
Armors Labs
Rpone.finance
Smart Contract Audit

Rpone.finance Audit Summary

Project name : Rpone.finance Contract

Project address: None

Code URL : https://hecoinfo.com/address/0x3D6B11aF826df79aACe31923f85CAc7F89c8f3e2#code

Projct target : Rpone.finance Contract Audit

Blockchain : Huobi ECO Chain（Heco）

Test result : PASSED

Audit Info

Audit NO : 0X202104200008

Audit Team : Armors Labs

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

Rpone.finance Audit

The Rpone.finance team asked us to review and audit their Rpone.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
Rpone.finance Audit	Rock ,Hosea, Rushairer	1.0.1	2021-04-20

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 Rpone.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 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
FSwapFactory.sol	276f42ea2f58bee7439330c7bef516f8

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

Contract code

```
/**
 *Submitted for verification at hecoinfo.com on 2021-04-19
*/

// File: contracts/interface/IFSwapFactory.sol

pragma solidity =0.5.16;

interface IFSwapFactory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    function feeTo() external view returns (address);

    function feeToSetter() external view returns (address);

    function feeToRate() external view returns (uint256);

    function initCodeHash() external view returns (bytes32);

    function getPair(address tokenA, address tokenB) external view returns (address pair);

    function allPairs(uint) external view returns (address pair);

    function allPairsLength() external view returns (uint);

    function createPair(address tokenA, address tokenB) external returns (address pair);

    function setFeeTo(address) external;

    function setFeeToSetter(address) external;

    function setFeeToRate(uint256) external;

    function setInitCodeHash(bytes32) external;

    function sortTokens(address tokenA, address tokenB) external pure returns (address token0, addres

    function pairFor(address tokenA, address tokenB) external view returns (address pair);

    function getReserves(address tokenA, address tokenB) external view returns (uint256 reserveA, uin

    function quote(uint256 amountA, uint256 reserveA, uint256 reserveB) external pure returns (uint25

    function getAmountOut(
        uint256 amountIn, uint256 reserveIn, uint256 reserveOut
    ) external view returns (uint256 amountOut);

    function getAmountIn(
        uint256 amountOut, uint256 reserveIn, uint256 reserveOut
    ) external view returns (uint256 amountIn);

    function getAmountsOut(uint256 amountIn, address[] calldata path) external view returns (uint256[

    function getAmountsIn(uint256 amountOut, address[] calldata path) external view returns (uint256[
}

// File: contracts/interface/IFSwapPair.sol

pragma solidity =0.5.16;

interface IFSwapPair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);
```

```
    function name() external pure returns (string memory);

    function symbol() external pure returns (string memory);

    function decimals() external pure returns (uint8);

    function totalSupply() external view returns (uint);

    function balanceOf(address owner) external view returns (uint);

    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);

    function transfer(address to, uint value) external returns (bool);

    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);

    function PERMIT_TYPEHASH() external pure returns (bytes32);

    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);

    function factory() external view returns (address);

    function token0() external view returns (address);

    function token1() external view returns (address);

    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTim

    function price0CumulativeLast() external view returns (uint);

    function price1CumulativeLast() external view returns (uint);

    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);

    function burn(address to) external returns (uint amount0, uint amount1);

    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;

    function skim(address to) external;

    function sync() external;

    function price(address token, uint256 baseDecimal) external view returns (uint256);
```

```
    function initialize(address, address) external;
}

// File: contracts/interface/IERC20.sol

pragma solidity =0.5.16;

interface IERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external view returns (string memory);

    function symbol() external view returns (string memory);

    function decimals() external view returns (uint8);

    function totalSupply() external view returns (uint);

    function balanceOf(address owner) external view returns (uint);

    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);

    function transfer(address to, uint value) external returns (bool);

    function transferFrom(address from, address to, uint value) external returns (bool);
}

// File: contracts/interface/IFSwapCallee.sol

pragma solidity =0.5.16;

interface IFSwapCallee {
    function uniswapV2Call(address sender, uint amount0, uint amount1, bytes calldata data) external;
}

// File: contracts/interface/IFSwapERC20.sol

pragma solidity =0.5.16;

interface IFSwapERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);

