Armors Labs

NetSwap

Smart Contract Audit

Armors Labs

NetSwap Audit Summary

Project name : NetSwap Contract

Project address: None

Code URL : https://github.com/Netswap/exchange-contracts

Commit : e34430282a0e705c035a7de355571c2b6bd07d2b

Project target : NetSwap Contract Audit

Blockchain : Metiz

Test result : PASSED

Audit Info

Audit NO : 0X202111240016

Audit Team : Armors Labs

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

NetSwap Audit

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

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

Armors Labs

Document information

Name	Auditor	Version	Date
NetSwap Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-11-24

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 NetSwap 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
NetSwapRouter.sol	161fd670ba5938c200428a8a3789a00a
INetswapRouter.sol	4c61a2ab1fe878ee834bcb1fe06383db
Multicall.sol	6e30ef37477c79aba9c76a701fdec84c
NetswapFactory.sol	fbb29c866cd93067d3c0a56ed74e801e

Vulnerability analysis

Vulnerability distribution

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

Armors Labs

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

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

Contract file

NetSwapRouter.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.5.0;

interface INetswapPair {
    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);
```

Armors Labs

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

pragma solidity =0.6.12;

// a library for performing overflow-safe math, courtesy of DappHub (https://github.com/dapphub/ds-ma

library SafeMathNetswap {
    function add(uint x, uint y) internal pure returns (uint z) {
        require((z = x + y) >= x, 'ds-math-add-overflow');
    }

    function sub(uint x, uint y) internal pure returns (uint z) {
        require((z = x - y) <= x, 'ds-math-sub-underflow');
    }

    function mul(uint x, uint y) internal pure returns (uint z) {
        require(y == 0 || (z = x * y) / y == x, 'ds-math-mul-overflow');
    }
}


pragma solidity >=0.5.0;

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

    function feeTo() external view returns (address);
    function feeRate() external view returns (uint);
    function feeToSetter() external view returns (address);

    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;
}

pragma solidity >=0.5.0;



library NetswapLibrary {
    using SafeMathNetswap for uint;

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

Armors Labs

Armors Labs

```
// calculates the CREATE2 address for a pair without making any external calls
function pairFor(address factory, address tokenA, address tokenB) internal pure returns (address
    (address token0, address token1) = sortTokens(tokenA, tokenB);
    pair = address(uint(keccak256(abi.encodePacked(
            hex'ff',
            factory,
            keccak256(abi.encodePacked(token0, token1)),
            hex'68cc803ebc27f23a62dd9f9251e76a9d6f2c659f76c92ffbd5e62d5b877384d6' // init code ha
        ))));
}

// fetches and sorts the reserves for a pair
function getReserves(address factory, address tokenA, address tokenB) internal view returns (uint
    (address token0,) = sortTokens(tokenA, tokenB);
    (uint reserve0, uint reserve1,) = INetswapPair(pairFor(factory, 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) internal pure returns (uint amountB) {
    require(amountA > 0, 'NetswapLibrary: INSUFFICIENT_AMOUNT');
    require(reserveA > 0 && reserveB > 0, 'NetswapLibrary: 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, uint feeRate) internal pure
    require(amountIn > 0, 'NetswapLibrary: INSUFFICIENT_INPUT_AMOUNT');
    require(reserveIn > 0 && reserveOut > 0, 'NetswapLibrary: INSUFFICIENT_LIQUIDITY');
    uint amountInWithFee = amountIn.mul(1000 - feeRate);
    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, uint feeRate) internal pure
    require(amountOut > 0, 'NetswapLibrary: INSUFFICIENT_OUTPUT_AMOUNT');
    require(reserveIn > 0 && reserveOut > 0, 'NetswapLibrary: INSUFFICIENT_LIQUIDITY');
    uint numerator = reserveIn.mul(amountOut).mul(1000);
    uint denominator = reserveOut.sub(amountOut).mul(1000 - feeRate);
    amountIn = (numerator / denominator).add(1);
}

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

// performs chained getAmountIn calculations on any number of pairs
function getAmountsIn(address factory, uint amountOut, address[] memory path, uint feeRate) inter
    require(path.length >= 2, 'NetswapLibrary: 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(factory, path[i - 1], path[i]);
        amounts[i - 1] = getAmountIn(amounts[i], reserveIn, reserveOut, feeRate);
    }
}
```

```
}

pragma solidity >=0.6.0;

