
# Bright Union Staking

## Smart Contract Audit

# Bright Union Staking Audit Summary

Project name : Bright Union Staking Contract

Project address: None

Code URL : https://etherscan.io/address/0x4b22e0273988bc592cb17e7caf14275e914a83bc#code

Code URL : https://etherscan.io/address/0x831dadaa0e6cdc250cbcd76bae6e441a46423b94#code

Code URL : https://etherscan.io/address/0x6f2fc1ad87185bb4149250cdded68e80f9c4bb26#code

Commit : None

Project target : Bright Union Staking Contract Audit

Blockchain : Ethereum

Test result : PASSED

Audit Info

Audit NO : 0X202108100019

Audit Team : Armors Labs

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

# Bright Union Staking Audit

The Bright Union Staking team asked us to review and audit their Bright Union Staking 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 |
| --- | --- | --- | --- |
| Bright Union Staking Audit | Rock, Sophia, Rushairer, Rico, David, Alice | 1.0.0 | 2021-08-10 |

### Audit results

Note that: This contract is upgradable and this audit is only valid for the current version.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Bright Union Staking 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 |
|---|---|
| IContractsRegistry.sol | 960f2890d07897bb0a2d282577ff6cd1 |
| Initializable.sol | 1f13c687b18a86375d530df0409def76 |
| IERC20.sol | 9a02a460e2e968db8dd4da7e7915ce3e |
| ReentrancyGuard.sol | caac637d3d236beea195c32d56842c69 |
| IERC20Permit.sol | f6af9c0d72a4cd21c7de0172b925564c |
| BrightStaking.sol | f7dd84cd40e86f526a097bd7e3d4326a |
| UpgradeableProxy.sol | 2794572b935e51918a554b7e134a7e4c |
| TransparentUpgradeableProxy.sol | 92fdfbb4f9ef925a55a159ec6f3a64fc |
| SafeERC20.sol | e9d17630956bdefb7f97106aad7e04b0 |
| SafeMath.sol | 5528da796ebf3c244f6025e5cf14a300 |
| IBrightStaking.sol | 6b2e3c050b10ceae10ab1dae28d2bc95 |
| Address.sol | 059d94823a0a61c585f3b93cff67ad76 |
| AccessControlUpgradeable.sol | a1551dd0721eeb9d0c1cd9223abd1504 |
| OwnableUpgradeable.sol | a20480ea74644e30cf2b2ed448bc1cad |
| ContextUpgradeable.sol | 19b266095cd60dceae050fa06d9db09c |
| ISTKBrightToken.sol | 6587410d3722fb0b871257850aaa0d12 |
| Proxy.sol | b103cd7bb9c483de6878f01a4d6a234f |
| ContractsRegistry.sol | 3a21edc700b1ff86b93f2a28d412aee6 |
| Globals.sol | 3cf4341c6071e880c45742012e06d763 |
| SafeMathUpgradeable.sol | 0c2fb1693b700e88ae1e43dbde2653a9 |
| Math.sol | 1b7eea4eebd06232f1607d281f3baf65 |
| LiquidityMiningStaking.sol | 1f5210f2c71284836fe5bc2e5616e9e7 |
| EnumerableSetUpgradeable.sol | 31f68a69eb47ac482ef5a79eb980f259 |
| IUniswapV2Pair.sol | 1d4d8e255a202a54b00503590bd11e30 |
| SafeERC20Upgradeable.sol | 2442edebc0104e422540ff253b5f2487 |
| AddressUpgradeable.sol | 0203807a078514459a0e0fd04ac473a0 |

| file | md5 |
| --- | --- |
| IERC20Upgradeable.sol | 49761febab52e18b9ab47fcc605f7003 |
| ILiquidityMiningStaking.sol | b8f9b32d49c222b0b5e6e5b38c5eeaab |
| IAbstractCooldownStaking.sol | 0af5258388553946370979cc96258a84 |
| AbstractCooldownStaking.sol | 9675c7d6cfb9e05db999afd2d81a6187 |