    function symbol() external pure returns (string memory);

    function decimals() external pure returns (uint8);

    function totalSupply() external view returns (uint);

    function balanceOf(address owner) external view returns (uint);

    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);

    function transfer(address to, uint value) external returns (bool);

    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
```

```
    function PERMIT_TYPEHASH() external pure returns (bytes32);

    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by
}

// File: contracts/SafeMath.sol

pragma solidity =0.5.16;

library SafeMath {
    uint256 constant WAD = 10 ** 18;
    uint256 constant RAY = 10 ** 27;

    function wad() public pure returns (uint256) {
        return WAD;
    }

    function ray() public pure returns (uint256) {
        return RAY;
    }

    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) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        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) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
```

```
    }

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

    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }

    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a <= b ? a : b;
    }

    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }

    function sqrt(uint256 a) internal pure returns (uint256 b) {
        if (a > 3) {
            b = a;
            uint256 x = a / 2 + 1;
            while (x < b) {
                b = x;
                x = (a / x + x) / 2;
            }
        } else if (a != 0) {
            b = 1;
        }
    }

    function wmul(uint256 a, uint256 b) internal pure returns (uint256) {
        return mul(a, b) / WAD;
    }

    function wmulRound(uint256 a, uint256 b) internal pure returns (uint256) {
        return add(mul(a, b), WAD / 2) / WAD;
    }

    function rmul(uint256 a, uint256 b) internal pure returns (uint256) {
        return mul(a, b) / RAY;
    }

    function rmulRound(uint256 a, uint256 b) internal pure returns (uint256) {
        return add(mul(a, b), RAY / 2) / RAY;
    }

    function wdiv(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(mul(a, WAD), b);
    }

    function wdivRound(uint256 a, uint256 b) internal pure returns (uint256) {
        return add(mul(a, WAD), b / 2) / b;
    }

    function rdiv(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(mul(a, RAY), b);
    }

    function rdivRound(uint256 a, uint256 b) internal pure returns (uint256) {
        return add(mul(a, RAY), b / 2) / b;
    }

    function wpow(uint256 x, uint256 n) internal pure returns (uint256) {
        uint256 result = WAD;
```

```
        while (n > 0) {
            if (n % 2 != 0) {
                result = wmul(result, x);
            }
            x = wmul(x, x);
            n /= 2;
        }
        return result;
    }

    function rpow(uint256 x, uint256 n) internal pure returns (uint256) {
        uint256 result = RAY;
        while (n > 0) {
            if (n % 2 != 0) {
                result = rmul(result, x);
            }
            x = rmul(x, x);
            n /= 2;
        }
        return result;
    }
}

// File: contracts/FSwapERC20.sol

pragma solidity =0.5.16;



contract FSwapERC20 is IFSwapERC20 {
    using SafeMath for uint;

    string public constant name = 'Rpone LP Token';
    string public constant symbol = 'RPT-LP';
    uint8 public constant decimals = 18;
    uint  public totalSupply;
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint)) public allowance;

    bytes32 public DOMAIN_SEPARATOR;
    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)"
    bytes32 public constant PERMIT_TYPEHASH = 0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c6
    mapping(address => uint) public nonces;

    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    constructor() public {
        uint chainId;
        assembly {
            chainId := chainid
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256('EIP712Domain(string name,string version,uint256 chainId,address verifyingC
                keccak256(bytes(name)),
                keccak256(bytes('1')),
                chainId,
                address(this)
            )
        );
    }

    function _mint(address to, uint value) internal {
        totalSupply = totalSupply.add(value);
        balanceOf[to] = balanceOf[to].add(value);
```

```
        emit Transfer(address(0), to, value);
    }

    function _burn(address from, uint value) internal {
        balanceOf[from] = balanceOf[from].sub(value);
        totalSupply = totalSupply.sub(value);
        emit Transfer(from, address(0), value);
    }

    function _approve(address owner, address spender, uint value) private {
        allowance[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    function _transfer(address from, address to, uint value) private {
        balanceOf[from] = balanceOf[from].sub(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(from, to, value);
    }

    function approve(address spender, uint value) external returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }

    function transfer(address to, uint value) external returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