// helper methods for interacting with ERC20 tokens and sending Metis that do not consistently return
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: APPROVE_F
    }

    function safeTransfer(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_
    }

    function safeTransferFrom(address token, address from, address to, uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, v
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_
    }

    function safeTransferMetis(address to, uint value) internal {
        (bool success,) = to.call{value:value}(new bytes(0));
        require(success, 'TransferHelper: METIS_TRANSFER_FAILED');
    }
}

pragma solidity >=0.6.2;

interface INetswapRouter {
    function factory() external pure returns (address);
    function Metis() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityMetis(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountMetis, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityMetis(
        address token,
        uint liquidity,
```

Armors Labs

```
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountMetis);
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityMetisWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountToken, uint amountMetis);
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapExactMetisForTokens(uint amountOutMin, address[] calldata path, address to, uint dea
        external
        payable
        returns (uint[] memory amounts);
    function swapTokensForExactMetis(uint amountOut, uint amountInMax, address[] calldata path, addre
        external
        returns (uint[] memory amounts);
    function swapExactTokensForMetis(uint amountIn, uint amountOutMin, address[] calldata path, addre
        external
        returns (uint[] memory amounts);
    function swapMetisForExactTokens(uint amountOut, address[] calldata path, address to, uint deadli
        external
        payable
        returns (uint[] memory amounts);

    function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external view returns (uint
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external view returns (uint
    function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memo
    function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memo
}

pragma solidity >=0.6.2;


interface INetswapRouter02 is INetswapRouter {
    function removeLiquidityMetisSupportingFeeOnTransferTokens(
        address token,
```

Armors Labs

```
        uint liquidity,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline
    ) external returns (uint amountMetis);
    function removeLiquidityMetisWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountMetis);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
    function swapExactMetisForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;
    function swapExactTokensForMetisSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
    function swapMining() external pure returns (address);
}


pragma solidity >=0.5.0;

interface IERC20Netswap {
    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);
}

interface ISwapMining {
    function swap(address account, address input, address output, uint256 amount) external returns (b
}

contract Ownable {
    address private _owner;

    constructor () internal {
```

Armors Labs

```
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

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

    function isOwner(address account) public view returns (bool) {
        return account == _owner;
    }

    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }

    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }


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

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
}


pragma solidity =0.6.12;

contract NetswapRouter is INetswapRouter02, Ownable {
    using SafeMathNetswap for uint;

    address public immutable override factory;
    address public immutable override Metis;
    address public override swapMining;

    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'NetswapRouter: EXPIRED');
        _;
    }

    constructor(address _factory, address _Metis) public {
        factory = _factory;
        Metis = _Metis;
    }

    function setSwapMining(address _swapMininng) public onlyOwner {
        swapMining = _swapMininng;
    }

    receive() external payable {}

    // **** ADD LIQUIDITY ****
    function _addLiquidity(
        address tokenA,
        address tokenB,
```

Armors Labs

```
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin
    ) internal virtual returns (uint amountA, uint amountB) {
        // create the pair if it doesn't exist yet
        if (INetswapFactory(factory).getPair(tokenA, tokenB) == address(0)) {
            INetswapFactory(factory).createPair(tokenA, tokenB);
        }
        (uint reserveA, uint reserveB) = NetswapLibrary.getReserves(factory, tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = NetswapLibrary.quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'NetswapRouter: INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = NetswapLibrary.quote(amountBDesired, reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'NetswapRouter: INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint amountA, uint amountB, uint liquidity)
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired, amountAMin
        address pair = NetswapLibrary.pairFor(factory, tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = INetswapPair(pair).mint(to);
    }
    function addLiquidityMetis(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline) returns (uint amountToken, uint amountMetis,
        (amountToken, amountMetis) = _addLiquidity(
            token,
            Metis,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountMetisMin
        );
        address pair = NetswapLibrary.pairFor(factory, token, Metis);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        // safe transfer Metis from router to pair
        TransferHelper.safeTransfer(Metis, pair, amountMetis);
        liquidity = INetswapPair(pair).mint(to);
        // refund dust metis, if any
        if (msg.value > amountMetis) TransferHelper.safeTransferMetis(msg.sender, msg.value - amountM
    }
```

Armors Labs

