Armors Labs

ORIGIN - Treasury

Smart Contract Audit

- ORIGIN - Treasury Audit Summary
- ORIGIN - Treasury Audit
  - Document information
    - Audit results
    - Audited target file
  - Vulnerability analysis
    - Vulnerability distribution
    - Summary of audit results
    - Contract file
    - Analysis of audit results
      - Re-Entrancy
      - Arithmetic Over/Under Flows
      - Unexpected Blockchain Currency
      - 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
      - Permission restrictions

Armors Labs

ORIGIN - Treasury Audit Summary

Project name : ORIGIN - Treasury Contract

Project address: None

Code URL : https://github.com/OriginBank/origin-core/blob/main/Treasury.sol

Commit : 2ad67778291787c5ec9c73347446066eb6e2150a

Project target : ORIGIN - Treasury Contract Audit

Blockchain : Polygon

Test result : PASSED

Audit Info

Audit NO : 0X202403040006

Audit Team : Armors Labs

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

ORIGIN - Treasury Audit

The ORIGIN - Treasury team asked us to review and audit their ORIGIN - Treasury 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
ORIGIN - Treasury Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2024-03-04

Audit results

WARNING:

- This audit only covers the Treasury.sol contract; other contracts are not included, and contracts related to interfaces within the contract are not within the scope of this audit.
- The owner can manage the curve variable address; the owner can call the swapReserve interface to execute token transactions (the administrator has the authority to misappropriate contract tokens).
- deposit (sender stakes, receiving a certain amount of OHM set by the administrator)
    - It is determined that the quantity of exchanged tokens is obtained 1:1 by the administrator-controlled isReserveToken or by the valuation method of the IBondCalculator interface.
    - The administrator can manage the ERC20 tokens for deposit and who can stake.

Armors Labs

- withdraw (sender burns OHM for withdrawal)
  - It is determined that the quantity of exchanged tokens is obtained 1:1 by the administrator-controlled isReserveToken or by the valuation method of the IBondCalculator interface.
  - The administrator can manage the ERC20 tokens available for withdrawal and who can withdraw.
- incurDebt (sender borrows the token set by the administrator by holding sOHM)
  - It is determined that the quantity of exchanged tokens is obtained 1:1 by the administrator-controlled isReserveToken or by the valuation method of the IBondCalculator interface.
  - The administrator can manage the borrowed ERC20 tokens and who can borrow.
  - The amount of sOHM held is the maximum borrowing limit.
- repayDebtWithReserve (sender repays the token set by the administrator)
  - It is determined that the quantity of exchanged tokens is obtained 1:1 by the administrator-controlled isReserveToken or by the valuation method of the IBondCalculator interface.
  - The administrator can manage the borrowed ERC20 tokens and who can repay them.
- repayDebtWithOHM (sender repays the token set by the administrator by burning OHM)
- manage - The administrator can withdraw assets in the token set by the administrator.
- mintRewards - The administrator can reward OHM to specified addresses.
- auditReserves - The administrator can inventory the total amount of liquidityTokens and reserveTokens.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the ORIGIN - Treasury 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.

Disclaimer

Armors Labs Reports is not and should not be regarded as an "approval" or "disapproval" of any particular project or team. These reports are not and should not be regarded as indicators of the economy or value of any "product" or "asset" created by any team. Armors do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'…)

Armors Labs Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Armors does not guarantee the safety or functionality of the technology agreed to be analyzed.

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. Armors Labs Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Audited target file

file	md5
Treasury.sol	9afc2c315ece0bbd6c32d20794ecadd3

Armors Labs

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
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