# Vulnerability analysis

## Vulnerability distribution

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

## Summary of audit results

| Vulnerability | status |
| --- | --- |
| Re-Entrancy | safe |
| Arithmetic Over/Under Flows | safe |
| Unexpected Blockchain Currency | safe |
| Delegatecall | safe |
| Default Visibilities | safe |
| Entropy Illusion | safe |
| External Contract Referencing | safe |
| Short Address/Parameter Attack | safe |
| Unchecked CALL Return Values | safe |
| Race Conditions / Front Running | safe |
| Denial Of Service (DOS) | safe |
| Block Timestamp Manipulation | safe |
| Constructors with Care | safe |
| Unintialised Storage Pointers | safe |
| Floating Points and Numerical Precision | safe |
| tx.origin Authentication | safe |

| Vulnerability | status |
| --- | --- |
| Permission restrictions | safe |

## Contract file

IContractsRegistry.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.4;

interface IContractsRegistry {

    function getUniswapBrightToETHPairContract() external view returns (address);

    function getBrightContract() external view returns (address);

    function getBrightStakingContract() external view returns (address);

    function getSTKBrightContract() external view returns (address);

    function getLiquidityMiningStakingContract() external view returns (address);

}
```

Initializable.sol

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

// solhint-disable-next-line compiler-version
pragma solidity >=0.4.24 <0.8.0;

import "../utils/AddressUpgradeable.sol";

/**
 * @dev This is a base contract to aid in writing upgradeable contracts, or any kind of contract that
 * behind a proxy. Since a proxied contract can't have a constructor, it's common to move constructor
 * external initializer function, usually called `initialize`. It then becomes necessary to protect t
 * function so it can only be called once. The {initializer} modifier provided by this contract will
 *
 * TIP: To avoid leaving the proxy in an uninitialized state, the initializer function should be call
 * possible by providing the encoded function call as the `_data` argument to {UpgradeableProxy-const
 *
 * CAUTION: When used with inheritance, manual care must be taken to not invoke a parent initializer
 * that all initializers are idempotent. This is not verified automatically as constructors are by So
 */
abstract contract Initializable {

    /**
     * @dev Indicates that the contract has been initialized.
     */
    bool private _initialized;

    /**
     * @dev Indicates that the contract is in the process of being initialized.
     */
    bool private _initializing;

    /**
     * @dev Modifier to protect an initializer function from being invoked twice.
     */
```

```
    modifier initializer() {
        require(_initializing || _isConstructor() || !_initialized, "Initializable: contract is alrea

        bool isTopLevelCall = !_initializing;
        if (isTopLevelCall) {
            _initializing = true;
            _initialized = true;
        }

        _;

        if (isTopLevelCall) {
            _initializing = false;
        }
    }

    /// @dev Returns true if and only if the function is running in the constructor
    function _isConstructor() private view returns (bool) {
        return !AddressUpgradeable.isContract(address(this));
    }
}
```

IERC20.sol

```
// 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 {
    /**
     * @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.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

ReentrancyGuard.sol

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

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Contract module that helps prevent reentrant calls to a function.
 *
 * Inheriting from `ReentrancyGuard` will make the {nonReentrant} modifier
 * available, which can be applied to functions to make sure there are no nested
 * (reentrant) calls to them.
 *
 * Note that because there is a single `nonReentrant` guard, functions marked as
 * `nonReentrant` may not call one another. This can be worked around by making
 * those functions `private`, and then adding `external` `nonReentrant` entry
 * points to them.
 *
 * TIP: If you would like to learn more about reentrancy and alternative ways
 * to protect against it, check out our blog post
 * https://blog.openzeppelin.com/reentrancy-after-istanbul/[Reentrancy After Istanbul].
 */
abstract contract ReentrancyGuard {
    // Booleans are more expensive than uint256 or any type that takes up a full
```

```
    // word because each write operation emits an extra SLOAD to first read the
    // slot's contents, replace the bits taken up by the boolean, and then write
    // back. This is the compiler's defense against contract upgrades and
    // pointer aliasing, and it cannot be disabled.

    // The values being non-zero value makes deployment a bit more expensive,
    // but in exchange the refund on every call to nonReentrant will be lower in
    // amount. Since refunds are capped to a percentage of the total
    // transaction's gas, it is best to keep them low in cases like this one, to
    // increase the likelihood of the full refund coming into effect.
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    constructor () internal {
        _status = _NOT_ENTERED;
    }

    /**
     * @dev Prevents a contract from calling itself, directly or indirectly.
     * Calling a `nonReentrant` function from another `nonReentrant`
     * function is not supported. It is possible to prevent this from happening
     * by making the `nonReentrant` function external, and make it call a
     * `private` function that does the actual work.
     */
    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;
    }
}
```

IERC20Permit.sol

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

pragma solidity =0.7.4;