    }

    function transferFrom(address from, address to, uint value) external returns (bool) {
        if (allowance[from][msg.sender] != uint(- 1)) {
            allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);
        }
        _transfer(from, to, value);
        return true;
    }

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by
        require(deadline >= block.timestamp, 'Swap: EXPIRED');
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++, deadlin
            )
        );
        address recoveredAddress = ecrecover(digest, v, r, s);
        require(recoveredAddress != address(0) && recoveredAddress == owner, 'Swap: INVALID_SIGNATURE
        _approve(owner, spender, value);
    }
}

// File: contracts/UQ112x112.sol

pragma solidity =0.5.16;

// a library for handling binary fixed point numbers (https://en.wikipedia.org/wiki/Q_(number_format)

// range: [0, 2**112 - 1]
// resolution: 1 / 2**112

library UQ112x112 {
    uint224 constant Q112 = 2 ** 112;

    // encode a uint112 as a UQ112x112
```

```
    function encode(uint112 y) internal pure returns (uint224 z) {
        z = uint224(y) * Q112;
        // never overflows
    }

    // divide a UQ112x112 by a uint112, returning a UQ112x112
    function uqdiv(uint224 x, uint112 y) internal pure returns (uint224 z) {
        z = x / uint224(y);
    }
}

// File: contracts/FSwapPair.sol

pragma solidity =0.5.16;







contract FSwapPair is IFSwapPair, FSwapERC20 {
    using SafeMath  for uint;
    using UQ112x112 for uint224;

    uint public constant MINIMUM_LIQUIDITY = 10 ** 3;
    bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)')));

    address public factory;
    address public token0;
    address public token1;

    uint112 private reserve0;           // uses single storage slot, accessible via getReserves
    uint112 private reserve1;           // uses single storage slot, accessible via getReserves
    uint32  private blockTimestampLast; // uses single storage slot, accessible via getReserves

    uint public price0CumulativeLast;
    uint public price1CumulativeLast;
    uint public kLast; // reserve0 * reserve1, as of immediately after the most recent liquidity even

    uint private unlocked = 1;
    modifier lock() {
        require(unlocked == 1, 'Swap: LOCKED');
        unlocked = 0;
        _;
        unlocked = 1;
    }

    function getReserves() public view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTi
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }

    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'Swap: TRANSFER_FAILED');
    }

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
```

```
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    constructor() public {
        factory = msg.sender;
    }

    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external {
        require(msg.sender == factory, 'Swap: FORBIDDEN');
        // sufficient check
        token0 = _token0;
        token1 = _token1;
    }

    function salvageToken(address _token) external {
        require(token1 != _token && token0 != _token, "not swap token");
        require(IFSwapFactory(factory).feeTo() != address(0), "feeTo is 0");
        address feeTo = IFSwapFactory(factory).feeTo();
        _safeTransfer(_token, feeTo, IERC20(_token).balanceOf(address(this)));

    }

    // update reserves and, on the first call per block, price accumulators
    function _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1) private {
        require(balance0 <= uint112(- 1) && balance1 <= uint112(- 1), 'FSwapSwap: OVERFLOW');
        uint32 blockTimestamp = uint32(block.timestamp % 2 ** 32);
        uint32 timeElapsed = blockTimestamp - blockTimestampLast;
        // overflow is desired
        if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
            // * never overflows, and + overflow is desired
            price0CumulativeLast += uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed;
            price1CumulativeLast += uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed;
        }
        reserve0 = uint112(balance0);
        reserve1 = uint112(balance1);
        blockTimestampLast = blockTimestamp;
        emit Sync(reserve0, reserve1);
    }

    // if fee is on, mint liquidity equivalent to 1/6th of the growth in sqrt(k)
    function _mintFee(uint112 _reserve0, uint112 _reserve1) private returns (bool feeOn) {
        address feeTo = IFSwapFactory(factory).feeTo();
        feeOn = feeTo != address(0);
        uint _kLast = kLast;
        // gas savings
        if (feeOn) {
            if (_kLast != 0) {
                uint rootK = SafeMath.sqrt(uint(_reserve0).mul(_reserve1));
                uint rootKLast = SafeMath.sqrt(_kLast);
                if (rootK > rootKLast) {
                    uint numerator = totalSupply.mul(rootK.sub(rootKLast));
                    uint denominator = rootK.mul(IFSwapFactory(factory).feeToRate()).add(rootKLast);
                    uint liquidity = numerator / denominator;
                    if (liquidity > 0) _mint(feeTo, liquidity);
                }
            }
        } else if (_kLast != 0) {
            kLast = 0;
        }
    }

    // this low-level function should be called from a contract which performs important safety check
```

```
