Armors Labs

FIDO

Smart Contract Audit

Armors Labs

FIDO Audit Summary

Project name : FIDO Contract

Project address: None

Code URL : https://github.com/fido-project/audit-contracts

Commit : 4f5387c53123c928fc7207d6b0af03179ae1c46a

Project target : FIDO Contract Audit

Blockchain : Huobi ECO Chain（Heco）

Test result : PASSED

Audit Info

Audit NO : 0X202105080008

Audit Team : Armors Labs

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

FIDO Audit

The FIDO team asked us to review and audit their FIDO 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
FIDO Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-05-08

Armors Labs

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 FIDO contract. The above should not be construed as investment advice.

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

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

Audited target file

file	md5
./ReentrancyGuard.sol	f3c1a971bfe306e3330870e4c918b631
./IDOInfo.sol	3d07082e3aba519265ac5604626868a5
./ERC20/ERC20Pausable.sol	5ba41b496341f26393f62f30572c037a
./ERC20/ERC20Mintable.sol	eca19f6cccd0fe57e66a29e93f22a728
./ERC20/ERC20Burnable.sol	0dca068ad242853e73f0cf501f9e3cdd
./ERC20/ERC20.sol	ebf1cff6a039a54f1a0568b38bc0e0f9
./StakeRewardPerBlock.sol	24746a23ec6711d1915c2d96629ee151
./IDOToken.sol	18807f36832c6b409f7b1bceb1e8d151
./StakeLPRewardPerDay.sol	587c5571ac1a36b2d408401628f95e41
./StakeTokenPool.sol	1d2e5278917ba95c720058df654959f1
./IDOFactory.sol	33d95bba0b9326602a0a95945969b408
./MFILPool.sol	17a98a57f950f51cd56002a221ec8dbf
./MFIL-IDOToken-Factory.sol	d252b31c0d8a2a7094b265ef796d21f0
./oracle/RateOracle.sol	7dc71dd75865bd11c522b0188dc15bf3
./FidoMargin.sol	2cce4a0c81f44ba5b62578cdee70e1b2
./Pausable.sol	e70cfad9554608935e5da690a1e8c81a
./libraries/TransferHelper.sol	9bf500f7d995b8348c9b5dcb3aa24312
./libraries/SafeMath.sol	e03e12206057e809eb76c5f681170c32
./Context.sol	2adbd82f6d055a4751566d4671512b03
./StakeLPRewardPerBlock.sol	6354cdbf357428bd5ecdd8a9898de912
./MFIL-IDOToken.sol	0042ccfe460baba7fd36e2222f8391df
./StakeRewardPerDay.sol	d7634f0eb3bad2ca0e8959e0f8852de7
./Ownable.sol	3c73ff1bfb400374dd48f30345945264
./MFIL.sol	eb7df44eaeecc07d4828f02aa0ce25d4
./FIDO-USDT.sol	60ab120afbc00261f1a67997ba7377f9
./FidoMember.sol	36df7526bede4bc66945dc2b74d8e160
./Migrations.sol	50b3b1dc92806dc8f604fc8c5c65518f
./FIDO.sol	943ff11ac96ea389da3b89c7591e404f
./IDOUserRouter.sol	ca9d18b7c322c91091e4b1be48f62bf5
./MdexRouter.sol	3645059c5fd05fed8f8b249fa9151da3

Armors Labs

file	md5
./interfaces/IIDOInfo.sol	ce29173dcd612f77336cfba16b621e56
./interfaces/IMdexFactory.sol	13e2c61c92e70f81275a984181fd7176
./interfaces/IFidoUsdtLPPool.sol	1ac8e832f3b8811c5d89bf8a72ad44df
./interfaces/IHFIL.sol	a2214ca7300b49e37218b1494834dcb6
./interfaces/IIDOToken.sol	4b4f9fa3bd54bf924a559014dd8450f3
./interfaces/IMFIL.sol	fb358aad558e600d8c283701e8778b2c
./interfaces/IRateOracle.sol	2d8b9715941c9d7582ed3553d28e55c2
./interfaces/IMdexPair.sol	082fc325db03f7698928f5afa7f9bf54
./interfaces/IERC20.sol	e0a41531d159d3a32f84b7a3ecf9fabb
./interfaces/IERC20Mintable.sol	ceb9c3b25228976d6e5ee7328ceff899
./interfaces/IFidoMember.sol	1e7368dbb56892fe14261b24abc58b2e

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
Unchecked CALL Return Values	safe

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

Contract file

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

interface IFidoMember {
  function baseRate() view external returns (uint256);

  function fidoOfficial() view external returns (address);

  function followers(address, uint256) view external returns (address);

  function inviter(address) view external returns (address);

  function isInvited(address) view external returns (bool);

  function isMember(address) view external returns (bool);

  function joinBlockHeight(address) view external returns (uint256);

  function mfilPool() view external returns (address);

  function multiRate() view external returns (uint256);

  function operator() view external returns (address);

  function owner() view external returns (address);

  function pool() view external returns (address);

  function rateDecimal() view external returns (uint256);

  function renounceOwnership() external;

  function stakeMin() view external returns (uint256);

  function transferOwnership(address newOwner) external;

  function transferOperatorship(address newOperator) external;

  function adjustRate(uint256 _baseRate, uint256 _multiRate) external;

  function changeRateDecimal(uint256 newRateDecimal) external;
```

Armors Labs

```
  function changeStakeMin(uint256 newStakeMin) external;

  function joinFido(address _inviter) external;

  function changeMFILPool(address newMfilpool) external;

  function changeFidoOfficial(address newFidoOfficial) external;

  function changePool(address newPool) external;

  function getFollowerCount(address member) view external returns (uint256);

  function caleInviteRate(address member) view external returns (address inviter0, address inviter1,

  event AdjustRate(uint256 baseRate, uint256 multiRate);

  event AdjustStakeMin(uint256 stakeMin);

  event NewMember(address indexed member, address indexed inviter, uint256 joinBlockHeight);

  event OperatorshipTransferred(address indexed previousOperator, address indexed newOperator);

  event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
}// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20Mintable {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount)
        external
        returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender)
        external
        view
        returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
```

Armors Labs

```
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external returns (bool);

    function mint(address recipient, uint256 amount) external;

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );

    event AddMinter(address indexed minter);
    event RemoveMinter(address indexed minter);
    event Mint(
        address indexed minter,
        address indexed recipient,
        uint256 amount
    );
}
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
```

Armors Labs

```
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
```

Armors Labs

```
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;
interface IMdexPair {
    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);
```

Armors Labs

```
    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;
}// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

interface IRateOracle {
  function owner() view external returns (address);

  function paused() view external returns (bool);

  function renounceOwnership() external;

  function requester() view external returns (address);

  function transferOwnership(address newOwner) external;

  function updater() view external returns (address);

  function pause() external;

  function unPause() external;

  function status() view external returns (bool);

  function lastRequestTime() view external returns (uint256);

  function lastUpdateTime() view external returns (uint256);

  function decimals() view external returns (uint8);

  function rate() view external returns (uint256);

  function request() external;

  function update(uint256 rate_) external;

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

  event Paused(address account);

  event Request(uint256 indexed timestamp);

  event RequestershipTransferred(address indexed previousRequester, address indexed newRequester);

  event Unpaused(address account);

  event Update(uint256 indexed timestamp, uint256 rate);

  event UpdatershipTransferred(address indexed previousUpdater, address indexed newUpdater);
}// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

interface IMFIL {
  function allowance(address owner, address spender) view external returns (uint256);

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

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

Armors Labs

```
  function burn(uint256 amount) external;

  function burnFrom(address account, uint256 amount) external;

  function decimals() view external returns (uint8);

  function decreaseAllowance(address spender, uint256 subtractedValue) external returns (bool);

  function increaseAllowance(address spender, uint256 addedValue) external returns (bool);

  function isMinter(address) view external returns (bool);

  function mint(address recipient, uint256 amount) external;

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

  function operator() view external returns (address);

  function owner() view external returns (address);

  function paused() view external returns (bool);

  function renounceOwnership() external;

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

  function totalSupply() view external returns (uint256);

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

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

  function transferOwnership(address newOwner) external;

  function transferOperatorship(address newOperator) external;

  function pause() external;

  function unPause() external;

  function addMinter(address minter) external;

  function removeMinter(address minter) external;

  function cap() view external returns (uint256);

  event AddMinter(address indexed minter);

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

  event Mint(address indexed minter, address indexed recipient, uint256 amount);

  event OperatorshipTransferred(address indexed previousOperator, address indexed newOperator);

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

  event Paused(address account);

  event RemoveMinter(address indexed minter);

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

  event Unpaused(address account);
}// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;
```

Armors Labs

```
interface IIDOToken {
  function allowance(address owner, address spender) view external returns (uint256);

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

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

  function burn(uint256 amount) external;

  function burnFrom(address account, uint256 amount) external;

  function dailySaleCap() view external returns (uint256);

  function decimals() view external returns (uint8);

  function decreaseAllowance(address spender, uint256 subtractedValue) external returns (bool);

  function factory() view external returns (address);

  function gasPrice() view external returns (uint256);

  function hardDrivePrice() view external returns (uint256);

  function hfil() view external returns (address);

  function hfilRecipient() view external returns (address);

  function idoEndTime() view external returns (uint256);

  function idoStartTime() view external returns (uint256);

  function idoStatus() view external returns (bool);

  function increaseAllowance(address spender, uint256 addedValue) external returns (bool);

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

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

  function paused() view external returns (bool);

  function price() view external returns (uint256);

  function router() view external returns (address);

  function sealPrice() view external returns (uint256);

  function sender() view external returns (address);

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

  function todayStartTime() view external returns (uint256);

  function totalSupply() view external returns (uint256);

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

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

  function operator() view external returns (address);

  function setPrice(uint256 gasPrice_, uint256 sealPrice_, uint256 hardDrivePrice_) external;

  function setTimes(uint256 idoStartTime_, uint256 idoEndTime_) external;
```

Armors Labs

```
  function changerouter(address _router) external;

  function stopIdo() external;

  function stopIdoBySender() external;

  function todayRemaind() view external returns (uint256);

  function pause() external;

  function unPause() external;

  function cap() view external returns (uint256);

  function startIdo(address _router) external;

  function ido(address recipient) external returns (uint256 amount);

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

  event IDO(address indexed to, uint256 cost, uint256 amount);

  event IDOStart(uint256 timestamp);

  event IDOStop(uint256 timestamp, address sender);

  event Paused(address account);

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

  event Unpaused(address account);
}// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

interface IHFIL {
  function allowance(address owner, address spender) view external returns (uint256);

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

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

  function burn(address account, uint256 amount) external;

  function changeUser(address new_operator, address new_pauser) external;

  function decimals() view external returns (uint8);

  function decreaseAllowance(address spender, uint256 subtractedValue) external returns (bool);

  function increaseAllowance(address spender, uint256 addedValue) external returns (bool);

  function mint(address account, uint256 amount) external;

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

  function pause() external;

  function paused() view external returns (bool);

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

  function totalSupply() view external returns (uint256);

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

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

Armors Labs

```
  function unpause() external;

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

  event Paused(address account);

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

  event Unpaused(address account);
}// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

interface IFidoUsdtLPPool {
  function BlastUpdateBlock() view external returns (uint256);

  function BrewardPreBlock() view external returns (uint256);

  function BrewardRate() view external returns (uint256);

  function BuserRate(address) view external returns (uint256);

  function BuserReward(address) view external returns (uint256);

  function adjustReward(uint256 amount) external;

  function caleLiquidity(uint256 amountADesired, uint256 amountBDesired, uint256 amountAMin, uint256

  function caleReward() view external returns (uint256 reward);

  function factory() view external returns (address);

  function fido() view external returns (address);

  function getStake() view external returns (uint256 amount);

  function halve() external returns (uint256 reward);

  function operator() view external returns (address);

  function owner() view external returns (address);

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

  function pause() external;

  function paused() view external returns (bool);

  function renounceOwnership() external;

  function rewardToken() view external returns (address);

  function stake(uint256 amountAIn, uint256 amountBIn) external returns (uint256 amountA, uint256 amo

  function stakeRate() view external returns (uint8);

  function tokenA() view external returns (address);

  function tokenB() view external returns (address);

  function totalStake() view external returns (uint256);

  function totalTokenBStake() view external returns (uint256);

  function transferOperatorship(address newOperator) external;
```

Armors Labs

```
    function transferOwnership(address newOwner) external;

    function unPause() external;

    function unStake() external returns (uint256 reward, uint256 amountA, uint256 amountB, uint256 liqu

    function usdt() view external returns (address);

    function userStake(address) view external returns (uint256);

    function userTokenBStake(address) view external returns (uint256);

    event NewReward(uint256 blockHeight, uint256 amount);

    event OperatorshipTransferred(address indexed previousOperator, address indexed newOperator);

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

    event Paused(address account);

    event Reward(address indexed receiver, uint256 reward);

    event Stake(address indexed sender, uint256 amountA, uint256 amountB, uint256 liquidity);

    event UnStake(address indexed receiver, uint256 amountA, uint256 amountB, uint256 liquidity);

    event Unpaused(address account);
}// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

interface IMdexFactory {
    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
```

Armors Labs

```
    function getAmountOut(uint256 amountIn, uint256 reserveIn, uint256 reserveOut) external view retu

    function getAmountIn(uint256 amountOut, uint256 reserveIn, uint256 reserveOut) external view retu

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

    function getAmountsIn(uint256 amountOut, address[] calldata path) external view returns (uint256[
}// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

interface IIDOInfo {
  function IDOList(uint256) view external returns (address);

  function exist(address) view external returns (bool);

  function isPool(address) view external returns (bool);

  function operator() view external returns (address);

  function owner() view external returns (address);

  function payer2pool(address) view external returns (address);

  function pool2idoToken(address) view external returns (address);

  function renounceOwnership() external;

  function stakeAddress(address) view external returns (address);

  function transferOwnership(address newOwner) external;

  function transferOperatorship(address newOperator) external;

  function IDOListCount() view external returns (uint256);

  function addIDO(address idoToken) external;

  function setPayer(address pool, address payer) external;

  function setStakeAddress(address idoToken, address _stakeAddress) external;

  event AddIDO(uint256 timestamp, address idoToken);

  event OperatorshipTransferred(address indexed previousOperator, address indexed newOperator);