/**
 * @dev Interface of the ERC20 Permit extension allowing approvals to be made via si
 * https://eips.ethereum.org/EIPS/eip-2612[EIP-2612].
 *
 * Adds the {permit} method, which can be used to change an
 * presenting a message signed by the account. By not rel
 * need to send a transaction, and thus is not required to hold Ether at all.
 */
interface IERC20Permit {
    /**
     * @dev Sets `value` as the allowance of `spender` over `owner`'s tokens,
     * given `owner`'s signed approval.
     *
```

```
     * IMPORTANT: The same issues {IERC20-approve} has related to transaction
     * ordering also apply here.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `deadline` must be a timestamp in the future.
     * - `v`, `r` and `s` must be a valid `secp256k1` signature from `owner`
     * over the EIP712-formatted function arguments.
     * - the signature must use ``owner``'s current nonce (see {nonces}).
     *
     * For more information on the signature format, see
     * https://eips.ethereum.org/EIPS/eip-2612#specification[relevant EIP
     * section].
     */
    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;

    /**
     * @dev Returns the current nonce for `owner`. This value must be
     * included whenever a signature is generated for {permit}.
     *
     * Every successful call to {permit} increases ``owner``'s nonce by one. This
     * prevents a signature from being used multiple times.
     */
    function nonces(address owner) external view returns (uint256);

    /**
     * @dev Returns the domain separator used in the encod
     */
    // solhint-disable-next-line func-name-mixedcase
    function DOMAIN_SEPARATOR() external view returns (bytes32);
}
```

BrightStaking.sol

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

pragma solidity ^0.7.4;

import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

import "./interfaces/IContractsRegistry.sol";
import "./interfaces/IBrightStaking.sol";
```

```
import "./interfaces/token/ISTKBrightToken.sol";
import "./interfaces/token/IERC20Permit.sol";
import "./Globals.sol";
import "./AbstractCooldownStaking.sol";

contract BrightStaking is IBrightStaking, AbstractCooldownStaking, OwnableUpgradeable {
    using SafeMath for uint256;

    IERC20 public brightToken;
    ISTKBrightToken public override stkBrightToken;
    uint256 public lastUpdateBlock;
    uint256 public rewardPerBlock;
    uint256 public totalPool;

    modifier updateRewardPool() {
        if (totalPool == 0) {
            lastUpdateBlock = block.number;
        }

        totalPool = totalPool.add(_calculateReward());
        lastUpdateBlock = block.number;
        _;
    }

    receive() payable external {
        revert('BrightStk: No ETH here');
    }

    function __BrightStaking_init(uint256 _rewardPerBlock, IContractsRegistry _contractsRegistry)
        external
        initializer
    {
        __Ownable_init();

        rewardPerBlock = _rewardPerBlock;

        brightToken = IERC20(_contractsRegistry.getBrightContract());
        stkBrightToken = ISTKBrightToken(_contractsRegistry.getSTKBrightContract());
    }

    function stake(uint256 _amountBright) external override updateRewardPool {
        brightToken.transferFrom(_msgSender(), address(this), _amountBright);

        _stake(_msgSender(), _amountBright);
    }

    /**
     * @dev Caller is the actual staker but stakes for _user
     */
    function stakeFor(address _user, uint256 _amountBright) external override updateRewardPool {
        brightToken.transferFrom(_msgSender(), address(this), _amountBright);

        _stake(_user, _amountBright);
    }

    function stakeWithPermit(uint256 _amountBright, uint8 _v, bytes32 _r, bytes32 _s) external overri
        IERC20Permit(address(brightToken)).permit(
            _msgSender(),
            address(this),
            _amountBright,
            MAX_INT,
            _v,
            _r,
            _s
        );
```

```
        brightToken.transferFrom(_msgSender(), address(this), _amountBright);
        _stake(_msgSender(), _amountBright);
    }

    function _stake(address _staker, uint256 _amountBright) internal {
        require(_amountBright > 0, "BrightStk: cant stake 0 tokens");
        uint256 amountStkBright = _convertToStkBright(_amountBright);
        stkBrightToken.mint(_staker, amountStkBright);

        totalPool = totalPool.add(_amountBright);

        emit StakedBright(_amountBright, amountStkBright, _staker);
    }

    function callWithdraw(uint256 _amountStkBrightUnlock) external override {
        require(_amountStkBrightUnlock > 0, "BrightStk: can't unlock 0 tokens");

        require(
            stkBrightToken.balanceOf(_msgSender()) >= _amountStkBrightUnlock,
            "BrightStk: not enough stkBright to unlock"
        );

        withdrawalsInfo[_msgSender()] = WithdrawalInfo(
            block.timestamp.add(WITHDRAWING_COOLDOWN_DURATION),
            _amountStkBrightUnlock
        );
    }

    function withdraw() external override updateRewardPool {
        uint256 _whenCanWithdrawBrightReward = whenCanWithdrawBrightReward(_msgSender());
        require(_whenCanWithdrawBrightReward != 0, "BrightStk: unlock not started/exp");
        require(_whenCanWithdrawBrightReward <= block.timestamp, "BrightStk: cooldown not reached");

        uint256 _amountStkBright = withdrawalsInfo[_msgSender()].amount;
        delete withdrawalsInfo[_msgSender()];

        require(
            stkBrightToken.balanceOf(_msgSender()) >= _amountStkBright,
            "BrightStk: not enough stkBright tokens to withdraw"
        );

        uint256 amountBright = _convertToBright(_amountStkBright);
        require(
            brightToken.balanceOf(address(this)) >= amountBright,
            "BrightStk: not enough Bright tokens in the pool"
        );
        stkBrightToken.burn(_msgSender(), _amountStkBright);

        totalPool = totalPool.sub(amountBright);

        brightToken.transfer(_msgSender(), amountBright);

        emit WithdrawnBright(amountBright, _amountStkBright, _msgSender());
    }

    function stakingReward(uint256 _amount) external view override returns (uint256) {
        return _convertToBright(_amount);
    }

    function getStakedBright(address _address) external view override returns (uint256) {
        uint256 balance = stkBrightToken.balanceOf(_address);
        return balance > 0 ? _convertToBright(balance) : 0;
    }

    function setRewardPerBlock(uint256 _amount) external override onlyOwner updateRewardPool {
```

```
        rewardPerBlock = _amount;
    }

    function sweepUnusedRewards() external override onlyOwner updateRewardPool {
        uint256 contractBalance = brightToken.balanceOf(address(this));

        require(
            contractBalance > totalPool,
            "BrightStk: There are no unused tokens to revoke"
        );

        uint256 unusedTokens = contractBalance.sub(totalPool);

        brightToken.transfer(_msgSender(), unusedTokens);
        emit SweepedUnusedRewards(_msgSender(), unusedTokens);
    }

    function outstandingRewards() external view returns (uint256) {
        return _calculateReward();
    }

    function _convertToStkBright(uint256 _amount) internal view returns (uint256) {
        uint256 tStkBrightToken = stkBrightToken.totalSupply();
        uint256 stakingPool = totalPool.add(_calculateReward());

        if (stakingPool > 0 && tStkBrightToken > 0) {
            _amount = tStkBrightToken.mul(_amount).div(stakingPool);
        }

        return _amount;
    }

    function _convertToBright(uint256 _amount) internal view returns (uint256) {
        uint256 tStkBrightToken = stkBrightToken.totalSupply();
        uint256 stakingPool = totalPool.add(_calculateReward());

        return tStkBrightToken > 0 ? stakingPool.mul(_amount).div(tStkBrightToken) : 0;
    }

    function _calculateReward() internal view returns (uint256) {
        uint256 blocksPassed = block.number.sub(lastUpdateBlock);
        return rewardPerBlock.mul(blocksPassed);
    }
}
```



