, and as such often send ether to various external user addresses. The operation of calling external contracts, or sending ether 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 Ether

- **Description:**
  Typically when ether 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 ether can exist in a contract without having executed any code. Contracts which rely on code execution for every ether sent to the contract can be vulnerable to attacks where ether 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 whether 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 vulnerabilities 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 ether 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 ether 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.

armors.io

contact@armors.io