Armors Labs

Contract file

```
// SPDX-License-Identifier: AGPL-3.0-or-later
pragma solidity 0.7.5;

library SafeMath {
```

```
  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");

    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    return sub(a, b, "SafeMath: subtraction overflow");
  }

  function sub(
    uint256 a,
    uint256 b,
    string memory errorMessage
  ) internal pure returns (uint256) {
    require(b <= a, errorMessage);
    uint256 c = a - b;

    return c;
  }

  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }

    uint256 c = a * b;
    require(c / a == b, "SafeMath: multiplication overflow");

    return c;
  }

  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    return div(a, b, "SafeMath: division by zero");
  }

  function div(
    uint256 a,
    uint256 b,
    string memory errorMessage
  ) internal pure returns (uint256) {
    require(b > 0, errorMessage);
    uint256 c = a / b;
    return c;
  }
}

library Address {
  function isContract(address account) internal view returns (bool) {
    // This method relies in extcodesize, which returns 0 for contracts in
    // construction, since the code is only stored at the end of the
    // constructor execution.

    uint256 size;
    // solhint-disable-next-line no-inline-assembly
    assembly {
      size := extcodesize(account)
    }
    return size > 0;
  }

  function functionCall(
    address target,
    bytes memory data,
    string memory errorMessage
```

Armors Labs

```
  ) internal returns (bytes memory) {
    return _functionCallWithValue(target, data, 0, errorMessage);
  }

  function _functionCallWithValue(
    address target,
    bytes memory data,
    uint256 weiValue,
    string memory errorMessage
  ) private returns (bytes memory) {
    require(isContract(target), "Address: call to non-contract");

    // solhint-disable-next-line avoid-low-level-calls
    (bool success, bytes memory returndata) = target.call{value: weiValue}(
      data
    );
    if (success) {
      return returndata;
    } else {
      if (returndata.length > 0) {
        // solhint-disable-next-line no-inline-assembly
        assembly {
          let returndata_size := mload(returndata)
          revert(add(32, returndata), returndata_size)
        }
      } else {
        revert(errorMessage);
      }
    }
  }

  function _verifyCallResult(
    bool success,
    bytes memory returndata,
    string memory errorMessage
  ) private pure returns (bytes memory) {
    if (success) {
      return returndata;
    } else {
      if (returndata.length > 0) {
        // solhint-disable-next-line no-inline-assembly
        assembly {
          let returndata_size := mload(returndata)
          revert(add(32, returndata), returndata_size)
        }
      } else {
        revert(errorMessage);
      }
    }
  }
}

interface ICurve {
  function exchange_underlying(
    uint256 i,
    uint256 j,
    uint256 dx,
    uint256 min_dy
  ) external;

  function exchange(uint256 i, uint256 j, uint256 dx, uint256 min_dy) external;

  function underlying_coins(uint256 arg0) external view returns (address);
}

interface IOwnable {
```

Armors Labs

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

  function renounceManagement() external;

  function pushManagement(address newOwner_) external;

  function pullManagement() external;
}

contract Ownable is IOwnable {
  address internal _owner;
  address internal _newOwner;

  event OwnershipPushed(
    address indexed previousOwner,
    address indexed newOwner
  );
  event OwnershipPulled(
    address indexed previousOwner,
    address indexed newOwner
  );

  constructor() {
    _owner = msg.sender;
    emit OwnershipPushed(address(0), _owner);
  }

  function manager() public view override returns (address) {
    return _owner;
  }

  modifier onlyManager() {
    require(_owner == msg.sender, "Ownable: caller is not the owner");
    _;
  }

  function renounceManagement() public virtual override onlyManager {
    emit OwnershipPushed(_owner, address(0));
    _owner = address(0);
  }

  function pushManagement(
    address newOwner_
  ) public virtual override onlyManager {
    require(newOwner_ != address(0), "Ownable: new owner is the zero address");
    emit OwnershipPushed(_owner, newOwner_);
    _newOwner = newOwner_;
  }

  function pullManagement() public virtual override {
    require(msg.sender == _newOwner, "Ownable: must be new owner to pull");
    emit OwnershipPulled(_owner, _newOwner);
    _owner = _newOwner;
  }
}

interface ISwapV2Router {
  function swapExactTokensForTokensSupportingFeeOnTransferTokens(
    uint256 amountIn,
    uint256 amountOutMin,
    address[] calldata path,
    address to,
    uint256 deadline
  ) external;
}
```

Armors Labs

```
interface IERC20 {
  function decimals() external view returns (uint8);