UpgradeableProxy.sol

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

pragma solidity >=0.6.0 <0.8.0;

import "./Proxy.sol";
import "../utils/Address.sol";

/**
 * @dev This contract implements an upgradeable proxy. It is upgradeable because calls are delegated
 * implementation address that can be changed. This address is stored in storage in the location spec
 * https://eips.ethereum.org/EIPS/eip-1967[EIP1967], so that it doesn't conflict with the storage lay
 * implementation behind the proxy.
 *
 * Upgradeability is only provided internally through {_upgradeTo}. For an externally upgradeable pro
 * {TransparentUpgradeableProxy}.
 */
```

```
contract UpgradeableProxy is Proxy {
    /**
     * @dev Initializes the upgradeable proxy with an initial implementation specified by `_logic`.
     *
     * If `_data` is nonempty, it's used as data in a delegate call to `_logic`. This will typically
     * function call, and allows initializating the storage of the proxy like a Solidity constructor.
     */
    constructor(address _logic, bytes memory _data) public payable {
        assert(_IMPLEMENTATION_SLOT == bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1
        _setImplementation(_logic);
        if(_data.length > 0) {
            Address.functionDelegateCall(_logic, _data);
        }
    }

    /**
     * @dev Emitted when the implementation is upgraded.
     */
    event Upgraded(address indexed implementation);