```
    // **** REMOVE LIQUIDITY ****
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountA, uint amountB) {
        address pair = NetswapLibrary.pairFor(factory, tokenA, tokenB);
        INetswapPair(pair).transferFrom(msg.sender, pair, liquidity); // send liquidity to pair
        (uint amount0, uint amount1) = INetswapPair(pair).burn(to);
        (address token0,) = NetswapLibrary.sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
        require(amountA >= amountAMin, 'NetswapRouter: INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'NetswapRouter: INSUFFICIENT_B_AMOUNT');
    }
    function removeLiquidityMetis(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountToken, uint amountMetis) {
        (amountToken, amountMetis) = removeLiquidity(
            token,
            Metis,
            liquidity,
            amountTokenMin,
            amountMetisMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, amountToken);
        TransferHelper.safeTransferMetis(to, amountMetis);
    }
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountA, uint amountB) {
        address pair = NetswapLibrary.pairFor(factory, tokenA, tokenB);
        uint value = approveMax ? uint(-1) : liquidity;
        INetswapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountA, amountB) = removeLiquidity(tokenA, tokenB, liquidity, amountAMin, amountBMin, to, d
    }
    function removeLiquidityMetisWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountToken, uint amountMetis) {
        address pair = NetswapLibrary.pairFor(factory, token, Metis);
        uint value = approveMax ? uint(-1) : liquidity;
        INetswapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountToken, amountMetis) = removeLiquidityMetis(token, liquidity, amountTokenMin, amountMet
```

Armors Labs

```
    }

    // **** REMOVE LIQUIDITY (supporting fee-on-transfer tokens) ****
    function removeLiquidityMetisSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountMetis) {
        (, amountMetis) = removeLiquidity(
            token,
            Metis,
            liquidity,
            amountTokenMin,
            amountMetisMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, IERC20Netswap(token).balanceOf(address(this)));
        TransferHelper.safeTransferMetis(to, amountMetis);
    }
    function removeLiquidityMetisWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountMetis) {
        address pair = NetswapLibrary.pairFor(factory, token, Metis);
        uint value = approveMax ? uint(-1) : liquidity;
        INetswapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        amountMetis = removeLiquidityMetisSupportingFeeOnTransferTokens(
            token, liquidity, amountTokenMin, amountMetisMin, to, deadline
        );
    }

    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, address[] memory path, address _to) internal virtual {
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = NetswapLibrary.sortTokens(input, output);
            uint amountOut = amounts[i + 1];
            if (swapMining != address(0)) {
                ISwapMining(swapMining).swap(msg.sender, input, output, amountOut);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) : (amountOut,
            address to = i < path.length - 2 ? NetswapLibrary.pairFor(factory, output, path[i + 2]) :
            INetswapPair(NetswapLibrary.pairFor(factory, input, output)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = NetswapLibrary.getAmountsOut(factory, amountIn, path, INetswapFactory(factory).feeR
        require(amounts[amounts.length - 1] >= amountOutMin, 'NetswapRouter: INSUFFICIENT_OUTPUT_AMOU
```

Armors Labs

```
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, NetswapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = NetswapLibrary.getAmountsIn(factory, amountOut, path, INetswapFactory(factory).feeR
        require(amounts[0] <= amountInMax, 'NetswapRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, NetswapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }
    function swapExactMetisForTokens(uint amountOutMin, address[] calldata path, address to, uint dea
        external
        virtual
        override
        payable
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[0] == Metis, 'NetswapRouter: INVALID_PATH');
        amounts = NetswapLibrary.getAmountsOut(factory, msg.value, path, INetswapFactory(factory).fee
        require(amounts[amounts.length - 1] >= amountOutMin, 'NetswapRouter: INSUFFICIENT_OUTPUT_AMOU
        // safe transfer metis from router to pair
        TransferHelper.safeTransfer(Metis, NetswapLibrary.pairFor(factory, path[0], path[1]), amounts
        _swap(amounts, path, to);
    }
    function swapTokensForExactMetis(uint amountOut, uint amountInMax, address[] calldata path, addre
        external
        virtual
        override
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == Metis, 'NetswapRouter: INVALID_PATH');
        amounts = NetswapLibrary.getAmountsIn(factory, amountOut, path, INetswapFactory(factory).feeR
        require(amounts[0] <= amountInMax, 'NetswapRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, NetswapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        TransferHelper.safeTransferMetis(to, amounts[amounts.length - 1]);
    }
    function swapExactTokensForMetis(uint amountIn, uint amountOutMin, address[] calldata path, addre
        external
        virtual
        override
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == Metis, 'NetswapRouter: INVALID_PATH');
        amounts = NetswapLibrary.getAmountsOut(factory, amountIn, path, INetswapFactory(factory).feeR
        require(amounts[amounts.length - 1] >= amountOutMin, 'NetswapRouter: INSUFFICIENT_OUTPUT_AMOU
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, NetswapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        TransferHelper.safeTransferMetis(to, amounts[amounts.length - 1]);
    }
```

Armors Labs