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

  function transfer(address recipient, uint256 amount) external returns (bool);

  function approve(address spender, uint256 amount) external returns (bool);

  function totalSupply() external view returns (uint256);

  function transferFrom(
    address sender,
    address recipient,
    uint256 amount
  ) external returns (bool);

  event Transfer(address indexed from, address indexed to, uint256 value);

  event Approval(address indexed owner, address indexed spender, uint256 value);
}

library SafeERC20 {
  using SafeMath for uint256;
  using Address for address;

  function safeTransfer(IERC20 token, address to, uint256 value) internal {
    _callOptionalReturn(
      token,
      abi.encodeWithSelector(token.transfer.selector, to, value)
    );
  }

  function safeTransferFrom(
    IERC20 token,
    address from,
    address to,
    uint256 value
  ) internal {
    _callOptionalReturn(
      token,
      abi.encodeWithSelector(token.transferFrom.selector, from, to, value)
    );
  }

  function _callOptionalReturn(IERC20 token, bytes memory data) private {
    bytes memory returndata = address(token).functionCall(
      data,
      "SafeERC20: low-level call failed"
    );
    if (returndata.length > 0) {
      // Return data is optional
      // solhint-disable-next-line max-line-length
      require(
        abi.decode(returndata, (bool)),
        "SafeERC20: ERC20 operation did not succeed"
      );
    }
  }
}

interface IERC20Mintable {
  function mint(uint256 amount_) external;

  function mint(address account_, uint256 ammount_) external;
}
```

Armors Labs

```
interface IOHMERC20 {
  function burnFrom(address account_, uint256 amount_) external;
}

interface IBondCalculator {
  function valuation(
    address pair_,
    uint amount_
  ) external view returns (uint _value);
}

contract Treasury is Ownable {
  using SafeMath for uint;
  using SafeERC20 for IERC20;

  event Deposit(address indexed token, uint amount, uint value);
  event Withdrawal(address indexed token, uint amount, uint value);
  event CreateDebt(
    address indexed debtor,
    address indexed token,
    uint amount,
    uint value
  );
  event RepayDebt(
    address indexed debtor,
    address indexed token,
    uint amount,
    uint value
  );
  event ReservesManaged(address indexed token, uint amount);
  event ReservesUpdated(uint indexed totalReserves);
  event ReservesAudited(uint indexed totalReserves);
  event RewardsMinted(
    address indexed caller,
    address indexed recipient,
    uint amount
  );
  event ChangeQueued(MANAGING indexed managing, address queued);
  event ChangeActivated(
    MANAGING indexed managing,
    address activated,
    bool result
  );

  enum MANAGING {
    RESERVEDEPOSITOR,
    RESERVESPENDER,
    RESERVETOKEN,
    RESERVEMANAGER,
    LIQUIDITYDEPOSITOR,
    LIQUIDITYTOKEN,
    LIQUIDITYMANAGER,
    DEBTOR,
    REWARDMANAGER,
    SOHM
  }

  address public immutable OHM;
  uint public immutable blocksNeededForQueue;

  address[] public reserveTokens; // Push only, beware false-positives.
  mapping(address => bool) public isReserveToken;
  mapping(address => uint) public reserveTokenQueue; // Delays changes to mapping.

  address[] public reserveDepositors; // Push only, beware false-positives. Only for viewing.
```

Armors Labs

```
mapping(address => bool) public isReserveDepositor;
mapping(address => uint) public reserveDepositorQueue; // Delays changes to mapping.

address[] public reserveSpenders; // Push only, beware false-positives. Only for viewing.
mapping(address => bool) public isReserveSpender;
mapping(address => uint) public reserveSpenderQueue; // Delays changes to mapping.

address[] public liquidityTokens; // Push only, beware false-positives.
mapping(address => bool) public isLiquidityToken;
mapping(address => uint) public LiquidityTokenQueue; // Delays changes to mapping.

address[] public liquidityDepositors; // Push only, beware false-positives. Only for viewing.
mapping(address => bool) public isLiquidityDepositor;
mapping(address => uint) public LiquidityDepositorQueue; // Delays changes to mapping.