    function mint(address to) external lock returns (uint liquidity) {
        (uint112 _reserve0, uint112 _reserve1,) = getReserves();
        // gas savings
        uint balance0 = IERC20(token0).balanceOf(address(this));
        uint balance1 = IERC20(token1).balanceOf(address(this));
        uint amount0 = balance0.sub(_reserve0);
        uint amount1 = balance1.sub(_reserve1);

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply;
        // gas savings, must be defined here since totalSupply can update in _mintFee
        if (_totalSupply == 0) {
            liquidity = SafeMath.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
            _mint(address(0), MINIMUM_LIQUIDITY);
            // permanently lock the first MINIMUM_LIQUIDITY tokens
        } else {
            liquidity = SafeMath.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply)
        }
        require(liquidity > 0, 'Swap: INSUFFICIENT_LIQUIDITY_MINTED');
        _mint(to, liquidity);

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1);
        // reserve0 and reserve1 are up-to-date
        emit Mint(msg.sender, amount0, amount1);
    }

    // this low-level function should be called from a contract which performs important safety check
    function burn(address to) external lock returns (uint amount0, uint amount1) {
        (uint112 _reserve0, uint112 _reserve1,) = getReserves();
        // gas savings
        address _token0 = token0;
        // gas savings
        address _token1 = token1;
        // gas savings
        uint balance0 = IERC20(_token0).balanceOf(address(this));
        uint balance1 = IERC20(_token1).balanceOf(address(this));
        uint liquidity = balanceOf[address(this)];

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply;
        // gas savings, must be defined here since totalSupply can update in _mintFee
        amount0 = liquidity.mul(balance0) / _totalSupply;
        // using balances ensures pro-rata distribution
        amount1 = liquidity.mul(balance1) / _totalSupply;
        // using balances ensures pro-rata distribution
        require(amount0 > 0 && amount1 > 0, 'Swap: INSUFFICIENT_LIQUIDITY_BURNED');
        _burn(address(this), liquidity);
        _safeTransfer(_token0, to, amount0);
        _safeTransfer(_token1, to, amount1);
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1);
        // reserve0 and reserve1 are up-to-date
        emit Burn(msg.sender, amount0, amount1, to);
    }

    // this low-level function should be called from a contract which performs important safety check
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
        require(amount0Out > 0 || amount1Out > 0, 'Swap: INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves();
        // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'Swap: INSUFFICIENT_LIQUIDITY');
```

```
        uint balance0;
        uint balance1;
        {// scope for _token{0,1}, avoids stack too deep errors
            address _token0 = token0;
            address _token1 = token1;
            require(to != _token0 && to != _token1, 'Swap: INVALID_TO');
            if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out);
            // optimistically transfer tokens
            if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out);
            // optimistically transfer tokens
            if (data.length > 0) IFSwapCallee(to).uniswapV2Call(msg.sender, amount0Out, amount1Out, d
            balance0 = IERC20(_token0).balanceOf(address(this));
            balance1 = IERC20(_token1).balanceOf(address(this));
        }
        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'Swap: INSUFFICIENT_INPUT_AMOUNT');
        {// scope for reserve{0,1}Adjusted, avoids stack too deep errors
            uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
            uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
            require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }

    // force balances to match reserves
    function skim(address to) external lock {
        address _token0 = token0;
        // gas savings
        address _token1 = token1;
        // gas savings
        _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).sub(reserve0));
        _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
    }

    // force reserves to match balances
    function sync() external lock {
        _update(IERC20(token0).balanceOf(address(this)), IERC20(token1).balanceOf(address(this)), res
    }

    function price(address token, uint256 baseDecimal) public view returns (uint256) {
        if ((token0 != token && token1 != token) || 0 == reserve0 || 0 == reserve1) {
            return 0;
        }
        if (token0 == token) {
            return uint256(reserve1).mul(baseDecimal).div(uint256(reserve0));
        } else {
            return uint256(reserve0).mul(baseDecimal).div(uint256(reserve1));
        }
    }

}

// File: contracts/FSwapFactory.sol

pragma solidity =0.5.16;



contract FSwapFactory is IFSwapFactory {
    using SafeMath for uint256;
    address public feeTo;
    address public feeToSetter;
```

```
    uint256 public feeToRate;
    bytes32 public initCodeHash;
    bool public initCode = false;

    mapping(address => mapping(address => address)) public getPair;
    address[] public allPairs;