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

  event SetPayer(uint256 timestamp, address pool, address payer);

  event SetStakeAddress(uint256 timestamp, address idoToken, address _stakeAddress);
}// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./interfaces/IERC20.sol";
import "./interfaces/IMdexFactory.sol";
import "./interfaces/IMdexPair.sol";

import "./libraries/SafeMath.sol";
import "./libraries/TransferHelper.sol";

contract MdexRouter {
    using SafeMath for uint256;

    address public factory;
```

Armors Labs

```
    constructor(address _factory) {
        factory = _factory;
    }

    function pairFor(address tokenA, address tokenB)
        public
        view
        returns (address pair)
    {
        pair = IMdexFactory(factory).pairFor(tokenA, tokenB);
    }

    // **** ADD LIQUIDITY ****
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin
    ) internal virtual returns (uint256 amountA, uint256 amountB) {
        // create the pair if it doesn't exist yet
        if (IMdexFactory(factory).getPair(tokenA, tokenB) == address(0)) {
            IMdexFactory(factory).createPair(tokenA, tokenB);
        }
        (uint256 reserveA, uint256 reserveB) =
            IMdexFactory(factory).getReserves(tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint256 amountBOptimal =
                IMdexFactory(factory).quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(
                    amountBOptimal >= amountBMin,
                    "MdexRouter: INSUFFICIENT_B_AMOUNT"
                );
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint256 amountAOptimal =
                    IMdexFactory(factory).quote(
                        amountBDesired,
                        reserveB,
                        reserveA
                    );
                assert(amountAOptimal <= amountADesired);
                require(
                    amountAOptimal >= amountAMin,
                    "MdexRouter: INSUFFICIENT_A_AMOUNT"
                );
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin
    )
        internal
        virtual
        returns (
```

Armors Labs

```
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        )
    {
        (amountA, amountB) = _addLiquidity(
            tokenA,
            tokenB,
            amountADesired,
            amountBDesired,
            amountAMin,
            amountBMin
        );
        address pair = pairFor(tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IMdexPair(pair).mint(address(this));
    }

    // **** REMOVE LIQUIDITY ****
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to
    ) internal virtual returns (uint256 amountA, uint256 amountB) {
        address pair = pairFor(tokenA, tokenB);
        IMdexPair(pair).transfer(pair, liquidity);
        // send liquidity to pair
        (uint256 amount0, uint256 amount1) = IMdexPair(pair).burn(to);
        (address token0, ) = IMdexFactory(factory).sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0
            ? (amount0, amount1)
            : (amount1, amount0);
        require(amountA >= amountAMin, "MdexRouter: INSUFFICIENT_A_AMOUNT");
        require(amountB >= amountBMin, "MdexRouter: INSUFFICIENT_B_AMOUNT");
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./Ownable.sol";
import "./Pausable.sol";
import "./libraries/SafeMath.sol";
import "./libraries/TransferHelper.sol";
import "./interfaces/IIDOToken.sol";
import "./interfaces/IFidoMember.sol";
import "./ReentrancyGuard.sol";

contract IDOUserRouter is Context, Ownable, Pausable, ReentrancyGuard {
    using SafeMath for uint256;

    uint256 public fidoFeeRate = 1; // fidoFee = fidoFeeRate / 10**fidoFeeRateDecimals
    uint8 public fidoFeeRateDecimals = 2;

    uint256 public insuranceFeeRate = 1; // insuranceFee = insuranceFeeRate / 10**insuranceFeeRateDec
    uint8 public insuranceFeeRateDecimals = 3;

    uint256 public inviteFeeRate = 10; // inviteFee = inviteFeeRate / 10**inviteFeeRateDecimals
    uint8 public inviteFeeRateDecimals = 2;

    address public operator;
```

Armors Labs

```
    mapping(address => bool) public approvedIDO;
    mapping(address => mapping(address => bool)) public userIDO;
    mapping(address => address[]) public userIDOs;
    address[] public approvedIDOlist;
    address public hfil;
    address public insuranceProvider;
    address public fidoFeeRecipient;
    address public FidoMember;

    event OperatorshipTransferred(
        address indexed previousOperator,
        address indexed newOperator
    );

    event ApproveIDO(uint256 timestamp, address indexed idoToken);
    event StopIDO(uint256 timestamp, address indexed idoToken);

    event IDO(
        address indexed sender,
        address indexed idoToken,
        uint256 hfil,
        uint256 amount,
        uint256 fidoFee,
        uint256 insurance,
        uint256 inviteFee
    );

    constructor(
        address _operator,
        address _hfil,
        address _FidoMember,
        address _insuranceProvider,
        address _fidoFeeRecipient
    ) Pausable() ReentrancyGuard() {
        operator = _operator;
        emit OperatorshipTransferred(address(0), operator);
        hfil = _hfil;
        FidoMember = _FidoMember;
        insuranceProvider = _insuranceProvider;
        fidoFeeRecipient = _fidoFeeRecipient;
    }

    modifier onlyOperator() {
        require(
            _msgSender() == operator,
            "Operable: caller is not the operator"
        );
        _;
    }

    function transferOperatorship(address newOperator) external onlyOwner {
        require(
            newOperator != address(0),
            "Operable: new operator is the zero address"
        );
        emit OperatorshipTransferred(operator, newOperator);
        operator = newOperator;
    }

    function pause() external onlyOperator {
        _pause();
    }

    function unPause() external onlyOperator {
        _unpause();
    }
```

Armors Labs

```
function changeFidoFeeRecipient(address _fidoFeeRecipient)
    external
    onlyOwner
{
    require(_fidoFeeRecipient != address(0), "wrong address");
    fidoFeeRecipient = _fidoFeeRecipient;
}

function changeInsuranceProvider(address _insuranceProvider)
    external
    onlyOwner
{
    require(_insuranceProvider != address(0), "wrong address");
    insuranceProvider = _insuranceProvider;
}

function changeFidoFeeRate(uint256 _fidoFeeRate, uint8 _fidoFeeRateDecimals)
    external
    onlyOperator
{
    require(
        _fidoFeeRateDecimals > 0,
        "fidoFeeRateDecimals must greater than zero"
    );
    fidoFeeRateDecimals = _fidoFeeRateDecimals;
    fidoFeeRate = _fidoFeeRate;
}

function changeInsuranceFeeRate(
    uint256 _insuranceFeeRate,
    uint8 _insuranceFeeRateDecimals
) external onlyOperator {
    require(
        _insuranceFeeRateDecimals > 0,
        "insuranceFeeRateDecimals must greater than zero"
    );
    insuranceFeeRateDecimals = _insuranceFeeRateDecimals;
    insuranceFeeRate = _insuranceFeeRate;
}

function changeInviteFeeRate(
    uint256 _inviteFeeRate,
    uint8 _inviteFeeRateDecimals
) external onlyOperator {
    require(
        _inviteFeeRateDecimals > 0,
        "inviteFeeRateDecimals must greater than zero"
    );
    inviteFeeRateDecimals = _inviteFeeRateDecimals;
    inviteFeeRate = _inviteFeeRate;
}

function userIDOsCount(address user) external view returns (uint256) {
    return userIDOs[user].length;
}

function approvedIDOlistCount() external view returns (uint256) {
    return approvedIDOlist.length;
}

function approveIDO(address idoToken) external onlyOperator {
    require(
        idoToken != address(0),
        "IDORouter: IDO address cannot be zero"
    );
```

Armors Labs

```
        require(!approvedIDO[idoToken], "IDORouter: IDO approved");
        approvedIDO[idoToken] = true;
        approvedIDOlist.push(idoToken);
        emit ApproveIDO(block.timestamp, idoToken);
    }

    function stopIDO(address idoToken) external onlyOperator {
        require(
            idoToken != address(0),
            "IDORouter: IDO address cannot be zero"
        );
        require(approvedIDO[idoToken], "IDORouter: IDO not approved");
        approvedIDO[idoToken] = false;
        for (uint256 index = 0; index < approvedIDOlist.length; index++) {
            if (approvedIDOlist[index] == idoToken) {
                approvedIDOlist[index] = approvedIDOlist[
                    approvedIDOlist.length - 1
                ];
                break;
            }
        }
        approvedIDOlist.pop();
        emit StopIDO(block.timestamp, idoToken);
    }

    function getIDOPrice(address idoToken) internal view returns (uint256) {
        return IIDOToken(idoToken).price();
    }

    function ido(address idoToken, uint256 idoAmount) external nonReentrant {
        require(approvedIDO[idoToken], "IDORouter: IDO not approved");
        require(idoAmount > 10**4, "IDORouter: IDO amount less than 0.01");

        uint256 price = getIDOPrice(idoToken);
        uint256 totalprice = price.mul(idoAmount).div(10**6);
        TransferHelper.safeTransferFrom(
            hfil,
            _msgSender(),
            idoToken,
            totalprice
        );
        idoAmount = IIDOToken(idoToken).ido(_msgSender());
        totalprice = price.mul(idoAmount).div(10**6);

        uint256 fidoFee =
            totalprice.mul(fidoFeeRate).div(10**uint256(fidoFeeRateDecimals));
        TransferHelper.safeTransferFrom(
            hfil,
            _msgSender(),
            fidoFeeRecipient,
            fidoFee
        );

        uint256 insurance =
            totalprice.mul(insuranceFeeRate).div(
                10**uint256(insuranceFeeRateDecimals)
            );
        TransferHelper.safeTransferFrom(
            hfil,
            _msgSender(),
            insuranceProvider,
            insurance
        );

        uint256 inviteFee =
            totalprice.mul(inviteFeeRate).div(
```

Armors Labs

```
                10**uint256(inviteFeeRateDecimals)
            );
        {
            address inviter0;
            address inviter1;
            address fido;
            uint256 rate0;
            uint256 rate1;
            uint256 fidoRate;
            uint256 fee;
            (inviter0, inviter1, fido, rate0, rate1, fidoRate) = IFidoMember(
                FidoMember
            )
                .caleInviteRate(_msgSender());
            fee = inviteFee.mul(rate0).div(
                10**IFidoMember(FidoMember).rateDecimal()
            );
            TransferHelper.safeTransferFrom(hfil, _msgSender(), inviter0, fee);
            fee = inviteFee.mul(rate1).div(
                10**IFidoMember(FidoMember).rateDecimal()
            );
            TransferHelper.safeTransferFrom(hfil, _msgSender(), inviter1, fee);
            fee = inviteFee.mul(fidoRate).div(
                10**IFidoMember(FidoMember).rateDecimal()
            );
            TransferHelper.safeTransferFrom(hfil, _msgSender(), fido, fee);
        }
        if (idoAmount > 0) {
            if (!userIDO[_msgSender()][idoToken]) {
                userIDO[_msgSender()][idoToken] = true;
                userIDOs[_msgSender()].push(idoToken);
            }
        }
        emit IDO(
            _msgSender(),
            idoToken,
            totalprice,
            idoAmount,
            fidoFee,
            insurance,
            inviteFee
        );
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./ERC20/ERC20.sol";
import "./ERC20/ERC20Burnable.sol";
import "./ERC20/ERC20Mintable.sol";
import "./Ownable.sol";

contract FIDO is ERC20Mintable, ERC20Burnable, Ownable {
    using SafeMath for uint256;

    address public operator;

    uint256 private _cap = 210000000 * 10**18;
    uint256 public shareRateDecimal = 4; // 10000
    uint8 public totalReleaseWeek = 25;
    uint8 public releasedWeek = 0;
    uint256 public lastRealeaseTime = 0;
    address[] public releaseRecipient;
    mapping(address => uint256) public releaseShareRate;
    event Release(uint256 timestamp, uint8 releaseCount);
```

Armors Labs

```
    event OperatorshipTransferred(
        address indexed previousOperator,
        address indexed newOperator
    );

    constructor(address _operator) ERC20("FIDO", "FIDO") Pausable() Ownable() {
        operator = _operator;
        emit OperatorshipTransferred(address(0), operator);
        _setupDecimals(18);
        addReleaseRecipient(0x9Dafc698200B9Bb509612ae39f007855d1c26B3D, 605);
        addReleaseRecipient(0xAf9736eC4814a2947B7B64354c612A41Be518b9f, 110);
        addReleaseRecipient(0x6fc89Ac788A31880f020AabA39dB49e05D803670, 100);
        addReleaseRecipient(0x6B8A58B626dE1Aa35b9E24d4e9012E33d7084CD5, 100); // 1%
        addReleaseRecipient(0xeb8d2AC91A66b8A7790c808a5B172E647a81103a, 60);
        addReleaseRecipient(0x90d5111f4C736Ac4A0B0f90589149cD25A894537, 57);
        addReleaseRecipient(0xF3DFAcED2aE482473BEC9Ab00863C51B132e8169, 50);
        addReleaseRecipient(0x78CfF87757fbE3a18a23d28A3Ad216b5F1d26a7F, 40);
        addReleaseRecipient(0xC6D294310A8D9946c458dD886A51EFA3DA04593f, 40);
        addReleaseRecipient(0x6CA91d3f8675D83bBEf4bb3C522CF75b1B4AAD1C, 40);
        addReleaseRecipient(0xA73558c94cBDB42eF189E68C9D993D252f6F252B, 40);
        addReleaseRecipient(0xE9088899D6b8b1082A051d6bC06AD2B6e7AECe31, 40);
        addReleaseRecipient(0xf6c062b71344650e0A23Ff7D10e00842147e5e21, 30);
        addReleaseRecipient(0xf1c0091B3eFEC6e621E285b6a980734b9C437b85, 30);
        addReleaseRecipient(0x3Fc3f768c5eDC881690b33292F499964917b8189, 30);
        addReleaseRecipient(0x71192f0df65a58982fFA4e413296389f98c853B7, 20);
        addReleaseRecipient(0xf0f9b88B3e66D61469DB584712B52372C1e55E69, 15);
        addReleaseRecipient(0xeb84d5762ba1A68c3d0723518D12Ad417Df49363, 15);
        addReleaseRecipient(0xB2c04a3B20B5a498bcAE1576900A000971EDb6Dd, 10);
        addReleaseRecipient(0x52abb7EC70685b0C484F3fbc20Cce473A6c00dD3, 10);
        addReleaseRecipient(0x1cBb40e2137741Dbf6A1aFc1fa0a95fB1016b30f, 10);
        addReleaseRecipient(0xBe19c8eCf41a3F38664827C89aAD87f16dF3De38, 10);
        addReleaseRecipient(0x1F6361D6907897610355853388 26E4F89cbA9a44, 10); // 0.1%
        addReleaseRecipient(0xe674816FD0C0e4062B7e43dC72c34cb3023fB825, 10);
        addReleaseRecipient(0x6924F8E39623a1f5aA776637d2008Ad1c4e16598, 5);
        addReleaseRecipient(0x979cAE9260C799E73cd320936b5c5A902D291636, 5);
        addReleaseRecipient(0x7Facf41272d5a8c490Cb79CfE84981169259d935, 5);
        addReleaseRecipient(0x0299386481015Ce66FC3818DE9E7d5302FFf5278, 3);
    }

    modifier onlyOperator() {
        require(
            _msgSender() == operator,
            "Operable: caller is not the operator"
        );
        _;
    }

    function transferOperatorship(address newOperator) external onlyOwner {
        require(
            newOperator != address(0),
            "Operable: new operator is the zero address"
        );
        emit OperatorshipTransferred(operator, newOperator);
        operator = newOperator;
    }

    function pause() external onlyOperator {
        _pause();
    }

    function unPause() external onlyOperator {
        _unpause();
    }

    function addMinter(address minter) external onlyOperator {
        _addMinter(minter);
```

Armors Labs