mapping(address => address) public bondCalculator; // bond calculator for liquidity token

address[] public reserveManagers; // Push only, beware false-positives. Only for viewing.
mapping(address => bool) public isReserveManager;
mapping(address => uint) public ReserveManagerQueue; // Delays changes to mapping.

address[] public liquidityManagers; // Push only, beware false-positives. Only for viewing.
mapping(address => bool) public isLiquidityManager;
mapping(address => uint) public LiquidityManagerQueue; // Delays changes to mapping.

address[] public debtors; // Push only, beware false-positives. Only for viewing.
mapping(address => bool) public isDebtor;
mapping(address => uint) public debtorQueue; // Delays changes to mapping.
mapping(address => uint) public debtorBalance;

address[] public rewardManagers; // Push only, beware false-positives. Only for viewing.
mapping(address => bool) public isRewardManager;
mapping(address => uint) public rewardManagerQueue; // Delays changes to mapping.

address public sOHM;
address public curve;
uint public sOHMQueue; // Delays change to sOHM address

uint public totalReserves; // Risk-free value of all assets
uint public totalDebt;

constructor(
  address _OHM,
  address _USDT,
  address _USDTOHM,
  address _calu,
  uint _blocksNeededForQueue,
  address _curve
) {
  require(_OHM != address(0));
  OHM = _OHM;

  isReserveToken[_USDT] = true;
  reserveTokens.push(_USDT);

  isLiquidityToken[_USDTOHM] = true;
  liquidityTokens.push(_USDTOHM);
  bondCalculator[_USDTOHM] = _calu;
  blocksNeededForQueue = _blocksNeededForQueue;
  curve = _curve;
}

/**
      @notice allow approved address to deposit an asset for OHM
      @param _amount uint
      @param _token address
```

Armors Labs

```
      @param _profit uint
      @return send_ uint
    */
function deposit(
  uint _amount,
  address _token,
  uint _profit
) external returns (uint send_) {
  require(isReserveToken[_token] || isLiquidityToken[_token], "Not accepted");
  IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);

  if (isReserveToken[_token]) {
    require(isReserveDepositor[msg.sender], "Not approved");
  } else {
    require(isLiquidityDepositor[msg.sender], "Not approved");
  }

  uint value = valueOf(_token, _amount);
  // mint OHM needed and store amount of rewards for distribution
  send_ = value.sub(_profit);
  IERC20Mintable(OHM).mint(msg.sender, send_);

  totalReserves = totalReserves.add(value);
  emit ReservesUpdated(totalReserves);

  emit Deposit(_token, _amount, value);
}

/**
      @notice allow approved address to burn OHM for reserves
      @param _amount uint
      @param _token address
    */
function withdraw(uint _amount, address _token) external {
  require(isReserveToken[_token], "Not accepted"); // Only reserves can be used for redemptions
  require(isReserveSpender[msg.sender] == true, "Not approved");

  uint value = valueOf(_token, _amount);
  IOHMERC20(OHM).burnFrom(msg.sender, value);

  totalReserves = totalReserves.sub(value);
  emit ReservesUpdated(totalReserves);

  IERC20(_token).safeTransfer(msg.sender, _amount);

  emit Withdrawal(_token, _amount, value);
}

/**
      @notice allow approved address to borrow reserves
      @param _amount uint
      @param _token address
    */
function incurDebt(uint _amount, address _token) external {
  require(isDebtor[msg.sender], "Not approved");
  require(isReserveToken[_token], "Not accepted");

  uint value = valueOf(_token, _amount);

  uint maximumDebt = IERC20(sOHM).balanceOf(msg.sender); // Can only borrow against sOHM held
  uint availableDebt = maximumDebt.sub(debtorBalance[msg.sender]);
  require(value <= availableDebt, "Exceeds debt limit");

  debtorBalance[msg.sender] = debtorBalance[msg.sender].add(value);
  totalDebt = totalDebt.add(value);
```

Armors Labs

```
    totalReserves = totalReserves.sub(value);
    emit ReservesUpdated(totalReserves);