```
function swapMetisForExactTokens(uint amountOut, address[] calldata path, address to, uint deadli
    external
    virtual
    override
    payable
    ensure(deadline)
    returns (uint[] memory amounts)
{
    require(path[0] == Metis, 'NetswapRouter: INVALID_PATH');
    amounts = NetswapLibrary.getAmountsIn(factory, amountOut, path, INetswapFactory(factory).feeR
    require(amounts[0] <= msg.value, 'NetswapRouter: EXCESSIVE_INPUT_AMOUNT');
    // safe transfer metis from router to pair
    TransferHelper.safeTransfer(Metis, NetswapLibrary.pairFor(factory, path[0], path[1]), amounts
    _swap(amounts, path, to);
    // refund dust metis, if any
    if (msg.value > amounts[0]) TransferHelper.safeTransferMetis(msg.sender, msg.value - amounts[
}

// **** SWAP (supporting fee-on-transfer tokens) ****
// requires the initial amount to have already been sent to the first pair
function _swapSupportingFeeOnTransferTokens(address[] memory path, address _to) internal virtual
    for (uint i; i < path.length - 1; i++) {
        (address input, address output) = (path[i], path[i + 1]);
        (address token0,) = NetswapLibrary.sortTokens(input, output);
        INetswapPair pair = INetswapPair(NetswapLibrary.pairFor(factory, input, output));
        uint amountInput;
        uint amountOutput;
        { // scope to avoid stack too deep errors
        (uint reserve0, uint reserve1,) = pair.getReserves();
        (uint reserveInput, uint reserveOutput) = input == token0 ? (reserve0, reserve1) : (reser
        amountInput = IERC20Netswap(input).balanceOf(address(pair)).sub(reserveInput);
        amountOutput = NetswapLibrary.getAmountOut(amountInput, reserveInput, reserveOutput, INet
        }
        if (swapMining != address(0)) {
            ISwapMining(swapMining).swap(msg.sender, input, output, amountOutput);
        }
        (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOutput) : (amountO
        address to = i < path.length - 2 ? NetswapLibrary.pairFor(factory, output, path[i + 2]) :
        pair.swap(amount0Out, amount1Out, to, new bytes(0));
    }
}
function swapExactTokensForTokensSupportingFeeOnTransferTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external virtual override ensure(deadline) {
    TransferHelper.safeTransferFrom(
        path[0], msg.sender, NetswapLibrary.pairFor(factory, path[0], path[1]), amountIn
    );
    uint balanceBefore = IERC20Netswap(path[path.length - 1]).balanceOf(to);
    _swapSupportingFeeOnTransferTokens(path, to);
    require(
        IERC20Netswap(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
        'NetswapRouter: INSUFFICIENT_OUTPUT_AMOUNT'
    );
}
function swapExactMetisForTokensSupportingFeeOnTransferTokens(
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
)
    external
    virtual
```

Armors Labs

```
        override
        payable
        ensure(deadline)
    {
        require(path[0] == Metis, 'NetswapRouter: INVALID_PATH');
        uint amountIn = msg.value;
        // safe transfer metis from router to pair
        TransferHelper.safeTransfer(Metis, NetswapLibrary.pairFor(factory, path[0], path[1]), amountI
        uint balanceBefore = IERC20Netswap(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20Netswap(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
            'NetswapRouter: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }
    function swapExactTokensForMetisSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
        external
        virtual
        override
        ensure(deadline)
    {
        require(path[path.length - 1] == Metis, 'NetswapRouter: INVALID_PATH');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, NetswapLibrary.pairFor(factory, path[0], path[1]), amountIn
        );
        _swapSupportingFeeOnTransferTokens(path, address(this));
        uint amountOut = IERC20Netswap(Metis).balanceOf(address(this));
        require(amountOut >= amountOutMin, 'NetswapRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferMetis(to, amountOut);
    }

    // **** LIBRARY FUNCTIONS ****
    function quote(uint amountA, uint reserveA, uint reserveB) public pure virtual override returns (
        return NetswapLibrary.quote(amountA, reserveA, reserveB);
    }