```
    }

    function removeMinter(address minter) external onlyOperator {
        _removeMinter(minter);
    }

    function addReleaseRecipient(address recipient, uint256 shareRate)
        internal
    {
        require(releaseShareRate[recipient] == 0, "already a releaseRecipient");
        releaseRecipient.push(recipient);
        releaseShareRate[recipient] = shareRate;
    }

    function releaseShare() external onlyOperator() {
        require(
            block.timestamp - lastRealeaseTime > 1 weeks,
            "FIDO: see you next week"
        );
        require(
            releasedWeek < totalReleaseWeek,
            "FIDO: no share needs to release"
        );
        uint256 amount;
        for (uint256 index = 0; index < releaseRecipient.length; index++) {
            amount = releaseShareRate[releaseRecipient[index]]
                .mul(_cap)
                .div(10**shareRateDecimal)
                .div(totalReleaseWeek);
            _mint(releaseRecipient[index], amount);
        }
        releasedWeek += 1;
        lastRealeaseTime = block.timestamp;
        emit Release(block.timestamp, releasedWeek);
    }

    /**
     * @dev Returns the cap on the token's total supply.
     */
    function cap() public view virtual returns (uint256) {
        return _cap;
    }

    /**
     * @dev See {ERC20-_beforeTokenTransfer}.
     *
     * Requirements:
     *
     * - minted tokens must not cause the total supply to go over the cap.
     */
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
        super._beforeTokenTransfer(from, to, amount);

        if (from == address(0)) {
            // When minting tokens
            require(totalSupply().add(amount) <= cap(), "ERC20: cap exceeded");
        }
    }
}
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;
```

Armors Labs

```
contract Migrations {
  address public owner = msg.sender;
  uint public last_completed_migration;

  modifier restricted() {
    require(
      msg.sender == owner,
      "This function is restricted to the contract's owner"
    );
    _;
  }

  function setCompleted(uint completed) public restricted {
    last_completed_migration = completed;
  }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./Ownable.sol";
import "./libraries/SafeMath.sol";
import "./interfaces/IFidoUsdtLPPool.sol";

interface IMFILPool {
    function userStake(address) external view returns (uint256);
}

contract FidoMember is Ownable {
    using SafeMath for uint256;
    address public operator;

    address public fidoOfficial;
    address public pool;
    address public mfilPool;

    // invite0Rate = baseRate + multiRate x (invite0 / (invite0 + invite1)) %
    // invite1Rate = 100 - invite0Rate %
    uint256 public baseRate = 40;
    uint256 public multiRate = 40;
    uint256 public rateDecimal = 2; // 100
    uint256 public stakeMin = 0;

    mapping(address => bool) public isMember;
    mapping(address => bool) public isInvited;
    mapping(address => address) public inviter;
    mapping(address => uint256) public joinBlockHeight;
    mapping(address => address[]) public followers;

    event NewMember(
        address indexed member,
        address indexed inviter,
        uint256 joinBlockHeight
    );

    event OperatorshipTransferred(
        address indexed previousOperator,
        address indexed newOperator
    );

    event AdjustRate(uint256 baseRate, uint256 multiRate);

    event AdjustStakeMin(uint256 stakeMin);

    constructor(
        address _operator,
```

Armors Labs

```
        address _fidoOfficial,
        address _pool
    ) Ownable() {
        operator = _operator;
        emit OperatorshipTransferred(address(0), operator);
        fidoOfficial = _fidoOfficial;
        pool = _pool;
    }

    modifier onlyOperator() {
        require(
            _msgSender() == operator,
            "Operable: caller is not the operator"
        );
        _;
    }

    function transferOperatorship(address newOperator) external onlyOwner {
        require(
            newOperator != address(0),
            "Operable: new operator is the zero address"
        );
        emit OperatorshipTransferred(operator, newOperator);
        operator = newOperator;
    }

    function adjustRate(uint256 _baseRate, uint256 _multiRate)
        external
        onlyOperator
    {
        baseRate = _baseRate;
        multiRate = _multiRate;
        emit AdjustRate(baseRate, multiRate);
    }

    function changeRateDecimal(uint256 newRateDecimal) external onlyOperator {
        rateDecimal = newRateDecimal;
    }

    function changeStakeMin(uint256 newStakeMin) external onlyOperator {
        stakeMin = newStakeMin;
        emit AdjustStakeMin(stakeMin);
    }

    function joinFido(address _inviter) external {
        require(!isMember[_msgSender()], "FidoMember: already joined");
        if (_inviter != address(0)) {
            require(isMember[_inviter], "FidoMember: inviter not joined");
            isInvited[_msgSender()] = true;
            inviter[_msgSender()] = _inviter;
            followers[_inviter].push(_msgSender());
        }
        isMember[_msgSender()] = true;
        joinBlockHeight[_msgSender()] = block.number;
        emit NewMember(_msgSender(), _inviter, block.number);
    }

    function changeMFILPool(address newMfilpool) external onlyOwner {
        mfilPool = newMfilpool;
    }

    function changeFidoOfficial(address newFidoOfficial) external onlyOwner {
        fidoOfficial = newFidoOfficial;
    }

    function changePool(address newPool) external onlyOwner {
```

Armors Labs

```
        pool = newPool;
    }

    function getFollowerCount(address member) external view returns (uint256) {
        return followers[member].length;
    }

    function caleInviteRate(address member)
        external
        view
        returns (
            address inviter0,
            address inviter1,
            address fido,
            uint256 rate0,
            uint256 rate1,
            uint256 fidoRate
        )
    {
        require(isMember[member], "FidoMember: not FIDO member.");
        uint256 mfilStake0;
        uint256 mfilStake1;
        uint256 mfilStake2 = IMFILPool(mfilPool).userStake(member);
        uint256 fullRate = 10**rateDecimal;
        if (!isInvited[member]) {
            return (fidoOfficial, fidoOfficial, fidoOfficial, 0, 0, fullRate);
        }
        inviter0 = inviter[member];
        uint256 stake0 = IFidoUsdtLPPool(pool).userTokenBStake(inviter0);
        inviter1 = isInvited[inviter0] ? inviter[inviter0] : fidoOfficial;
        if (inviter1 == fidoOfficial) {
            if (stake0 >= stakeMin) {
                if (mfilStake2 == 0) {
                    return (
                        inviter0,
                        fidoOfficial,
                        fidoOfficial,
                        baseRate + multiRate,
                        0,
                        fullRate.sub(baseRate + multiRate)
                    );
                }
                mfilStake0 = IMFILPool(mfilPool).userStake(inviter0);
                rate0 = baseRate + multiRate;
                if (mfilStake0 < mfilStake2) {
                    rate0 = rate0.mul(mfilStake0).div(mfilStake2);
                }
                return (
                    inviter0,
                    fidoOfficial,
                    fidoOfficial,
                    rate0,
                    0,
                    fullRate.sub(rate0)
                );
            } else {
                return (
                    fidoOfficial,
                    fidoOfficial,
                    fidoOfficial,
                    0,
                    0,
                    fullRate
                );
            }
        }
```

Armors Labs

```
        uint256 stake1 = IFidoUsdtLPPool(pool).userTokenBStake(inviter1);
        if (stake0 < stakeMin) {
            if (stake1 < stakeMin) {
                return (
                    fidoOfficial,
                    fidoOfficial,
                    fidoOfficial,
                    0,
                    0,
                    fullRate
                );
            } else {
                rate1 = fullRate.sub(baseRate);
                if (mfilStake2 == 0) {
                    return (
                        fidoOfficial,
                        inviter1,
                        fidoOfficial,
                        0,
                        rate1,
                        fullRate.sub(rate1)
                    );
                }
                mfilStake1 = IMFILPool(mfilPool).userStake(inviter1);
                if (mfilStake1 < mfilStake2) {
                    rate1 = rate1.mul(mfilStake1).div(mfilStake2);
                }
                return (
                    fidoOfficial,
                    inviter1,
                    fidoOfficial,
                    0,
                    rate1,
                    fullRate.sub(rate1)
                );
            }
        }
        if (stake1 < stakeMin) {
            inviter1 = fidoOfficial;
            if (stake0 < stakeMin) {
                return (
                    fidoOfficial,
                    fidoOfficial,
                    fidoOfficial,
                    0,
                    0,
                    fullRate
                );
            } else {
                if (mfilStake2 == 0) {
                    return (
                        inviter0,
                        fidoOfficial,
                        fidoOfficial,
                        baseRate + multiRate,
                        0,
                        fullRate.sub(baseRate + multiRate)
                    );
                }
                rate0 = baseRate + multiRate;
                mfilStake0 = IMFILPool(mfilPool).userStake(inviter0);
                if (mfilStake0 < mfilStake2) {
                    rate0 = rate0.mul(mfilStake0).div(mfilStake2);
                }
                return (
                    inviter0,
```

Armors Labs

```
                    fidoOfficial,
                    fidoOfficial,
                    rate0,
                    0,
                    fullRate.sub(rate0)
                );
            }
        }
        fido = fidoOfficial;
        rate0 = baseRate.add(
            stake0.mul(multiRate).mul(fullRate).div(stake0.add(stake1)).div(
                fullRate
            )
        );
        rate1 = fullRate.sub(rate0);
        if (mfilStake2 > 0) {
            mfilStake0 = IMFILPool(mfilPool).userStake(inviter0);
            mfilStake1 = IMFILPool(mfilPool).userStake(inviter1);
            if (mfilStake0 < mfilStake2) {
                rate0 = rate0.mul(mfilStake0).div(mfilStake2);
            }
            if (mfilStake1 < mfilStake2) {
                rate1 = rate1.mul(mfilStake1).div(mfilStake2);
            }
            fidoRate = fullRate.sub(rate0.add(rate1));
        }
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./StakeLPRewardPerBlock.sol";

contract FidoUsdtLPPool is StakeLPRewardPerBlock {

    address public usdt;
    address public fido;
    constructor(
        address _facotry,
        address _operator,
        address _usdt,
        address _fido
    ) StakeLPRewardPerBlock(_facotry, _operator, _usdt, _fido, _fido) {
        usdt = _usdt;
        fido = _fido;
    }

}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./ERC20/ERC20.sol";
import "./ERC20/ERC20Burnable.sol";
import "./ERC20/ERC20Mintable.sol";
import "./ERC20/ERC20Pausable.sol";
import "./Ownable.sol";

contract MFIL is ERC20Mintable, ERC20Burnable, Ownable {
    using SafeMath for uint256;

    address public operator;
    uint256 private _cap = 2000000000 * 10**18;

    event OperatorshipTransferred(
```

Armors Labs

```
        address indexed previousOperator,
        address indexed newOperator
    );

    constructor(address _operator)
        ERC20("Mirror FileCoin", "MFIL")
        Pausable()
        Ownable()
    {
        operator = _operator;
        emit OperatorshipTransferred(address(0), operator);
        _setupDecimals(18);
    }

    modifier onlyOperator() {
        require(
            _msgSender() == operator,
            "Operable: caller is not the operator"
        );
        _;
    }

    function transferOperatorship(address newOperator) external onlyOwner {
        require(
            newOperator != address(0),
            "Operable: new operator is the zero address"
        );
        emit OperatorshipTransferred(operator, newOperator);
        operator = newOperator;
    }

    function pause() external onlyOperator {
        _pause();
    }

    function unPause() external onlyOperator {
        _unpause();
    }

    function addMinter(address minter) external onlyOperator {
        _addMinter(minter);
    }

    function removeMinter(address minter) external onlyOperator {
        _removeMinter(minter);
    }

    /**
     * @dev Returns the cap on the token's total supply.
     */
    function cap() public view virtual returns (uint256) {
        return _cap;
    }

    /**
     * @dev See {ERC20-_beforeTokenTransfer}.
     *
     * Requirements:
     *
     * - minted tokens must not cause the total supply to go over the cap.
     */
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
```

Armors Labs

```
        super._beforeTokenTransfer(from, to, amount);

        if (from == address(0)) {
            // When minting tokens
            require(totalSupply().add(amount) <= cap(), "ERC20: cap exceeded");
        }
    }
}
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

import "./Context.sol";
/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract Ownable is Context {
    address private _owner;