    IERC20(_token).transfer(msg.sender, _amount);

    emit CreateDebt(msg.sender, _token, _amount, value);
  }

  /**
        @notice allow approved address to repay borrowed reserves with reserves
        @param _amount uint
        @param _token address
     */
  function repayDebtWithReserve(uint _amount, address _token) external {
    require(isDebtor[msg.sender], "Not approved");
    require(isReserveToken[_token], "Not accepted");

    IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);

    uint value = valueOf(_token, _amount);
    debtorBalance[msg.sender] = debtorBalance[msg.sender].sub(value);
    totalDebt = totalDebt.sub(value);

    totalReserves = totalReserves.add(value);
    emit ReservesUpdated(totalReserves);

    emit RepayDebt(msg.sender, _token, _amount, value);
  }

  /**
        @notice allow approved address to repay borrowed reserves with OHM
        @param _amount uint
     */
  function repayDebtWithOHM(uint _amount) external {
    require(isDebtor[msg.sender], "Not approved");

    IOHMERC20(OHM).burnFrom(msg.sender, _amount);

    debtorBalance[msg.sender] = debtorBalance[msg.sender].sub(_amount);
    totalDebt = totalDebt.sub(_amount);

    emit RepayDebt(msg.sender, OHM, _amount, _amount);
  }

  /**
        @notice allow approved address to withdraw assets
        @param _token address
        @param _amount uint
     */
  function manage(address _token, uint _amount) external {
    if (isLiquidityToken[_token]) {
      require(isLiquidityManager[msg.sender], "Not approved");
    } else {
      require(isReserveManager[msg.sender], "Not approved");
    }

    uint value = valueOf(_token, _amount);
    require(value <= excessReserves(), "Insufficient reserves");

    totalReserves = totalReserves.sub(value);
    emit ReservesUpdated(totalReserves);

    IERC20(_token).safeTransfer(msg.sender, _amount);

    emit ReservesManaged(_token, _amount);
  }
```

Armors Labs

```
/**
      @notice send epoch reward to staking contract
    */
function mintRewards(address _recipient, uint _amount) external {
  require(isRewardManager[msg.sender], "Not approved");
  require(_amount <= excessReserves(), "Insufficient reserves");

  IERC20Mintable(OHM).mint(_recipient, _amount);

  emit RewardsMinted(msg.sender, _recipient, _amount);
}

/**
      @notice returns excess reserves not backing tokens
      @return uint
    */
function excessReserves() public view returns (uint) {
  return totalReserves.sub(IERC20(OHM).totalSupply().sub(totalDebt));
}

/**
      @notice takes inventory of all tracked assets
      @notice always consolidate to recognized reserves before audit
    */
function auditReserves() external onlyManager {
  uint reserves;
  for (uint i = 0; i < reserveTokens.length; i++) {
    reserves = reserves.add(
      valueOf(
        reserveTokens[i],
        IERC20(reserveTokens[i]).balanceOf(address(this))
      )
    );
  }
  for (uint i = 0; i < liquidityTokens.length; i++) {
    reserves = reserves.add(
      valueOf(
        liquidityTokens[i],
        IERC20(liquidityTokens[i]).balanceOf(address(this))
      )
    );
  }
  totalReserves = reserves;
  emit ReservesUpdated(reserves);
  emit ReservesAudited(reserves);
}

/**
      @notice returns OHM valuation of asset
      @param _token address
      @param _amount uint
      @return value_ uint
    */
function valueOf(
  address _token,
  uint _amount
) public view returns (uint value_) {
  if (isReserveToken[_token]) {
    // convert amount to match OHM decimals
    value_ = _amount.mul(10 ** IERC20(OHM).decimals()).div(
      10 ** IERC20(_token).decimals()
    );
  } else if (isLiquidityToken[_token]) {
    value_ = IBondCalculator(bondCalculator[_token]).valuation(
      _token,
```

Armors Labs