    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
    }

    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }

    function createPair(address tokenA, address tokenB) external returns (address pair) {
        require(tokenA != tokenB, 'SwapFactory: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'SwapFactory: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'SwapFactory: PAIR_EXISTS');
        // single check is sufficient
        bytes memory bytecode = type(FSwapPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        IFSwapPair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair;
        // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'SwapFactory: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'SwapFactory: FORBIDDEN');
        require(_feeToSetter != address(0), "SwapFactory: FeeToSetter is zero address");
        feeToSetter = _feeToSetter;
    }

    function setFeeToRate(uint256 _rate) external {
        require(msg.sender == feeToSetter, 'SwapFactory: FORBIDDEN');
        require(_rate > 0, "SwapFactory: FEE_TO_RATE_OVERFLOW");
        feeToRate = _rate.sub(1);
    }

    function setInitCodeHash(bytes32 _initCodeHash) external {
        require(msg.sender == feeToSetter, 'SwapFactory: FORBIDDEN');
        require(initCode == false, "SwapFactory: Do not repeat settings initCodeHash");
        initCodeHash = _initCodeHash;
        initCode = true;
    }

    // returns sorted token addresses, used to handle return values from pairs sorted in this order
    function sortTokens(address tokenA, address tokenB) public pure returns (address token0, address
        require(tokenA != tokenB, 'SwapFactory: IDENTICAL_ADDRESSES');
        (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'SwapFactory: ZERO_ADDRESS');
    }
```

```
function getInitCodeHash() public pure returns (bytes32) {
    return keccak256(abi.encodePacked(type(FSwapPair).creationCode));
}

// calculates the CREATE2 address for a pair without making any external calls
function pairFor(address tokenA, address tokenB) public view returns (address pair) {
    (address token0, address token1) = sortTokens(tokenA, tokenB);
    pair = address(uint(keccak256(abi.encodePacked(
            hex'ff',
            address(this),
            keccak256(abi.encodePacked(token0, token1)),
            initCodeHash
        ))));
}
// fetches and sorts the reserves for a pair
function getReserves(address tokenA, address tokenB) public view returns (uint reserveA, uint res
    (address token0,) = sortTokens(tokenA, tokenB);
    (uint reserve0, uint reserve1,) = IFSwapPair(pairFor(tokenA, tokenB)).getReserves();
    (reserveA, reserveB) = tokenA == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
}

// given some amount of an asset and pair reserves, returns an equivalent amount of the other ass
function quote(uint amountA, uint reserveA, uint reserveB) public pure returns (uint amountB) {
    require(amountA > 0, 'SwapFactory: INSUFFICIENT_AMOUNT');
    require(reserveA > 0 && reserveB > 0, 'SwapFactory: INSUFFICIENT_LIQUIDITY');
    amountB = amountA.mul(reserveB) / reserveA;
}

// given an input amount of an asset and pair reserves, returns the maximum output amount of the
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) public view returns (uint a
    require(amountIn > 0, 'SwapFactory: INSUFFICIENT_INPUT_AMOUNT');
    require(reserveIn > 0 && reserveOut > 0, 'SwapFactory: INSUFFICIENT_LIQUIDITY');
    uint amountInWithFee = amountIn.mul(997);
    uint numerator = amountInWithFee.mul(reserveOut);
    uint denominator = reserveIn.mul(1000).add(amountInWithFee);
    amountOut = numerator / denominator;
}

// given an output amount of an asset and pair reserves, returns a required input amount of the o
function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) public view returns (uint a
    require(amountOut > 0, 'SwapFactory: INSUFFICIENT_OUTPUT_AMOUNT');
    require(reserveIn > 0 && reserveOut > 0, 'SwapFactory: INSUFFICIENT_LIQUIDITY');
    uint numerator = reserveIn.mul(amountOut).mul(1000);
    uint denominator = reserveOut.sub(amountOut).mul(997);
    amountIn = (numerator / denominator).add(1);
}

// performs chained getAmountOut calculations on any number of pairs
function getAmountsOut(uint amountIn, address[] memory path) public view returns (uint[] memory a
    require(path.length >= 2, 'SwapFactory: INVALID_PATH');
    amounts = new uint[](path.length);
    amounts[0] = amountIn;
    for (uint i; i < path.length - 1; i++) {
        (uint reserveIn, uint reserveOut) = getReserves(path[i], path[i + 1]);
        amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut);
    }
}

// performs chained getAmountIn calculations on any number of pairs
function getAmountsIn(uint amountOut, address[] memory path) public view returns (uint[] memory a
    require(path.length >= 2, 'SwapFactory: INVALID_PATH');
    amounts = new uint[](path.length);
    amounts[amounts.length - 1] = amountOut;
    for (uint i = path.length - 1; i > 0; i--) {
```

```
            (uint reserveIn, uint reserveOut) = getReserves(path[i - 1], path[i]);
            amounts[i - 1] = getAmountIn(amounts[i], reserveIn, reserveOut);
        }
    }
}
```

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.

armors.io

contact@armors.io