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    function _initOwner(address owner_) internal {
        require(owner_ != address(0), "Ownable: owner cannot be init to zero address");
        _owner = owner_;
        emit OwnershipTransferred(address(0), _owner);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
```

Armors Labs

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./libraries/SafeMath.sol";
import "./Context.sol";

contract StakeRewardPerDay is Context {
    using SafeMath for uint256;

    uint256 public rewardRate; // Global Reward per Unit
    uint256 public constant blockCountPerDay = 20 * 60 * 24; // 3s - 1 block, 1min - 20 blocks
    uint256 public rewardToday;
    uint256 public rewardPreBlock; // rewardToday / blockCountPerDay
    uint256 public lastUpdateBlock; // last adjust block height
    uint256 public totalStake;
    uint256 public todayStartBlock;
    uint256 public todayEndBlock; // todayStartBlock + blockCountPerDay
    mapping(address => uint256) public userStake;
    mapping(address => uint256) public userReward;
    mapping(address => uint256) public userRate;

    constructor() {
        rewardRate = 0;
        rewardToday = 0;
        rewardPreBlock = 0;
        totalStake = 0;
        lastUpdateBlock = block.number;
        todayStartBlock = block.number;
        todayEndBlock = todayStartBlock + blockCountPerDay;
    }

    modifier updateRate() {
        if (
            rewardPreBlock > 0 &&
            totalStake > 0 &&
            block.number > lastUpdateBlock &&
            todayEndBlock > lastUpdateBlock
        ) {
            uint256 duringBlock;
            if (block.number < todayEndBlock) {
                duringBlock = block.number - lastUpdateBlock;
                lastUpdateBlock = block.number;
            } else {
                duringBlock = todayEndBlock - lastUpdateBlock;
                lastUpdateBlock = todayEndBlock;
            }
            uint256 deltaRate = rewardPreBlock.mul(duringBlock).div(totalStake);
            rewardRate = rewardRate.add(deltaRate);
        }
```

Armors Labs

```
        if(rewardPreBlock == 0){
            lastUpdateBlock = block.number;
        }
        _;
    }

    modifier updateEndRate() {
        if (
            rewardPreBlock > 0 &&
            totalStake > 0 &&
            todayEndBlock > lastUpdateBlock
        ) {
            uint256 duringBlock = todayEndBlock - lastUpdateBlock;
            uint256 deltaRate = rewardPreBlock.mul(duringBlock).div(totalStake);
            rewardRate = rewardRate.add(deltaRate);
        }
        if(rewardPreBlock == 0){
            lastUpdateBlock = block.number;
        }
        _;
    }

    modifier getReward() {
        if (userStake[_msgSender()] > 0 && rewardRate > userRate[_msgSender()]) {
            uint256 deltaReward =
                rewardRate.sub(userRate[_msgSender()]).mul(
                    userStake[_msgSender()]
                );
            userReward[_msgSender()] = userReward[_msgSender()].add(
                deltaReward
            );
        }
        userRate[_msgSender()] = rewardRate;
        _;
    }

    function _addNewReward(uint256 amount) internal updateEndRate{
        rewardToday = amount;
        rewardPreBlock = amount.mul(1e18).div(blockCountPerDay);
        todayStartBlock = block.number;
        todayEndBlock = todayStartBlock + blockCountPerDay;
    }

    function _stake(uint256 amount) updateRate getReward internal virtual {
        userStake[_msgSender()] = userStake[_msgSender()].add(amount);
        totalStake = totalStake.add(amount);
    }
    function _halveNoUpdate() internal returns (uint256 reward){
        reward = userReward[_msgSender()].div(1e18);
        userReward[_msgSender()] = 0;
    }

    function _halve() updateRate getReward internal returns (uint256 reward){
        return _halveNoUpdate();
    }

    function _unStake() updateRate getReward internal virtual returns (uint256 stake, uint256 reward)
        stake = userStake[_msgSender()];
        totalStake = totalStake.sub(stake);
        userStake[_msgSender()] = 0;
        reward = _halveNoUpdate();
    }

    function _getStake() view internal returns (uint256 stake) {
        stake = userStake[_msgSender()];
    }
```

Armors Labs

```
    function _caleReward() view internal returns (uint256 reward) {
        if(userStake[_msgSender()] == 0){
            return 0;
        }
        uint256 tempRate = rewardRate;
        if (
            rewardPreBlock > 0 &&
            totalStake > 0 &&
            block.number > lastUpdateBlock &&
            todayEndBlock > lastUpdateBlock
        ) {
            uint256 duringBlock;
            if (block.number < todayEndBlock) {
                duringBlock = block.number - lastUpdateBlock;
            } else {
                duringBlock = todayEndBlock - lastUpdateBlock;
            }
            uint256 deltaRate = rewardPreBlock.mul(duringBlock).div(totalStake);
            tempRate = tempRate.add(deltaRate);
        }
        reward = userReward[_msgSender()];
        if (tempRate > userRate[_msgSender()]) {
            uint256 deltaReward =
                tempRate.sub(userRate[_msgSender()]).mul(
                    userStake[_msgSender()]
                );
            reward = reward.add(
                deltaReward
            );
        }
        reward = reward.div(1e18);
    }

}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./StakeLPRewardPerDay.sol";
import "./interfaces/IFidoMember.sol";
import "./interfaces/IMFIL.sol";

interface IMFILPool {
    function lendFil(uint256 totalT, uint256 userStakeFil)
        external
        returns (uint256 lendAmount);

    function returnFil(uint256 amount) external;
}

interface IMfilIdoTokenLPPoolFactory {
    function operator() external view returns (address);

    function MfilStakePool() external view returns (address);

    function FidoMember() external view returns (address);

    function inviteFeeRate() external view returns (uint256);

    function inviteFeeRateDecimals() external view returns (uint256);
}

contract MfilIdoTokenLPPool is StakeLPRewardPerDay {
    using SafeMath for uint256;
```

Armors Labs

```
    address public mfil;
    address public idoToken;
    address public lpPoolFactory;

    uint256 public totalMfilLend;
    mapping(address => uint256) public userMfilLend;

    constructor(
        address _factory,
        address fidoOwner,
        address _mfil,
        address _idoToken
    ) StakeLPRewardPerDay(_factory, fidoOwner, _mfil, _idoToken, _mfil) {
        lpPoolFactory = _msgSender();
        mfil = _mfil;
        idoToken = _idoToken;
    }

    modifier onlyMfilStakePool() {
        require(
            _msgSender() ==
                IMfilIdoTokenLPPoolFactory(lpPoolFactory).MfilStakePool(),
            "MFIL-IDOTOKEN-LP-POOL: caller is not the MfilStakePool"
        );
        _;
    }

    function MfilStakePool() external view returns (address) {
        return IMfilIdoTokenLPPoolFactory(lpPoolFactory).MfilStakePool();
    }

    function FidoMember() external view returns (address) {
        return IMfilIdoTokenLPPoolFactory(lpPoolFactory).FidoMember();
    }

    function inviteFeeRateDecimals() external view returns (uint256) {
        return
            IMfilIdoTokenLPPoolFactory(lpPoolFactory).inviteFeeRateDecimals();
    }

    function inviteFeeRate() external view returns (uint256) {
        return IMfilIdoTokenLPPoolFactory(lpPoolFactory).inviteFeeRate();
    }

    function _lendFil(uint256 totalT, uint256 userStakeFil)
        internal
        returns (uint256 lendAmount)
    {
        lendAmount = IMFILPool(this.MfilStakePool()).lendFil(
            totalT,
            userStakeFil
        );
        userMfilLend[_msgSender()] = userMfilLend[_msgSender()].add(lendAmount);
        totalMfilLend = totalMfilLend.add(lendAmount);
    }

    function _returnFIL() internal returns (uint256 amount) {
        amount = userMfilLend[_msgSender()];
        IMFILPool(this.MfilStakePool()).returnFil(amount);
        userMfilLend[_msgSender()] = 0;
        totalMfilLend = totalMfilLend.sub(amount);
    }

    function _stakeHook(
        uint256 amountAIn,
        uint256 amountBIn,
```

Armors Labs

```
        uint256 amountA,
        uint256 amountB,
        uint256 liquidity
    ) internal override {
        amountB = amountB.mul(stakeRate).div(stakeRate - 1);
        _lendFil(amountB, amountA);
    }

    function _unStakeHook(
        uint256 reward,
        uint256 amountA,
        uint256 amountB,
        uint256 liquidity
    ) internal override {
        _returnFIL();
        uint256 inviteFee =
            reward.mul(this.inviteFeeRate()).div(
                10**uint256(this.inviteFeeRateDecimals())
            );
        {
            address inviter0;
            address inviter1;
            address fido;
            uint256 rate0;
            uint256 rate1;
            uint256 fidoRate;
            uint256 fee;
            (inviter0, inviter1, fido, rate0, rate1, fidoRate) = IFidoMember(
                this.FidoMember()
            )
                .caleInviteRate(_msgSender());
            fee = inviteFee.mul(rate0).div(
                10**IFidoMember(this.FidoMember()).rateDecimal()
            );
            IMFIL(mfil).mint(inviter0, fee);
            fee = inviteFee.mul(rate1).div(
                10**IFidoMember(this.FidoMember()).rateDecimal()
            );
            IMFIL(mfil).mint(inviter1, fee);
            fee = inviteFee.mul(fidoRate).div(
                10**IFidoMember(this.FidoMember()).rateDecimal()
            );
            IMFIL(mfil).mint(fido, fee);
        }
    }

    function newReward(uint256 amount)
        external
        virtual
        override
        onlyMfilStakePool
        whenNotPaused
    {
        _addNewReward(amount);
        emit NewReward(block.number, amount);
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./StakeRewardPerBlock.sol";
import "./MdexRouter.sol";
import "./Ownable.sol";
import "./Pausable.sol";
import "./ReentrancyGuard.sol";
```

Armors Labs

```
import "./interfaces/IERC20Mintable.sol";

contract StakeLPRewardPerBlock is
    StakeRewardPerBlock,
    MdexRouter,
    Ownable,
    Pausable,
    ReentrancyGuard
{
    using SafeMath for uint256;

    uint8 public stakeRate = 5;

    address public tokenA; // for one
    address public tokenB; // for stakeRate
    address public rewardToken; // for reward
    address public operator;

    mapping(address => uint256) public userTokenBStake;
    uint256 public totalTokenBStake;

    event Stake(
        address indexed sender,
        uint256 amountA,
        uint256 amountB,
        uint256 liquidity
    );
    event UnStake(
        address indexed receiver,
        uint256 amountA,
        uint256 amountB,
        uint256 liquidity
    );
    event Reward(address indexed receiver, uint256 reward);
    event OperatorshipTransferred(
        address indexed previousOperator,
        address indexed newOperator
    );
    event NewReward(uint256 blockHeight, uint256 amount);

    constructor(
        address _factory,
        address _operator,
        address _tokenA,
        address _tokenB,
        address _rewardToken
    )
        StakeRewardPerBlock()
        MdexRouter(_factory)
        Ownable()
        Pausable()
        ReentrancyGuard()
    {
        operator = _operator;
        emit OperatorshipTransferred(address(0), operator);
        tokenA = _tokenA;
        tokenB = _tokenB;
        rewardToken = _rewardToken;
    }

    modifier onlyOperator() {
        require(
            _msgSender() == operator,
            "Operable: caller is not the operator"
        );
        _;
```

Armors Labs

```
    }

    function transferOperatorship(address newOperator) external onlyOwner {
        require(
            newOperator != address(0),
            "Operable: new operator is the zero address"
        );
        emit OperatorshipTransferred(operator, newOperator);
        operator = newOperator;
    }

    function pause() external onlyOperator {
        _pause();
    }

    function unPause() external onlyOperator {
        _unpause();
    }

    function caleLiquidity(
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin
    ) external view returns (uint256 amountA, uint256 amountB) {
        require(
            IMdexFactory(factory).getPair(tokenA, tokenB) != address(0),
            "MdexRouter: Pair Not Found!"
        );
        (uint256 reserveA, uint256 reserveB) =
            IMdexFactory(factory).getReserves(tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint256 amountBOptimal =
                IMdexFactory(factory).quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(
                    amountBOptimal >= amountBMin,
                    "MdexRouter: INSUFFICIENT_B_AMOUNT"
                );
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint256 amountAOptimal =
                    IMdexFactory(factory).quote(
                        amountBDesired,
                        reserveB,
                        reserveA
                    );
                assert(amountAOptimal <= amountADesired);
                require(
                    amountAOptimal >= amountAMin,
                    "MdexRouter: INSUFFICIENT_A_AMOUNT"
                );
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }

    function stake(uint256 amountAIn, uint256 amountBIn)
        external
        whenNotPaused
        nonReentrant
        returns (
            uint256 amountA,
            uint256 amountB,
```

Armors Labs

```
            uint256 liquidity
        )
    {
        require(amountAIn > 0 && amountBIn > 0, "STAKE: amount is zero");
        uint256 tempAmountB = amountBIn.div(stakeRate);
        (amountA, amountB, liquidity) = addLiquidity(
            tokenA,
            tokenB,
            amountAIn,
            tempAmountB,
            0,
            0
        );
        amountB = amountB.mul(stakeRate - 1);
        TransferHelper.safeTransferFrom(
            tokenB,
            _msgSender(),
            address(this),
            amountB
        );
        userTokenBStake[_msgSender()] = userTokenBStake[_msgSender()].add(
            amountB
        );
        totalTokenBStake = totalTokenBStake.add(amountB);
        _stake(liquidity);
        emit Stake(_msgSender(), amountA, amountB, liquidity);
    }

    function getStake() external view whenNotPaused returns (uint256 amount) {
        amount = _getStake();
    }

    function halve()
        external
        whenNotPaused
        nonReentrant
        returns (uint256 reward)
    {
        reward = _Bhalve();
        IERC20Mintable(rewardToken).mint(_msgSender(), reward);
        emit Reward(_msgSender(), reward);
    }

    function unStake()
        external
        whenNotPaused
        nonReentrant
        returns (
            uint256 reward,
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        )
    {
        (liquidity, reward) = _unStake();
        require(liquidity > 0, "UNSTAKE: no stake");
        IERC20Mintable(rewardToken).mint(_msgSender(), reward);
        emit Reward(_msgSender(), reward);
        (amountA, amountB) = removeLiquidity(
            tokenA,
            tokenB,
            liquidity,
            0,
            0,
            _msgSender()
        );
```

Armors Labs

```
        IERC20(tokenB).transfer(_msgSender(), userTokenBStake[_msgSender()]);
        amountB.add(userTokenBStake[_msgSender()]);
        userTokenBStake[_msgSender()] = 0;
        emit UnStake(_msgSender(), amountA, amountB, liquidity);
    }

    function adjustReward(uint256 amount) external onlyOperator whenNotPaused {
        _setReward(amount);
        emit NewReward(block.number, amount);
    }

    function caleReward() external view whenNotPaused returns (uint256 reward) {
        reward = _BcaleReward();
    }
}
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
}
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
```

Armors Labs

```
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, 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 (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
```

Armors Labs

```
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
```

Armors Labs

```
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;
// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return t
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
    }
```