    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut)
        public
        view
        virtual
        override
        returns (uint amountOut)
    {
        return NetswapLibrary.getAmountOut(amountIn, reserveIn, reserveOut, INetswapFactory(factory).
    }

    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut)
        public
        view
        virtual
        override
        returns (uint amountIn)
    {
        return NetswapLibrary.getAmountIn(amountOut, reserveIn, reserveOut, INetswapFactory(factory).
    }

    function getAmountsOut(uint amountIn, address[] memory path)
        public
        view
        virtual
```

Armors Labs

```
        override
        returns (uint[] memory amounts)
    {
        return NetswapLibrary.getAmountsOut(factory, amountIn, path, INetswapFactory(factory).feeRate
    }

    function getAmountsIn(uint amountOut, address[] memory path)
        public
        view
        virtual
        override
        returns (uint[] memory amounts)
    {
        return NetswapLibrary.getAmountsIn(factory, amountOut, path, INetswapFactory(factory).feeRate
    }
}
```

INetswapRouter.sol

```
// Just for reference
pragma solidity >=0.6.2;

interface INetswapRouter {
    function factory() external pure returns (address);
    function Metis() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityMetis(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountMetis, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityMetis(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountMetis);
    function removeLiquidityWithPermit(
        address tokenA,
```

Armors Labs

```
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityMetisWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountToken, uint amountMetis);
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapExactMetisForTokens(uint amountOutMin, address[] calldata path, address to, uint dea
        external
        payable
        returns (uint[] memory amounts);
    function swapTokensForExactMetis(uint amountOut, uint amountInMax, address[] calldata path, addre
        external
        returns (uint[] memory amounts);
    function swapExactTokensForMetis(uint amountIn, uint amountOutMin, address[] calldata path, addre
        external
        returns (uint[] memory amounts);
    function swapMetisForExactTokens(uint amountOut, address[] calldata path, address to, uint deadli
        external
        payable
        returns (uint[] memory amounts);

    function removeLiquidityMetisSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline
    ) external returns (uint amountMetis);
    function removeLiquidityMetisWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountMetisMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountMetis);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
```

Armors Labs

```
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
    function swapExactMetisForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;
    function swapExactTokensForMetisSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;

    function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external view returns (uint
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external view returns (uint
    function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memo
    function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memo
}
```

Multicall.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.5.0;
pragma experimental ABIEncoderV2;

/// @title Multicall - Aggregate results from multiple read-only function calls

interface IERC20 {
    function balanceOf(address user) external view returns (uint256);
}

contract Multicall {
    address public metis = 0xDeadDeAddeAddEAddeadDEaDDEAdDeaDDeAD0000;
    struct Call {
        address target;
        bytes callData;
    }
    function aggregate(Call[] memory calls) public returns (uint256 blockNumber, bytes[] memory retur
        blockNumber = block.number;
        returnData = new bytes[](calls.length);
        for(uint256 i = 0; i < calls.length; i++) {
            (bool success, bytes memory ret) = calls[i].target.call(calls[i].callData);
            require(success);
            returnData[i] = ret;
        }
    }
    // Helper functions
    function getMetisBalance(address addr) public view returns(uint256 balance) {
        balance = IERC20(metis).balanceOf(addr);
    }
    function getCurrentBlockTimestamp() public view returns (uint256 timestamp) {
        timestamp = block.timestamp;
    }
}
```

Armors Labs

NetswapFactory.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.5.0;

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

    function feeTo() external view returns (address);
    function feeRate() external view returns (uint);
    function feeToSetter() external view returns (address);

    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 setFeeRate(uint) external;
    function setFeeToSetter(address) external;
}

pragma solidity =0.6.12;

// a library for performing overflow-safe math, courtesy of DappHub (https://github.com/dapphub/ds-ma

library SafeMathNetswap {
    function add(uint x, uint y) internal pure returns (uint z) {
        require((z = x + y) >= x, 'ds-math-add-overflow');
    }

    function sub(uint x, uint y) internal pure returns (uint z) {
        require((z = x - y) <= x, 'ds-math-sub-underflow');
    }

    function mul(uint x, uint y) internal pure returns (uint z) {
        require(y == 0 || (z = x * y) / y == x, 'ds-math-mul-overflow');
    }
}

pragma solidity =0.6.12;


contract NetswapERC20 {
    using SafeMathNetswap for uint;

    string public constant name = 'Netswap LP Token';
    string public constant symbol = 'NLP';
    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 {
```

Armors Labs