```
      _amount
    );
  }
}

/**
      @notice queue address to change boolean in mapping
      @param _managing MANAGING
      @param _address address
      @return bool
   */
function queue(
  MANAGING _managing,
  address _address
) external onlyManager returns (bool) {
  require(_address != address(0));
  if (_managing == MANAGING.RESERVEDEPOSITOR) {
    // 0
    reserveDepositorQueue[_address] = block.number.add(blocksNeededForQueue);
  } else if (_managing == MANAGING.RESERVESPENDER) {
    // 1
    reserveSpenderQueue[_address] = block.number.add(blocksNeededForQueue);
  } else if (_managing == MANAGING.RESERVETOKEN) {
    // 2
    reserveTokenQueue[_address] = block.number.add(blocksNeededForQueue);
  } else if (_managing == MANAGING.RESERVEMANAGER) {
    // 3
    ReserveManagerQueue[_address] = block.number.add(
      blocksNeededForQueue.mul(2)
    );
  } else if (_managing == MANAGING.LIQUIDITYDEPOSITOR) {
    // 4
    LiquidityDepositorQueue[_address] = block.number.add(
      blocksNeededForQueue
    );
  } else if (_managing == MANAGING.LIQUIDITYTOKEN) {
    // 5
    LiquidityTokenQueue[_address] = block.number.add(blocksNeededForQueue);
  } else if (_managing == MANAGING.LIQUIDITYMANAGER) {
    // 6
    LiquidityManagerQueue[_address] = block.number.add(
      blocksNeededForQueue.mul(2)
    );
  } else if (_managing == MANAGING.DEBTOR) {
    // 7
    debtorQueue[_address] = block.number.add(blocksNeededForQueue);
  } else if (_managing == MANAGING.REWARDMANAGER) {
    // 8
    rewardManagerQueue[_address] = block.number.add(blocksNeededForQueue);
  } else if (_managing == MANAGING.SOHM) {
    // 9
    sOHMQueue = block.number.add(blocksNeededForQueue);
  } else return false;

  emit ChangeQueued(_managing, _address);
  return true;
}

/**
      @notice verify queue then set boolean in mapping
      @param _managing MANAGING
      @param _address address
      @param _calculator address
      @return bool
   */
function toggle(
```

Armors Labs

```
    MANAGING _managing,
    address _address,
    address _calculator
  ) external onlyManager returns (bool) {
    require(_address != address(0));
    bool result;
    if (_managing == MANAGING.RESERVEDEPOSITOR) {
      // 0
      if (requirements(reserveDepositorQueue, isReserveDepositor, _address)) {
        reserveDepositorQueue[_address] = 0;
        if (!listContains(reserveDepositors, _address)) {
          reserveDepositors.push(_address);
        }
      }
      result = !isReserveDepositor[_address];
      isReserveDepositor[_address] = result;
    } else if (_managing == MANAGING.RESERVESPENDER) {
      // 1
      if (requirements(reserveSpenderQueue, isReserveSpender, _address)) {
        reserveSpenderQueue[_address] = 0;
        if (!listContains(reserveSpenders, _address)) {
          reserveSpenders.push(_address);
        }
      }
      result = !isReserveSpender[_address];
      isReserveSpender[_address] = result;
    } else if (_managing == MANAGING.RESERVETOKEN) {
      // 2
      if (requirements(reserveTokenQueue, isReserveToken, _address)) {
        reserveTokenQueue[_address] = 0;
        if (!listContains(reserveTokens, _address)) {
          reserveTokens.push(_address);
        }
      }
      result = !isReserveToken[_address];
      isReserveToken[_address] = result;
    } else if (_managing == MANAGING.RESERVEMANAGER) {
      // 3
      if (requirements(ReserveManagerQueue, isReserveManager, _address)) {
        reserveManagers.push(_address);
        ReserveManagerQueue[_address] = 0;
        if (!listContains(reserveManagers, _address)) {
          reserveManagers.push(_address);
        }
      }
      result = !isReserveManager[_address];
      isReserveManager[_address] = result;
    } else if (_managing == MANAGING.LIQUIDITYDEPOSITOR) {
      // 4
      if (
        requirements(LiquidityDepositorQueue, isLiquidityDepositor, _address)
      ) {
        liquidityDepositors.push(_address);
        LiquidityDepositorQueue[_address] = 0;
        if (!listContains(liquidityDepositors, _address)) {
          liquidityDepositors.push(_address);
        }
      }
      result = !isLiquidityDepositor[_address];
      isLiquidityDepositor[_address] = result;
    } else if (_managing == MANAGING.LIQUIDITYTOKEN) {
      // 5
      if (requirements(LiquidityTokenQueue, isLiquidityToken, _address)) {
        LiquidityTokenQueue[_address] = 0;
        if (!listContains(liquidityTokens, _address)) {
          liquidityTokens.push(_address);
```

Armors Labs