Armors Labs

```
    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 safeTransferETH(address to, uint value) internal {
        (bool success,) = to.call{value : value}(new bytes(0));
        require(success, 'TransferHelper: ETH_TRANSFER_FAILED');
    }
}// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

import "./Context.sol";

/**
 * @dev Contract module which allows children to implement an emergency stop
 * mechanism that can be triggered by an authorized account.
 *
 * This module is used through inheritance. It will make available the
 * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
 * the functions of your contract. Note that they will not be pausable by
 * simply including this module, only once the modifiers are put in place.
 */
abstract contract Pausable is Context {
    /**
     * @dev Emitted when the pause is triggered by `account`.
     */
    event Paused(address account);

    /**
     * @dev Emitted when the pause is lifted by `account`.
     */
    event Unpaused(address account);

    bool private _paused;

    /**
     * @dev Initializes the contract in unpaused state.
     */
    constructor () {
        _paused = false;
    }

    /**
     * @dev Returns true if the contract is paused, and false otherwise.
     */
    function paused() public view virtual returns (bool) {
        return _paused;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    modifier whenNotPaused() {
```

Armors Labs

```
        require(!paused(), "Pausable: paused");
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    modifier whenPaused() {
        require(paused(), "Pausable: not paused");
        _;
    }

    /**
     * @dev Triggers stopped state.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    function _pause() internal virtual whenNotPaused {
        _paused = true;
        emit Paused(_msgSender());
    }

    /**
     * @dev Returns to normal state.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    function _unpause() internal virtual whenPaused {
        _paused = false;
        emit Unpaused(_msgSender());
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./Ownable.sol";
import "./interfaces/IERC20.sol";
import "./libraries/SafeMath.sol";

contract FidoMargin is Ownable {
    using SafeMath for uint256;

    address public operator;
    address public marginToken;
    mapping (address => uint256) public margin;
    mapping (address => bool) public withdrawStatus;
    mapping (address => uint256) public withdrawAmount;
    mapping (address => address) public withdrawRecipient;

    event DepositMargin(uint256 timestamp, address indexed sender, uint256 amount);
    event WithdrawRequest(uint256 timestamp, address indexed requester, uint256 amount, address index
    event WithdrawResult(uint256 timestamp, address indexed requester, bool result);

    event OperatorshipTransferred(
        address indexed previousOperator,
        address indexed newOperator
    );
```

Armors Labs

```
    constructor (address _operator, address _marginToken) Ownable() {
        operator = _operator;
        emit OperatorshipTransferred(address(0), operator);
        marginToken = _marginToken;
    }

    modifier onlyOperator() {
        require(
            _msgSender() == operator,
            "Operable: caller is not the operator"
        );
        _;
    }

    function transferOperatorship(address newOperator) external onlyOwner {
        require(
            newOperator != address(0),
            "Operable: new operator is the zero address"
        );
        emit OperatorshipTransferred(operator, newOperator);
        operator = newOperator;
    }

    function depositMargin(uint256 amount) external {
        IERC20(marginToken).transferFrom(_msgSender(), address(this), amount);
        margin[_msgSender()] = margin[_msgSender()].add(amount);
        emit DepositMargin(block.timestamp, _msgSender(), amount);
    }

    function withdrawRequest(address recipient, uint256 amount) external {
        require(!withdrawStatus[_msgSender()], "FidoMargin: already sent a request");
        require(margin[_msgSender()] >= amount, "FidoMargin: insufficient margin");

        withdrawStatus[_msgSender()] = true;
        withdrawAmount[_msgSender()] = amount;
        withdrawRecipient[_msgSender()] = recipient;
        emit WithdrawRequest(block.timestamp, _msgSender(), amount, recipient);
    }

    function withdrawResponse(address requester, bool result) external onlyOperator {
        require(withdrawStatus[requester], "FidoMargin: request not found");
        if(result){
            IERC20(marginToken).transfer(withdrawRecipient[requester], withdrawAmount[requester]);
            margin[_msgSender()] = margin[_msgSender()].sub(withdrawAmount[requester]);
        }
        withdrawStatus[requester] = false;
        withdrawRecipient[requester] = address(0);
        withdrawAmount[requester] = 0;
        emit WithdrawResult(block.timestamp, requester, result);
    }
}// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "../Ownable.sol";
import "../Pausable.sol";

contract RateOracle is Ownable, Pausable {
    address public updater;
    address public requester;

    event UpdatershipTransferred(
        address indexed previousUpdater,
        address indexed newUpdater
    );
```

Armors Labs

```
    event RequestershipTransferred(
        address indexed previousRequester,
        address indexed newRequester
    );

    modifier onlyUpdater() {
        require(_msgSender() == updater, "Oracle: caller is not the updater");
        _;
    }

    modifier onlyRequester() {
        require(
            _msgSender() == requester,
            "Oracle: caller is not the requester"
        );
        _;
    }

    bool private _status = false; // request but not updated -> false; request and updated -> true
    uint256 private _lastUpdateTime = 0; // last updated timestamp
    uint256 private _lastRequestTime = 0; // last request timestamp

    uint8 private _decimals = 18; // decimal
    uint256 private _rate; // rate

    event Request(uint256 indexed timestamp);
    event Update(uint256 indexed timestamp, uint256 rate);

    constructor(address _updater, address _requester) Ownable() Pausable() {
        updater = _updater;
        emit UpdatershipTransferred(address(0), updater);
        requester = _requester;
        emit RequestershipTransferred(address(0), requester);

        _lastRequestTime = block.timestamp;
        emit Request(block.timestamp);
    }

    function transferRequestership(address newRequester) external onlyOwner {
        require(
            newRequester != address(0),
            "Oracle: new Requester is the zero address"
        );
        emit RequestershipTransferred(requester, newRequester);
        requester = newRequester;
    }

    function transferUpdatership(address newUpdater) external onlyOwner {
        require(
            newUpdater != address(0),
            "Oracle: new Updater is the zero address"
        );
        emit UpdatershipTransferred(updater, newUpdater);
        updater = newUpdater;
    }

    function pause() external onlyOwner {
        _pause();
    }

    function unPause() external onlyOwner {
        _unpause();
    }

    function status() external view returns (bool) {
        return _status;
```

Armors Labs

Armors Labs

```
    }

    function lastRequestTime() external view returns (uint256) {
        return _lastRequestTime;
    }

    function lastUpdateTime() external view returns (uint256) {
        return _lastUpdateTime;
    }

    function decimals() external view whenNotPaused returns (uint8) {
        return _decimals;
    }

    function rate() external view whenNotPaused returns (uint256) {
        return _rate;
    }

    function request() external onlyRequester whenNotPaused {
        require(_status, "Oracle: request already called");
        _status = false;
        _lastRequestTime = block.timestamp;
        emit Request(block.timestamp);
    }

    function update(uint256 rate_) external onlyUpdater whenNotPaused {
        require(!_status, "Oracle: no request called");
        require(rate_ > 0, "Oracle: rate can not be zero");
        _status = true;
        _lastUpdateTime = block.timestamp;
        _rate = rate_;
        emit Update(block.timestamp, _rate);
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./MFIL-IDOToken.sol";
import "./interfaces/IIDOInfo.sol";

contract MfilIdoTokenLPPoolFactory is Ownable {
    address public mfil;
    address public factory;
    address public MfilStakePool;
    address public operator;
    address public IDOInfo;
    address public FidoMember;

    uint256 public inviteFeeRate = 50; // inviteFee = inviteFeeRate / 10**inviteFeeRateDecimals
    uint8 public inviteFeeRateDecimals = 3;

    mapping (address => address) public ido2pool;

    event OperatorshipTransferred(
        address indexed previousOperator,
        address indexed newOperator
    );

    event NewPool(address indexed idoToken, address indexed pool);

    constructor(
        address _operator,
        address _mfil,
        address _factory,
        address _IDOInfo,
```

Armors Labs

```
        address _FidoMember,
        address _MfilStakePool
    ) Ownable() {
        operator = _operator;
        emit OperatorshipTransferred(address(0), operator);
        mfil = _mfil;
        factory = _factory;
        IDOInfo = _IDOInfo;
        FidoMember = _FidoMember;
        MfilStakePool = _MfilStakePool;

    }

    modifier onlyOperator() {
        require(
            _msgSender() == operator,
            "Operable: caller is not the operator"
        );
        _;
    }

    function transferOperatorship(address newOperator) external onlyOwner {
        require(
            newOperator != address(0),
            "Operable: new operator is the zero address"
        );
        emit OperatorshipTransferred(operator, newOperator);
        operator = newOperator;
    }

    function newPool(address idoToken) onlyOperator external returns (address pool) {
        require(IIDOInfo(IDOInfo).exist(idoToken), "PoolFactory: idoToken not found");
        require(ido2pool[idoToken] == address(0), "PoolFactory: pool exist");
        MfilIdoTokenLPPool newpool = new MfilIdoTokenLPPool(factory, operator, mfil, idoToken);
        pool = address(newpool);
        ido2pool[idoToken] = pool;
        emit NewPool(idoToken, pool);
    }

    function changePoolOperator(address idoToken, address newOperator) onlyOperator external {
        require(ido2pool[idoToken] != address(0), "PoolFactory: pool not found");
        MfilIdoTokenLPPool(ido2pool[idoToken]).transferOperatorship(newOperator);
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./StakeTokenPool.sol";
import "./interfaces/IFidoMember.sol";
import "./interfaces/IIDOInfo.sol";
import "./interfaces/IRateOracle.sol";
import "./interfaces/IMFIL.sol";
import "./interfaces/IHFIL.sol";
import "./interfaces/IERC20Mintable.sol";
import "./libraries/TransferHelper.sol";
import "./ReentrancyGuard.sol";

interface IMfilIdoTokenLPPool {
    function newReward(uint256 amount) external;

    function decimals() external view returns (uint8);
}

contract MFILPool is StakeTokenPool, ReentrancyGuard {
    using SafeMath for uint256;
```

Armors Labs

```
    address public hfil;
    address public mfil;
    address public fido;

    address public FidoMember;
    address public operator;
    address public fidoFeeRecipient;
    address public rateOracle;
    address public IDOInfo;
    address public HFILRecipient;

    uint256 public totalMfilLend;
    mapping(address => uint256) public MfilLend;
    mapping(address => uint256) public poolLastPaid;
    mapping(address => uint256) public poolFeeRate;

    uint256 public totalPaid;
    mapping(address => uint256) public poolTotalPaid;

    uint256 public tomorrowReward;

    uint256 public lendRewardRate = 12; // lendReward = lendRewardRate / 10**lendRewardRateDecimals
    uint8 public lendRewardRateDecimals = 5;

    uint256 public fidoFeeRate = 1; // fidoFee = fidoFeeRate / 10**fidoFeeRateDecimals
    uint8 public fidoFeeRateDecimals = 2;

    uint256 public inviteFeeRate = 10; // inviteFee = inviteFeeRate / 10**inviteFeeRateDecimals
    uint8 public inviteFeeRateDecimals = 2;

    uint256 public poolDefaultFeeRate = 100; // poolFee = poolFeeRate / 10**poolFeeDecimals
    uint256 public poolFeeDecimals = 3;

    uint256 public poolFidoFeeRate = 100; // poolFidoFee = poolFidoFeeRate / 10**poolFidoFeeDecimals
    uint256 public poolFidoFeeDecimals = 3;

    event OperatorshipTransferred(
        address indexed previousOperator,
        address indexed newOperator
    );

    event Stake(address indexed sender, uint256 amount);

    event UnStake(
        address indexed receiver,
        uint256 amount,
        uint256 mfilReward,
        uint256 fidoReward
    );

    event Reward(
        address indexed receiver,
        uint256 mfilReward,
        uint256 fidoReward
    );

    event PayDaily(
        address indexed pool,
        address indexed payer,
        uint256 deposit,
        uint256 reward,
        uint256 fidoFee,
        uint256 lendReward
    );
```

Armors Labs

```
    event NewReward(uint256 blockHeight, uint256 amount);
    event SetReward(uint256 blockHeight, uint256 amount);

    constructor(
        address _operator,
        address _hfil,
        address _mfil,
        address _fido,
        address _IDOInfo,
        address _FidoMember,
        address _fidoFeeRecipient,
        address _HFILRecipient,
        address _rateOracle
    ) StakeTokenPool() ReentrancyGuard() {
        operator = _operator;
        emit OperatorshipTransferred(address(0), operator);
        hfil = _hfil;
        mfil = _mfil;
        fido = _fido;
        FidoMember = _FidoMember;
        fidoFeeRecipient = _fidoFeeRecipient;
        HFILRecipient = _HFILRecipient;
        IDOInfo = _IDOInfo;
        rateOracle = _rateOracle;
    }

    modifier onlyOperator() {
        require(
            _msgSender() == operator,
            "Operable: caller is not the operator"
        );
        _;
    }

    function transferOperatorship(address newOperator) external onlyOwner {
        require(
            newOperator != address(0),
            "Operable: new operator is the zero address"
        );
        emit OperatorshipTransferred(operator, newOperator);
        operator = newOperator;
    }

    function changeRateOracle(address _rateOracle)
        external
        onlyOwner
        returns (bool)
    {
        require(_rateOracle != address(0), "new address is zero address");
        rateOracle = _rateOracle;
        return true;
    }

    function changeIDOInfo(address _IDOInfo) external onlyOwner returns (bool) {
        require(_IDOInfo != address(0), "new address is zero address");
        IDOInfo = _IDOInfo;
        return true;
    }

    function changeFidoMember(address _FidoMember)
        external
        onlyOwner
        returns (bool)
    {
        require(_FidoMember != address(0), "new address is zero address");
        FidoMember = _FidoMember;
```