```
            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, 'Netswap: 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, 'Netswap: INVALID_SIGNAT
        _approve(owner, spender, value);
```

Armors Labs

```
    }
}

pragma solidity =0.6.12;

// a library for performing various math operations

library Math {
    function min(uint x, uint y) internal pure returns (uint z) {
        z = x < y ? x : y;
    }

    // babylonian method (https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_
    function sqrt(uint y) internal pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }
}

pragma solidity =0.6.12;

// 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);
    }
}

pragma solidity >=0.5.0;

interface IERC20Netswap {
    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);
}
```

Armors Labs

```
pragma solidity >=0.5.0;

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

pragma solidity =0.6.12;

contract NetswapPair is NetswapERC20 {
    using SafeMathNetswap  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, 'Netswap: 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))), 'Netswap: 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, 'Netswap: FORBIDDEN'); // sufficient check
        token0 = _token0;
```

Armors Labs

```
        token1 = _token1;
    }

    // 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), 'Netswap: 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 = INetswapFactory(factory).feeTo();
        feeOn = feeTo != address(0);
        uint _kLast = kLast; // gas savings
        if (feeOn) {
            if (_kLast != 0) {
                uint rootK = Math.sqrt(uint(_reserve0).mul(_reserve1));
                uint rootKLast = Math.sqrt(_kLast);
                if (rootK > rootKLast) {
                    uint numerator = totalSupply.mul(rootK.sub(rootKLast));
                    uint denominator = rootK.mul(5).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 = IERC20Netswap(token0).balanceOf(address(this));
        uint balance1 = IERC20Netswap(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 u
        if (_totalSupply == 0) {
            liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
           _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY toke
        } else {
            liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _
        }
        require(liquidity > 0, 'Netswap: 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) {
```

Armors Labs

```
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        address _token0 = token0;                                // gas savings
        address _token1 = token1;                                // gas savings
        uint balance0 = IERC20Netswap(_token0).balanceOf(address(this));
        uint balance1 = IERC20Netswap(_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 u
        amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribu
        amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribu
        require(amount0 > 0 && amount1 > 0, 'Netswap: INSUFFICIENT_LIQUIDITY_BURNED');
        _burn(address(this), liquidity);
        _safeTransfer(_token0, to, amount0);
        _safeTransfer(_token1, to, amount1);
        balance0 = IERC20Netswap(_token0).balanceOf(address(this));
        balance1 = IERC20Netswap(_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, 'Netswap: INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'Netswap: 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, 'Netswap: 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) INetswapCallee(to).netswapCall(msg.sender, amount0Out, amount1Out, data)
        balance0 = IERC20Netswap(_token0).balanceOf(address(this));
        balance1 = IERC20Netswap(_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, 'Netswap: INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
        uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(INetswapFactory(factory).feeRate
        uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(INetswapFactory(factory).feeRate
        require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**2)
        }

        _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, IERC20Netswap(_token0).balanceOf(address(this)).sub(reserve0));
        _safeTransfer(_token1, to, IERC20Netswap(_token1).balanceOf(address(this)).sub(reserve1));
    }

    // force reserves to match balances
    function sync() external lock {
        _update(IERC20Netswap(token0).balanceOf(address(this)), IERC20Netswap(token1).balanceOf(addre
    }
```

Armors Labs

```
}

pragma solidity =0.6.12;



contract NetswapFactory is INetswapFactory {
    using SafeMathNetswap  for uint;
    uint public override feeRate = 3;
    address public override feeTo;
    address public override feeToSetter;

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

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

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

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

    function pairCodeHash() external pure returns (bytes32) {
        return keccak256(type(NetswapPair).creationCode);
    }

    function createPair(address tokenA, address tokenB) external override returns (address pair) {
        require(tokenA != tokenB, 'Netswap: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'Netswap: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'Netswap: PAIR_EXISTS'); // single check is su
        bytes memory bytecode = type(NetswapPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        NetswapPair(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 override {
        require(msg.sender == feeToSetter, 'Netswap: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeRate(uint _feeRate) external override {
        require(msg.sender == feeToSetter, 'Netswap: FORBIDDEN');
        feeRate = _feeRate;
    }

    function setFeeToSetter(address _feeToSetter) external override {
        require(msg.sender == feeToSetter, 'Netswap: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }

}
```

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.

Armors Labs

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 specifers 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

Armors Labs

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.

Armors Labs

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.

Armors Labs

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.

Armors Labs

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