```
      }
    }
    result = !isLiquidityToken[_address];
    isLiquidityToken[_address] = result;
    bondCalculator[_address] = _calculator;
  } else if (_managing == MANAGING.LIQUIDITYMANAGER) {
    // 6
    if (requirements(LiquidityManagerQueue, isLiquidityManager, _address)) {
      LiquidityManagerQueue[_address] = 0;
      if (!listContains(liquidityManagers, _address)) {
        liquidityManagers.push(_address);
      }
    }
    result = !isLiquidityManager[_address];
    isLiquidityManager[_address] = result;
  } else if (_managing == MANAGING.DEBTOR) {
    // 7
    if (requirements(debtorQueue, isDebtor, _address)) {
      debtorQueue[_address] = 0;
      if (!listContains(debtors, _address)) {
        debtors.push(_address);
      }
    }
    result = !isDebtor[_address];
    isDebtor[_address] = result;
  } else if (_managing == MANAGING.REWARDMANAGER) {
    // 8
    if (requirements(rewardManagerQueue, isRewardManager, _address)) {
      rewardManagerQueue[_address] = 0;
      if (!listContains(rewardManagers, _address)) {
        rewardManagers.push(_address);
      }
    }
    result = !isRewardManager[_address];
    isRewardManager[_address] = result;
  } else if (_managing == MANAGING.SOHM) {
    // 9
    sOHMQueue = 0;
    sOHM = _address;
    result = true;
  } else return false;

  emit ChangeActivated(_managing, _address, result);
  return true;
}

/**
      @notice checks requirements and returns altered structs
      @param queue_ mapping( address => uint )
      @param status_ mapping( address => bool )
      @param _address address
      @return bool
   */
function requirements(
  mapping(address => uint) storage queue_,
  mapping(address => bool) storage status_,
  address _address
) internal view returns (bool) {
  if (!status_[_address]) {
    require(queue_[_address] != 0, "Must queue");
    require(queue_[_address] <= block.number, "Queue not expired");
    return true;
  }
  return false;
}
```

Armors Labs

```
  /**
        @notice checks array to ensure against duplicate
        @param _list address[]
        @param _token address
        @return bool
     */
  function listContains(
    address[] storage _list,
    address _token
  ) internal view returns (bool) {
    for (uint i = 0; i < _list.length; i++) {
      if (_list[i] == _token) {
        return true;
      }
    }
    return false;
  }

  /**
   * @notice allow manager to swap reserve tokens to main coins
   * @param i uint256 // sell token
   * @param j uint256 // buy token
   */
  function swapReserve(
    uint256 i,
    uint256 j,
    // address _fromToken,
    // address _toToken,
    uint _amount
  ) external onlyManager {
    address fromToken = ICurve(curve).underlying_coins(i);
    address toToken = ICurve(curve).underlying_coins(j);
    require(isReserveToken[fromToken], "Not accepted");
    require(isReserveToken[toToken], "Not accepted");
    require(
      IERC20(fromToken).balanceOf(address(this)) >= _amount,
      "Insufficient reserves"
    );

    IERC20(fromToken).approve(curve, _amount);
    ICurve(curve).exchange_underlying(i, j, _amount, 1);

    emit ReservesManaged(fromToken, _amount);
  }

  function setCurve(address _curve) external onlyManager {
    curve = _curve;
  }
}
```

Armors Labs

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!
```

Armors Labs

- **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.

Armors Labs

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!
```

Armors Labs

- **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

Armors Labs

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 Labs

armors.io

contact@armors.io