Armors Labs

```
        return true;
    }

    function changeFidoFeeRecipient(address _fidoFeeRecipient)
        external
        onlyOwner
        returns (bool)
    {
        require(_fidoFeeRecipient != address(0), "new address is zero address");
        fidoFeeRecipient = _fidoFeeRecipient;
        return true;
    }

    function changeHFILRecipient(address _HFILRecipient)
        external
        onlyOwner
        returns (bool)
    {
        require(_HFILRecipient != address(0), "new address is zero address");
        HFILRecipient = _HFILRecipient;
        return true;
    }

    function pause() external onlyOperator {
        _pause();
    }

    function unPause() external onlyOperator {
        _unpause();
    }

    function changeFidoFeeRate(uint256 _fidoFeeRate, uint8 _fidoFeeRateDecimals)
        external
        onlyOperator
    {
        require(
            _fidoFeeRateDecimals > 0,
            "fidoFeeRateDecimals must greater than zero"
        );
        fidoFeeRateDecimals = _fidoFeeRateDecimals;
        fidoFeeRate = _fidoFeeRate;
    }

    function changeInviteFeeRate(
        uint256 _inviteFeeRate,
        uint8 _inviteFeeRateDecimals
    ) external onlyOperator {
        require(
            _inviteFeeRateDecimals > 0,
            "inviteFeeRateDecimals must greater than zero"
        );
        inviteFeeRateDecimals = _inviteFeeRateDecimals;
        inviteFeeRate = _inviteFeeRate;
    }

    function changeLendRewardRate(
        uint256 _lendRewardRate,
        uint8 _lendRewardRateDecimals
    ) external onlyOperator {
        require(
            _lendRewardRateDecimals > 0,
            "lendRewardRateDecimals must greater than zero"
        );
        lendRewardRateDecimals = _lendRewardRateDecimals;
        lendRewardRate = _lendRewardRate;
    }
```

Armors Labs

```
function changePoolDefaultFeeRate(
    uint256 _poolDefaultFeeRate,
    uint8 _poolFidoFeeDecimals
) external onlyOperator {
    require(
        _poolFidoFeeDecimals > 0,
        "poolFidoFeeDecimals must greater than zero"
    );
    poolFidoFeeDecimals = _poolFidoFeeDecimals;
    poolDefaultFeeRate = _poolDefaultFeeRate;
}

function changePoolFeeRate(address pool, uint256 _poolFeeRate)
    external
    onlyOperator
{
    poolFeeRate[pool] = _poolFeeRate;
}

function changePoolFidoFeeRate(
    uint256 _poolFidoFeeRate,
    uint8 _poolFidoFeeDecimals
) external onlyOperator {
    require(
        _poolFidoFeeDecimals > 0,
        "poolFidoFeeDecimals must greater than zero"
    );
    poolFidoFeeDecimals = _poolFidoFeeDecimals;
    poolFidoFeeRate = _poolFidoFeeRate;
}

function caleReward()
    external
    view
    whenNotPaused
    returns (uint256 mfilReward, uint256 fidoReward)
{
    mfilReward = _caleReward();
    fidoReward = _BcaleReward();
}

function halve()
    external
    whenNotPaused
    nonReentrant
    returns (uint256 mfilReward, uint256 fidoReward)
{
    mfilReward = _halve();
    fidoReward = _Bhalve();

    if (fidoReward > 0) {
        uint256 inviteFee =
            fidoReward.mul(inviteFeeRate).div(
                10**uint256(inviteFeeRateDecimals)
            );
        fidoReward = fidoReward.sub(inviteFee);
        {
            address inviter0;
            address inviter1;
            address fidoer;
            uint256 rate0;
            uint256 rate1;
            uint256 fidoRate;
            uint256 fee;
            (
```

Armors Labs

```
                inviter0,
                inviter1,
                fidoer,
                rate0,
                rate1,
                fidoRate
            ) = IFidoMember(FidoMember).caleInviteRate(_msgSender());
            fee = inviteFee.mul(rate0).div(
                10**IFidoMember(FidoMember).rateDecimal()
            );
            IERC20Mintable(fido).mint(inviter0, fee);
            fee = inviteFee.mul(rate1).div(
                10**IFidoMember(FidoMember).rateDecimal()
            );
            IERC20Mintable(fido).mint(inviter1, fee);
            fee = inviteFee.mul(fidoRate).div(
                10**IFidoMember(FidoMember).rateDecimal()
            );
            IERC20Mintable(fido).mint(fidoer, fee);
        }
        IERC20Mintable(fido).mint(fidoFeeRecipient, fidoReward);
    }
    if (mfilReward > 0) {
        IMFIL(mfil).transfer(_msgSender(), mfilReward);
    }
    emit Reward(_msgSender(), mfilReward, fidoReward);
}

function stake(uint256 amount) external whenNotPaused nonReentrant {
    require(amount > 0, "STAKE: amount is zero");
    TransferHelper.safeTransferFrom(
        mfil,
        _msgSender(),
        address(this),
        amount
    );
    _stake(amount);
}

function unStake()
    external
    whenNotPaused
    returns (
        uint256 amount,
        uint256 mfilReward,
        uint256 fidoReward
    )
{
    (amount, mfilReward, fidoReward) = _unStakeAll();
    require(amount > 0, "UNSTAKE: amount is zero");
    IMFIL(mfil).transfer(_msgSender(), amount);
    if (fidoReward > 0) {
        uint256 inviteFee =
            fidoReward.mul(inviteFeeRate).div(
                10**uint256(inviteFeeRateDecimals)
            );
        fidoReward = fidoReward.sub(inviteFee);
        {
            address inviter0;
            address inviter1;
            address fidoer;
            uint256 rate0;
            uint256 rate1;
            uint256 fidoRate;
            uint256 fee;
            (
```

Armors Labs

```
                inviter0,
                inviter1,
                fidoer,
                rate0,
                rate1,
                fidoRate
            ) = IFidoMember(FidoMember).caleInviteRate(_msgSender());
            fee = inviteFee.mul(rate0).div(
                10**IFidoMember(FidoMember).rateDecimal()
            );
            IERC20Mintable(fido).mint(inviter0, fee);
            fee = inviteFee.mul(rate1).div(
                10**IFidoMember(FidoMember).rateDecimal()
            );
            IERC20Mintable(fido).mint(inviter1, fee);
            fee = inviteFee.mul(fidoRate).div(
                10**IFidoMember(FidoMember).rateDecimal()
            );
            IERC20Mintable(fido).mint(fidoer, fee);
        }
        IERC20Mintable(fido).mint(fidoFeeRecipient, fidoReward);
    }
    if (mfilReward > 0) {
        IMFIL(mfil).transfer(_msgSender(), mfilReward);
    }
    emit Reward(_msgSender(), mfilReward, fidoReward);
    emit UnStake(_msgSender(), amount, mfilReward, fidoReward);
}

function lendFil(uint256 totalT, uint256 userStakeFil)
    external
    whenNotPaused
    returns (uint256 lendAmount)
{
    require(
        IIDOInfo(IDOInfo).isPool(_msgSender()),
        "LENDFIL: not from pool"
    );
    // require(IRateOracle(rateOracle).rate() > 0,"LENDFIL: oracle rate is zero");
    lendAmount = totalT.mul(IRateOracle(rateOracle).rate()).div(
        10**6
        // 10**IMfilIdoTokenLPPool(_msgSender()).decimals()
    );
    // oracle & mfil decimals both 18
    if (lendAmount > userStakeFil) {
        lendAmount = lendAmount.sub(userStakeFil);
    } else {
        lendAmount = 0;
    }
    MfilLend[_msgSender()] = MfilLend[_msgSender()].add(lendAmount);
    totalMfilLend = totalMfilLend.add(lendAmount);
}

function returnFil(uint256 amount) external whenNotPaused {
    require(
        IIDOInfo(IDOInfo).isPool(_msgSender()),
        "RETURNFIL: not from pool"
    );
    totalMfilLend = totalMfilLend.sub(amount);
    MfilLend[_msgSender()] = MfilLend[_msgSender()].add(amount);
}

function _getPoolFeeRate(address pool) internal view returns (uint256 fee) {
    fee = poolFeeRate[pool] == 0 ? poolDefaultFeeRate : poolFeeRate[pool];
}
```

Armors Labs

```
    function getPoolFeeRate(address pool) external view returns (uint256) {
        return _getPoolFeeRate(pool);
    }

    function payDaily(uint256 deposit) external whenNotPaused {
        address pool = IIDOInfo(IDOInfo).payer2pool(_msgSender());
        require(IIDOInfo(IDOInfo).isPool(pool), "MFILPOOL: no pool to pay");
        uint256 newDeposit = 0;
        newDeposit = deposit.sub(
            deposit.mul(_getPoolFeeRate(pool)).div(10**poolFeeDecimals)
        );
        TransferHelper.safeTransferFrom(
            hfil,
            _msgSender(),
            HFILRecipient,
            newDeposit
        );
        newDeposit = newDeposit.sub(
            deposit.mul(poolFidoFeeRate).div(10**poolFidoFeeDecimals)
        );
        totalPaid = totalPaid.add(newDeposit);
        poolTotalPaid[pool] = poolTotalPaid[pool].add(newDeposit);
        poolLastPaid[pool] = block.timestamp;
        uint256 lendRate = totalStake.mul(100).div(totalMfilLend);
        if (lendRate < 10) {
            // If lend / totalStake < 10%, give all to miners
            IMFIL(mfil).mint(pool, newDeposit);
            IMfilIdoTokenLPPool(pool).newReward(newDeposit);
            emit PayDaily(pool, _msgSender(), deposit, newDeposit, 0, 0);
        } else {
            uint256 tempLend = totalMfilLend.div(10**12); // 0.0001 * 100
            tempLend = tempLend.mul(tempLend).mul(tempLend);
            uint256 temptotalStake = totalStake.div(10**14); // 0.0001
            temptotalStake = temptotalStake.mul(temptotalStake).mul(
                temptotalStake
            );
            lendRate = lendRewardRate.mul(tempLend).div(temptotalStake).div(
                10**uint256(lendRewardRateDecimals)
            );
            uint256 fidoFee;
            if (lendRate >= 100) {
                // give all to stake
                fidoFee = newDeposit.mul(fidoFeeRate).div(
                    10**uint256(fidoFeeRateDecimals)
                );
                newDeposit = newDeposit.sub(fidoFee);
                tomorrowReward = tomorrowReward.add(newDeposit);
                IMFIL(mfil).mint(address(this), newDeposit);
                IMFIL(mfil).mint(fidoFeeRecipient, fidoFee);
                IMfilIdoTokenLPPool(pool).newReward(0);
                emit PayDaily(
                    pool,
                    _msgSender(),
                    deposit,
                    0,
                    fidoFee,
                    newDeposit
                );
            } else {
                uint256 lendReward =
                    newDeposit
                        .mul(lendRewardRate)
                        .mul(tempLend)
                        .div(temptotalStake)
                        .div(10**(uint256(lendRewardRateDecimals) + 2));
                fidoFee = lendReward.mul(fidoFeeRate).div(
```

Armors Labs

```
                    10**uint256(fidoFeeRateDecimals)
                );
                newDeposit = newDeposit.sub(lendReward);
                lendReward = lendReward.sub(fidoFee);
                tomorrowReward = tomorrowReward.add(lendReward);
                IMFIL(mfil).mint(pool, newDeposit);
                IMfilIdoTokenLPPool(pool).newReward(newDeposit);
                IMFIL(mfil).mint(address(this), lendReward);
                IMFIL(mfil).mint(fidoFeeRecipient, fidoFee);
                emit PayDaily(
                    pool,
                    _msgSender(),
                    deposit,
                    newDeposit,
                    fidoFee,
                    lendReward
                );
            }
        }
    }

    function newReward() external onlyOperator whenNotPaused {
        _addNewReward(tomorrowReward);
        emit NewReward(block.number, tomorrowReward);
        tomorrowReward = 0;
    }

    function setReward(uint256 amount) external onlyOperator whenNotPaused {
        _setReward(amount);
        emit SetReward(block.number, amount);
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./Ownable.sol";
import "./IDOToken.sol";
import "./interfaces/IMdexPair.sol";
import "./interfaces/IMdexFactory.sol";


contract IDOFactory is Ownable {
    using SafeMath for uint256;

    address[] public IDOlist;
    mapping(address => bool) public isIDO;
    mapping(address => address[]) public userIDOs;
    address public hfil;
    address public mfil;
    address public operator; // for start ido
    address public mdexFactory;

    event NewIDO(address indexed sender, address indexed ido);

    event OperatorshipTransferred(
        address indexed previousOperator,
        address indexed newOperator
    );

    constructor(
        address _operator,
        address _hfil,
        address _mfil,
        address _mdexFactory
    ) Ownable() {
```

Armors Labs

```
        operator = _operator;
        emit OperatorshipTransferred(address(0), operator);
        hfil = _hfil;
        mfil = _mfil;
        mdexFactory = _mdexFactory;
    }

    modifier onlyOperator() {
        require(
            _msgSender() == operator,
            "Operable: caller is not the operator"
        );
        _;
    }

    function transferOperatorship(address newOperator) external onlyOwner {
        require(
            newOperator != address(0),
            "Operable: new operator is the zero address"
        );
        emit OperatorshipTransferred(operator, newOperator);
        operator = newOperator;
    }

    function IDOlistCount() external view returns (uint256) {
        return IDOlist.length;
    }

    function userIDOsCount(address user) external view returns (uint256) {
        return userIDOs[user].length;
    }

    function _addLiquidity(
        address pair,
        address idoToken,
        uint256 idoAmount,
        uint256 mfilAmount
    ) internal {
        IERC20(idoToken).transfer(pair, idoAmount);
        TransferHelper.safeTransferFrom(mfil, _msgSender(), pair, mfilAmount);
        IMdexPair(pair).mint(address(0));
    }


    function newIDO(
        string calldata _name,
        string calldata _node,
        address _hfilRecipient,
        uint256 _cap,
        uint256 _gasPrice,
        uint256 _sealPrice,
        uint256 _hardDrivePrice,
        uint256 _dailySaleCap,
        uint256 _idoStartTime,
        uint256 _idoEndTime
    ) external returns (address idoAddress) {
        require(
            _dailySaleCap >= 1e6,
            "IDOFactory: dailysalecap should more than 1"
        );
        IDOToken ido = new IDOToken(
            _name,
            _node,
            hfil,
            _hfilRecipient,
            _cap,
```

Armors Labs

```
            _dailySaleCap
        );
        ido.setPrice(_gasPrice, _sealPrice, _hardDrivePrice);
        ido.setTimes(_idoStartTime, _idoEndTime);
        IDOlist.push(address(ido));
        userIDOs[_msgSender()].push(address(ido));
        address pair = IMdexFactory(mdexFactory).createPair(mfil, address(ido));
        uint256 totalPrice = _gasPrice.add(_sealPrice).add(_hardDrivePrice).div(100);
        _addLiquidity(pair, address(ido), 1e4, totalPrice);
        emit NewIDO(_msgSender(), address(ido));
        isIDO[address(ido)] = true;
        return address(ido);
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./Ownable.sol";
import "./Pausable.sol";
import "./ReentrancyGuard.sol";
import "./StakeRewardPerDay.sol";

contract StakeTokenPool is StakeRewardPerDay, Ownable, Pausable {

    using SafeMath for uint256;

    uint256 public BrewardRate; // Global Reward per Unit
    uint256 public BrewardPreBlock;
    uint256 public BlastUpdateBlock; // last adjust block height

    mapping(address => uint256) public BuserReward;
    mapping(address => uint256) public BuserRate;

    constructor() StakeRewardPerDay() {
        BrewardRate = 0;
        BrewardPreBlock = 0;
        BlastUpdateBlock = block.number;
    }

    modifier BupdateRate() {
        if (
            BrewardPreBlock > 0 &&
            totalStake > 0 &&
            block.number > BlastUpdateBlock
        ) {
            uint256 deltaRate =
                BrewardPreBlock.mul(block.number - BlastUpdateBlock).div(
                    totalStake
                );
            BrewardRate = BrewardRate.add(deltaRate);
        }
        BlastUpdateBlock = block.number;
        _;
    }

    modifier BgetReward() {
        if (
            userStake[_msgSender()] > 0 && BrewardRate > BuserRate[_msgSender()]
        ) {
            uint256 deltaReward =
                BrewardRate.sub(BuserRate[_msgSender()]).mul(
                    userStake[_msgSender()]
                );
            BuserReward[_msgSender()] = BuserReward[_msgSender()].add(
                deltaReward
```

Armors Labs

```
            );
        }
        BuserRate[_msgSender()] = BrewardRate;
        _;
    }

    function _setReward(uint256 amount) internal BupdateRate {
        BrewardPreBlock = amount.mul(1e18);
    }

    function _stake(uint256 amount) internal override updateRate getReward BupdateRate BgetReward {
        userStake[_msgSender()] = userStake[_msgSender()].add(amount);
        totalStake = totalStake.add(amount);
    }

    function _BhalveNoUpdate() internal returns (uint256 reward) {
        reward = BuserReward[_msgSender()].div(1e18);
        BuserReward[_msgSender()] = 0;
    }

    function _Bhalve() internal BupdateRate BgetReward returns (uint256 reward) {
        return _BhalveNoUpdate();
    }

    function _unStakeAll()
        internal
        updateRate
        getReward
        BupdateRate
        BgetReward
        returns (uint256 stake, uint256 reward, uint256 rewardB)
    {
        stake = userStake[_msgSender()];
        totalStake = totalStake.sub(stake);
        userStake[_msgSender()] = 0;
        reward = _halveNoUpdate();
        rewardB = _BhalveNoUpdate();
    }

    function _BcaleReward() internal view returns (uint256 reward) {
        if (userStake[_msgSender()] == 0) {
            return 0;
        }
        uint256 tempRate = BrewardRate;
        if (
            BrewardPreBlock > 0 &&
            totalStake > 0 &&
            block.number > BlastUpdateBlock
        ) {
            uint256 deltaRate =
                BrewardPreBlock.mul(block.number - BlastUpdateBlock).div(
                    totalStake
                );
            tempRate = tempRate.add(deltaRate);
        }
        reward = BuserReward[_msgSender()];
        if (tempRate > BuserRate[_msgSender()]) {
            uint256 deltaReward =
                tempRate.sub(BuserRate[_msgSender()]).mul(
                    userStake[_msgSender()]
                );
            reward = reward.add(deltaReward);
        }
        reward = reward.div(1e18);
    }
```

Armors Labs

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

pragma solidity ^0.7.6;

import "./StakeRewardPerDay.sol";
import "./MdexRouter.sol";
import "./Ownable.sol";
import "./Pausable.sol";
import "./ReentrancyGuard.sol";

contract StakeLPRewardPerDay is
    StakeRewardPerDay,
    MdexRouter,
    Ownable,
    Pausable,
    ReentrancyGuard
{
    using SafeMath for uint256;

    uint8 public stakeRate = 2;

    address public tokenA; // for one
    address public tokenB; // for stakeRate
    address public rewardToken; // for reward
    address public operator;

    mapping(address => uint256) public userTokenBStake;
    uint256 public totalTokenBStake;

    event Stake(
        address indexed sender,
        uint256 amountA,
        uint256 amountB,
        uint256 liquidity
    );
    event UnStake(
        address indexed receiver,
        uint256 amountA,
        uint256 amountB,
        uint256 liquidity
    );
    event Reward(address indexed receiver, uint256 reward);
    event OperatorshipTransferred(
        address indexed previousOperator,
        address indexed newOperator
    );
    event NewReward(uint256 blockHeight, uint256 amount);

    constructor(
        address _factory,
        address _operator,
        address _tokenA,
        address _tokenB,
        address _rewardToken
    )
        StakeRewardPerDay()
        MdexRouter(_factory)
        Ownable()
        Pausable()
        ReentrancyGuard()
    {
        operator = _operator;
        emit OperatorshipTransferred(address(0), operator);
        tokenA = _tokenA;
        tokenB = _tokenB;
        rewardToken = _rewardToken;
```

Armors Labs

```
    }

    modifier onlyOperator() {
        require(
            _msgSender() == operator,
            "Operable: caller is not the operator"
        );
        _;
    }

    function transferOperatorship(address newOperator) external onlyOwner {
        require(
            newOperator != address(0),
            "Operable: new operator is the zero address"
        );
        emit OperatorshipTransferred(operator, newOperator);
        operator = newOperator;
    }

    function pause() external onlyOperator {
        _pause();
    }

    function unPause() external onlyOperator {
        _unpause();
    }

    function caleLiquidity(
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin
    ) external view returns (uint256 amountA, uint256 amountB) {
        require(
            IMdexFactory(factory).getPair(tokenA, tokenB) != address(0),
            "MdexRouter: Pair Not Found!"
        );
        (uint256 reserveA, uint256 reserveB) =
            IMdexFactory(factory).getReserves(tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint256 amountBOptimal =
                IMdexFactory(factory).quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(
                    amountBOptimal >= amountBMin,
                    "MdexRouter: INSUFFICIENT_B_AMOUNT"
                );
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint256 amountAOptimal =
                    IMdexFactory(factory).quote(
                        amountBDesired,
                        reserveB,
                        reserveA
                    );
                assert(amountAOptimal <= amountADesired);
                require(
                    amountAOptimal >= amountAMin,
                    "MdexRouter: INSUFFICIENT_A_AMOUNT"
                );
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }
```

Armors Labs

```
function stake(uint256 amountAIn, uint256 amountBIn)
    external
    whenNotPaused
    nonReentrant
    returns (
        uint256 amountA,
        uint256 amountB,
        uint256 liquidity
    )
{
    require(amountAIn > 0 && amountBIn > 0, "STAKE: amount is zero");
    uint256 tempAmountB = amountBIn.div(stakeRate);
    (amountA, amountB, liquidity) = addLiquidity(
        tokenA,
        tokenB,
        amountAIn,
        tempAmountB,
        0,
        0
    );
    amountB = amountB.mul(stakeRate - 1);
    TransferHelper.safeTransferFrom(
        tokenB,
        _msgSender(),
        address(this),
        amountB
    );
    userTokenBStake[_msgSender()] = userTokenBStake[_msgSender()].add(
        amountB
    );
    totalTokenBStake = totalTokenBStake.add(amountB);
    _stake(liquidity);
    emit Stake(_msgSender(), amountA, amountB, liquidity);
    _stakeHook(amountAIn, amountBIn, amountA, amountB, liquidity);
}

function _stakeHook(
    uint256 amountAIn,
    uint256 amountBIn,
    uint256 amountA,
    uint256 amountB,
    uint256 liquidity
) internal virtual {}

function getStake() external view whenNotPaused returns (uint256 amount) {
    amount = _getStake();
}

function halve()
    external
    whenNotPaused
    nonReentrant
    returns (uint256 reward)
{
    reward = _halve();
    IERC20(rewardToken).transfer(_msgSender(), reward);
    emit Reward(_msgSender(), reward);
}

function unStake()
    external
    whenNotPaused
    nonReentrant
    returns (
        uint256 reward,
```

Armors Labs

```
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        )
    {
        (liquidity, reward) = _unStake();
        require(liquidity > 0, "UNSTAKE: no stake");
        IERC20(rewardToken).transfer(_msgSender(), reward);
        emit Reward(_msgSender(), reward);
        (amountA, amountB) = removeLiquidity(
            tokenA,
            tokenB,
            liquidity,
            0,
            0,
            _msgSender()
        );
        IERC20(tokenB).transfer(_msgSender(), userTokenBStake[_msgSender()]);
        amountB.add(userTokenBStake[_msgSender()]);
        userTokenBStake[_msgSender()] = 0;
        emit UnStake(_msgSender(), amountA, amountB, liquidity);
        _unStakeHook(reward, amountA, amountB, liquidity);
    }

    function _unStakeHook(
        uint256 reward,
        uint256 amountA,
        uint256 amountB,
        uint256 liquidity
    ) internal virtual {}

    function newReward(uint256 amount)
        external
        virtual
        onlyOperator
        whenNotPaused
    {
        _addNewReward(amount);
        emit NewReward(block.number, amount);
    }

    function caleReward() external view whenNotPaused returns (uint256 reward) {
        reward = _caleReward();
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./ERC20/ERC20.sol";
import "./ERC20/ERC20Burnable.sol";
import "./ERC20/ERC20Mintable.sol";
import "./libraries/TransferHelper.sol";

interface IIDOFactory {
    function operator() external view returns (address);
}

contract IDOToken is ERC20Burnable {
    using SafeMath for uint256;

    uint256 private _cap;
    uint256 public dailySaleCap;
    uint256 public price;
    uint256 public gasPrice;
    uint256 public sealPrice;
```

Armors Labs

```
    uint256 public hardDrivePrice;
    uint256 public idoStartTime;
    uint256 public idoEndTime;
    uint256 private _todayRemaind;
    uint256 public todayStartTime;
    address public hfil;
    address public hfilRecipient;
    address public router;
    address public factory;
    address public sender;

    string public node;

    bool public idoStatus;

    event IDOStart(uint256 timestamp);
    event IDOStop(uint256 timestamp, address sender);
    event IDO(address indexed to, uint256 cost, uint256 amount);

    constructor(
        string memory _name,
        string memory _node,
        address _hfil,
        address _hfilRecipient,
        uint256 cap_,
        uint256 _dailySaleCap
    ) ERC20(_name, "T") Pausable() {
        _setupDecimals(6);
        sender = tx.origin;
        factory = _msgSender();
        node = _node;
        hfil = _hfil;
        hfilRecipient = _hfilRecipient;
        _cap = cap_;
        dailySaleCap = _dailySaleCap;
        idoStatus = false;
        _mint(_msgSender(), 1 * 10**4);
    }

    modifier onlyOperator() {
        require(
            _msgSender() == IIDOFactory(factory).operator(),
            "Operable: caller is not the operator"
        );
        _;
    }

    modifier onlyFactory() {
        require(_msgSender() == factory, "IDOToken: only factory can call");
        _;
    }

    modifier onlySender() {
        require(_msgSender() == sender, "IDOToken: only sender can call");
        _;
    }

    modifier idoEnabled() {
        require(idoStatus, "IDOToken: ido not start yet");
        _;
    }

    modifier idoStarted() {
        require(block.timestamp >= idoStartTime, "IDOToken: ido not start yet");
        _;
    }
```

Armors Labs

```
    modifier idoNotEnded() {
        require(block.timestamp <= idoEndTime, "IDOToken: ido ended");
        _;
    }

    modifier updateToday() {
        if (block.timestamp >= todayStartTime + 86400) {
            todayStartTime = block.timestamp;
            _todayRemaind = dailySaleCap;
        }
        _;
    }

    function operator() view external returns (address) {
        return IIDOFactory(factory).operator();
    }

    function setPrice(
        uint256 gasPrice_,
        uint256 sealPrice_,
        uint256 hardDrivePrice_
    ) external onlyFactory {
        gasPrice = gasPrice_;
        sealPrice = sealPrice_;
        hardDrivePrice = hardDrivePrice_;
        price = gasPrice.add(sealPrice).add(hardDrivePrice);
    }

    function setTimes(uint256 idoStartTime_, uint256 idoEndTime_)
        external
        onlyFactory
    {
        idoStartTime = idoStartTime_;
        idoEndTime = idoEndTime_;
    }

    function changerouter(address _router) external onlyOperator {
        require(_router != address(0), "IDO: wrong address");
        router = _router;
    }

    function stopIdo() external onlyOperator {
        idoStatus = false;
        idoEndTime = block.timestamp;
        emit IDOStop(block.timestamp, _msgSender());
    }

    function stopIdoBySender() external onlySender {
        idoStatus = false;
        idoEndTime = block.timestamp;
        emit IDOStop(block.timestamp, _msgSender());
    }

    function todayRemaind() view external returns (uint256) {
        if (block.timestamp >= todayStartTime + 86400) {
            return dailySaleCap;
        }
        return _todayRemaind;
    }

    function pause() external onlyOperator {
        _pause();
    }

    function unPause() external onlyOperator {
```

Armors Labs

```
        _unpause();
    }

    /**
     * @dev Returns the cap on the token's total supply.
     */
    function cap() public view virtual returns (uint256) {
        return _cap;
    }

    /**
     * @dev See {ERC20-_beforeTokenTransfer}.
     *
     * Requirements:
     *
     * - minted tokens must not cause the total supply to go over the cap.
     */
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
        super._beforeTokenTransfer(from, to, amount);

        if (from == address(0)) {
            // When minting tokens
            require(totalSupply().add(amount) <= cap(), "ERC20: cap exceeded");
        }
    }

    function startIdo(address _router) external onlyOperator {
        require(!idoStatus, "IDO: ido already started!");
        require(_router != address(0), "IDO: wrong address");
        router = _router;
        idoStatus = true;
        emit IDOStart(block.timestamp);
        todayStartTime = block.timestamp;
        _todayRemaind = dailySaleCap;
    }

    function ido(address recipient)
        external
        idoEnabled
        idoStarted
        idoNotEnded
        updateToday
        returns (uint256 amount)
    {
        require(_todayRemaind > 0, "IDO: sold out today");
        require(_msgSender() == router, "IDO: not from fido");
        uint256 totalPrice = IERC20(hfil).balanceOf(address(this));
        uint256 sendBack = 0;
        require(totalPrice > 0, "IDO: no pay no gain");
        amount = totalPrice.mul(10 ** decimals()).div(price);
        require(amount > 0, "IDO: amount must greater than zero");
        if (amount >= _todayRemaind) {
            amount = _todayRemaind;
            sendBack = totalPrice.sub(amount.mul(price).div(10**decimals()));
        }
        _todayRemaind = _todayRemaind.sub(amount);
        if (sendBack > 0) {
            TransferHelper.safeTransfer(hfil, recipient, sendBack);
            totalPrice = totalPrice.sub(sendBack);
        }
        TransferHelper.safeTransfer(hfil, hfilRecipient, totalPrice);
        _mint(recipient, amount);
```

Armors Labs

```
        emit IDO(recipient, totalPrice, amount);
    }
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;

import "./libraries/SafeMath.sol";
import "./Context.sol";

contract StakeRewardPerBlock is Context {
    using SafeMath for uint256;

    uint256 public BrewardRate; // Global Reward per Unit
    uint256 public BrewardPreBlock;
    uint256 public BlastUpdateBlock; // last adjust block height
    uint256 public totalStake;

    mapping(address => uint256) public userStake;
    mapping(address => uint256) public BuserReward;
    mapping(address => uint256) public BuserRate;

    constructor() {
        BrewardRate = 0;
        BrewardPreBlock = 0;
        totalStake = 0;
        BlastUpdateBlock = block.number;
    }

    modifier BupdateRate() {
        if (
            BrewardPreBlock > 0 &&
            totalStake > 0 &&
            block.number > BlastUpdateBlock
        ) {
            uint256 deltaRate =
                BrewardPreBlock.mul(block.number - BlastUpdateBlock).div(
                    totalStake
                );
            BrewardRate = BrewardRate.add(deltaRate);
        }
        BlastUpdateBlock = block.number;
        _;
    }

    modifier BgetReward() {
        if (
            userStake[_msgSender()] > 0 && BrewardRate > BuserRate[_msgSender()]
        ) {
            uint256 deltaReward =
                BrewardRate.sub(BuserRate[_msgSender()]).mul(
                    userStake[_msgSender()]
                );
            BuserReward[_msgSender()] = BuserReward[_msgSender()].add(
                deltaReward
            );
        }
        BuserRate[_msgSender()] = BrewardRate;
        _;
    }

    function _setReward(uint256 amount) internal BupdateRate {
        BrewardPreBlock = amount.mul(1e18);
    }

    function _stake(uint256 amount) internal BupdateRate BgetReward {
```

Armors Labs

```
        userStake[_msgSender()] = userStake[_msgSender()].add(amount);
        totalStake = totalStake.add(amount);
    }

    function _BhalveNoUpdate() internal returns (uint256 reward) {
        reward = BuserReward[_msgSender()].div(1e18);
        BuserReward[_msgSender()] = 0;
    }

    function _Bhalve() internal BupdateRate BgetReward returns (uint256 reward) {
        return _BhalveNoUpdate();
    }

    function _unStake()
        internal
        BupdateRate
        BgetReward
        returns (uint256 stake, uint256 reward)
    {
        stake = userStake[_msgSender()];
        totalStake = totalStake.sub(stake);
        userStake[_msgSender()] = 0;
        reward = _BhalveNoUpdate();
    }

    function _getStake() internal view returns (uint256 stake) {
        stake = userStake[_msgSender()];
    }

    function _BcaleReward() internal view returns (uint256 reward) {
        if (userStake[_msgSender()] == 0) {
            return 0;
        }
        uint256 tempRate = BrewardRate;
        if (
            BrewardPreBlock > 0 &&
            totalStake > 0 &&
            block.number > BlastUpdateBlock
        ) {
            uint256 deltaRate =
                BrewardPreBlock.mul(block.number - BlastUpdateBlock).div(
                    totalStake
                );
            tempRate = tempRate.add(deltaRate);
        }
        reward = BuserReward[_msgSender()];
        if (tempRate > BuserRate[_msgSender()]) {
            uint256 deltaReward =
                tempRate.sub(BuserRate[_msgSender()]).mul(
                    userStake[_msgSender()]
                );
            reward = reward.add(deltaReward);
        }
        reward = reward.div(1e18);
    }
}
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

import "../Context.sol";
import "../interfaces/IERC20.sol";
import "../libraries/SafeMath.sol";

/**
 * @dev Implementation of the {IERC20} interface.
```

Armors Labs

```
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /**
     * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
     * a default value of 18.
     *
     * To select a different value for {decimals}, use {_setupDecimals}.
     *
     * All three of these values are immutable: they can only be set once during
     * construction.
     */
    constructor (string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
        _decimals = 18;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view virtual returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view virtual returns (string memory) {
        return _symbol;
    }

    /**
```

Armors Labs

```
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
     * called.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view virtual returns (uint8) {
        return _decimals;
    }

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view virtual override returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view virtual override returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override returns (uint256)
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
```

Armors Labs

```
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * Requirements:
     *
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC2
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
```

Armors Labs

```
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");
```

Armors Labs

```
        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal virtual {
        _decimals = decimals_;
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks]
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
}
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

import "../Context.sol";
import "./ERC20Pausable.sol";

/**
 * @dev Extension of {ERC20} that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in a way that can be
 * recognized off-chain (via event analysis).
 */
abstract contract ERC20Burnable is Context, ERC20Pausable {
    using SafeMath for uint256;

    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the caller's
     * allowance.
     *
     * See {ERC20-_burn} and {ERC20-allowance}.
     *
     * Requirements:
     *
     * - the caller must have allowance for ``accounts``'s tokens of at least
     * `amount`.
```

Armors Labs

```
     */
    function burnFrom(address account, uint256 amount) public virtual {
        uint256 decreasedAllowance = allowance(account, _msgSender()).sub(amount, "ERC20: burn amount

        _approve(account, _msgSender(), decreasedAllowance);
        _burn(account, amount);
    }
}
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

import "../Context.sol";
import "./ERC20Pausable.sol";

abstract contract ERC20Mintable is Context, ERC20Pausable {
    using SafeMath for uint256;

    mapping(address => bool) public isMinter;

    event AddMinter(address indexed minter);
    event RemoveMinter(address indexed minter);
    event Mint(
        address indexed minter,
        address indexed recipient,
        uint256 amount
    );

    modifier onlyMinter() {
        require(isMinter[_msgSender()], "ERC20: sender is not minter");
        _;
    }

    function _addMinter(address minter) internal {
        require(!isMinter[minter], "ERC20: already a minter");
        isMinter[minter] = true;
        emit AddMinter(minter);
    }

    function _removeMinter(address minter) internal {
        require(isMinter[minter], "ERC20: not a minter");
        isMinter[minter] = false;
        emit RemoveMinter(minter);
    }

    function mint(address recipient, uint256 amount) external onlyMinter {
        _mint(recipient, amount);
        emit Mint(_msgSender(), recipient, amount);
    }
}
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

import "./ERC20.sol";
import "../Pausable.sol";

/**
 * @dev ERC20 token with pausable token transfers, minting and burning.
 *
 * Useful for scenarios such as preventing trades until the end of an evaluation
 * period, or having an emergency switch for freezing all token transfers in the
 * event of a large bug.
 */
abstract contract ERC20Pausable is ERC20, Pausable {
    /**
```

Armors Labs

```
     * @dev See {ERC20-_beforeTokenTransfer}.
     *
     * Requirements:
     *
     * - the contract must not be paused.
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual override
        super._beforeTokenTransfer(from, to, amount);

        require(!paused(), "ERC20Pausable: token transfer while paused");
    }
}
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

import "./Ownable.sol";

contract IDOInfo is Ownable {
    address public operator;

    mapping(address => bool) public exist;
    mapping(address => bool) public isPool;
    address[] public IDOList;
    mapping(address => address) public payer2pool;
    mapping(address => address) public stakeAddress;
    mapping(address => address) public pool2idoToken;

    event OperatorshipTransferred(
        address indexed previousOperator,
        address indexed newOperator
    );

    event AddIDO(uint256 timestamp, address idoToken);

    event SetPayer(uint256 timestamp, address pool, address payer);

    event SetStakeAddress(
        uint256 timestamp,
        address idoToken,
        address _stakeAddress
    );

    constructor(address _operator) Ownable() {
        operator = _operator;
        emit OperatorshipTransferred(address(0), operator);
    }

    modifier onlyOperator() {
        require(
            _msgSender() == operator,
            "Operable: caller is not the operator"
        );
        _;
    }

    function transferOperatorship(address newOperator) external onlyOwner {
        require(
            newOperator != address(0),
            "Operable: new operator is the zero address"
        );
        emit OperatorshipTransferred(operator, newOperator);
        operator = newOperator;
    }

    function IDOListCount() external view returns (uint256) {
        return IDOList.length;
```

Armors Labs

```
    }

    function addIDO(address idoToken) external onlyOperator {
        require(!exist[idoToken], "IDOINFO: already here");
        exist[idoToken] = true;
        IDOList.push(idoToken);
        emit AddIDO(block.timestamp, idoToken);
    }

    function setPayer(address pool, address payer) external onlyOperator {
        require(isPool[pool], "IDOINFO: pool not found");
        payer2pool[payer] = pool;
        emit SetPayer(block.timestamp, pool, payer);
    }

    function setStakeAddress(address idoToken, address _stakeAddress)
        external
        onlyOperator
    {
        require(exist[idoToken], "IDOINFO: idoToken not found");
        isPool[_stakeAddress] = true;
        pool2idoToken[_stakeAddress] = idoToken;
        stakeAddress[idoToken] = _stakeAddress;
        emit SetStakeAddress(block.timestamp, idoToken, _stakeAddress);
    }
}
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

abstract contract ReentrancyGuard {

    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    constructor () {
        _status = _NOT_ENTERED;
    }

    modifier nonReentrant() {
        // On the first call to nonReentrant, _notEntered will be true
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");

        // Any calls to nonReentrant after this point will fail
        _status = _ENTERED;

        _;

        // By storing the original value once again, a refund is triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _status = _NOT_ENTERED;
    }
}
```

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 specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

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

```
PASSED!
```

- **Security suggestion:**
  no.

Armors Labs

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

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

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

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

```
PASSED!
```

- **Security suggestion:**
  no.

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.

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.

Armors Labs

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.

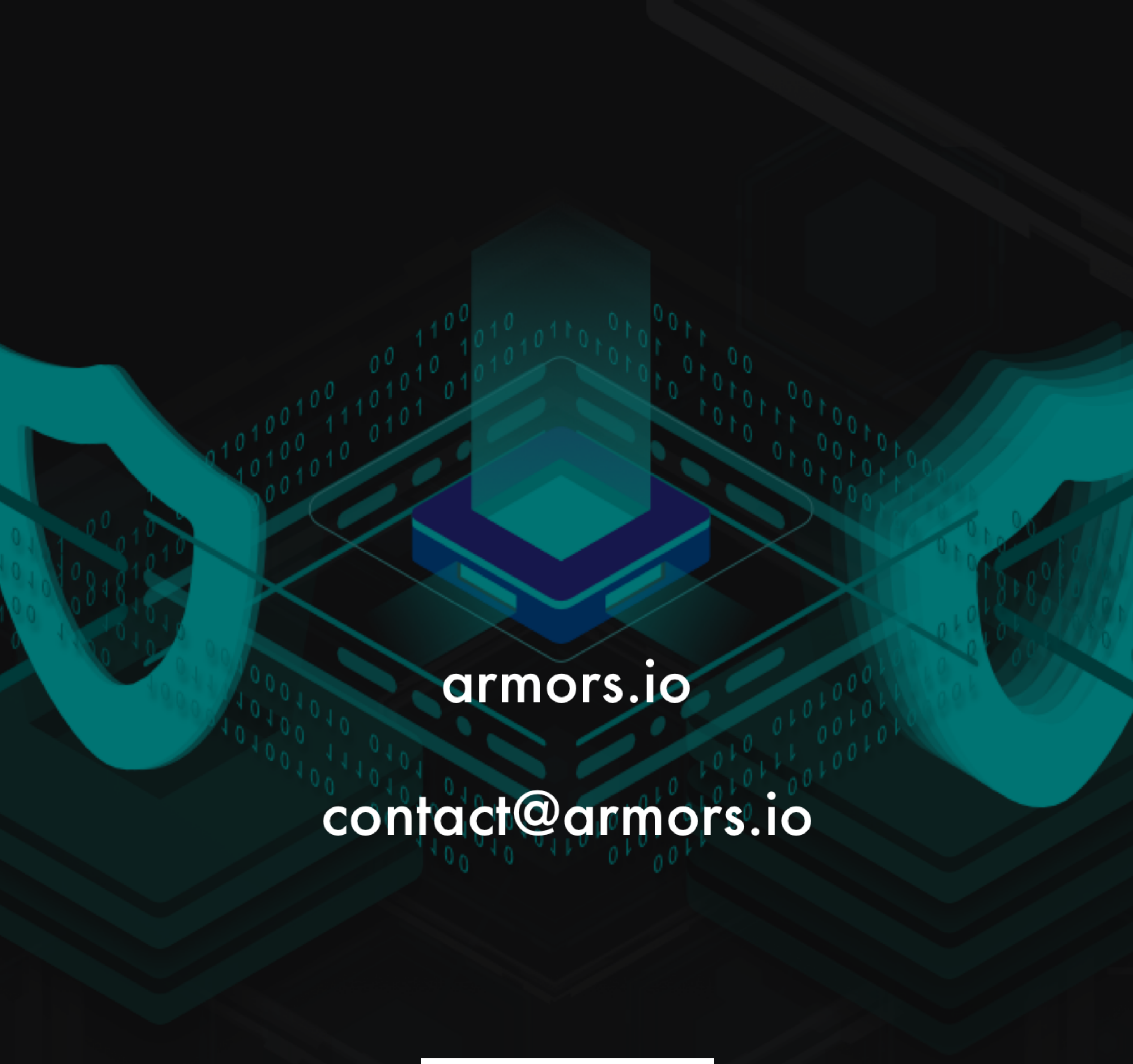
Armors Labs

Permission restrictions

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

```
PASSED!
```

- **Security suggestion:**
  no.



armors.io

contact@armors.io