    /**
     * @dev Storage slot with the address of the current implementation.
     * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, and is
     * validated in the constructor.
     */
    bytes32 private constant _IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a

    /**
     * @dev Returns the current implementation address.
     */
    function _implementation() internal view virtual override returns (address impl) {
        bytes32 slot = _IMPLEMENTATION_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            impl := sload(slot)
        }
    }

    /**
     * @dev Upgrades the proxy to a new implementation.
     *
     * Emits an {Upgraded} event.
     */
    function _upgradeTo(address newImplementation) internal virtual {
        _setImplementation(newImplementation);
        emit Upgraded(newImplementation);
    }

    /**
     * @dev Stores a new address in the EIP1967 implementation slot.
     */
    function _setImplementation(address newImplementation) private {
        require(Address.isContract(newImplementation), "UpgradeableProxy: new implementation is not a

        bytes32 slot = _IMPLEMENTATION_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newImplementation)
        }
    }
}
```

TransparentUpgradeableProxy.sol

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

pragma solidity >=0.6.0 <0.8.0;

import "./UpgradeableProxy.sol";

/**
 * @dev This contract implements a proxy that is upgradeable by an admin.
 *
 * To avoid https://medium.com/nomic-labs-blog/malicious-backdoors-in-ethereum-proxies-62629adf3357[p
 * clashing], which can potentially be used in an attack, this contract uses the
 * https://blog.openzeppelin.com/the-transparent-proxy-pattern/[transparent proxy pattern]. This patt
 * things that go hand in hand:
 *
 * 1. If any account other than the admin calls the proxy, the call will be forwarded to the implemen
 * that call matches one of the admin functions exposed by the proxy itself.
 * 2. If the admin calls the proxy, it can access the admin functions, but its calls will never be fo
 * implementation. If the admin tries to call a function on the implementation it will fail with an e
 * "admin cannot fallback to proxy target".
 *
 * These properties mean that the admin account can only be used for admin actions like upgrading the
 * the admin, so it's best if it's a dedicated account that is not used for anything else. This will
 * to sudden errors when trying to call a function from the proxy implementation.
 *
 * Our recommendation is for the dedicated account to be an instance of the {ProxyAdmin} contract. If
 * you should think of the `ProxyAdmin` instance as the real administrative interface of your proxy.
 */
contract TransparentUpgradeableProxy is UpgradeableProxy {
    /**
     * @dev Initializes an upgradeable proxy managed by `_admin`, backed by the implementation at `_l
     * optionally initialized with `_data` as explained in {UpgradeableProxy-constructor}.
     */
    constructor(address _logic, address admin_, bytes memory _data) public payable UpgradeableProxy(_
        assert(_ADMIN_SLOT == bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1));
        _setAdmin(admin_);
    }

    /**
     * @dev Emitted when the admin account has changed.
     */
    event AdminChanged(address previousAdmin, address newAdmin);

    /**
     * @dev Storage slot with the admin of the contract.
     * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
     * validated in the constructor.
     */
    bytes32 private constant _ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a71785

    /**
     * @dev Modifier used internally that will delegate the call to the implementation unless the sen
     */
    modifier ifAdmin() {
        if (msg.sender == _admin()) {
            _;
        } else {
            _fallback();
        }
    }

    /**
     * @dev Returns the current admin.
     *
     * NOTE: Only the admin can call this function. See {ProxyAdmin-getProxyAdmin}.
     *
```

```
     * TIP: To get this value clients can read directly from the storage slot shown below (specified
     * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
     * `0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103`
     */
    function admin() external ifAdmin returns (address admin_) {
        admin_ = _admin();
    }

    /**
     * @dev Returns the current implementation.
     *
     * NOTE: Only the admin can call this function. See {ProxyAdmin-getProxyImplementation}.
     *
     * TIP: To get this value clients can read directly from the storage slot shown below (specified
     * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
     * `0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc`
     */
    function implementation() external ifAdmin returns (address implementation_) {
        implementation_ = _implementation();
    }

    /**
     * @dev Changes the admin of the proxy.
     *
     * Emits an {AdminChanged} event.
     *
     * NOTE: Only the admin can call this function. See {ProxyAdmin-changeProxyAdmin}.
     */
    function changeAdmin(address newAdmin) external virtual ifAdmin {
        require(newAdmin != address(0), "TransparentUpgradeableProxy: new admin is the zero address")
        emit AdminChanged(_admin(), newAdmin);
        _setAdmin(newAdmin);
    }

    /**
     * @dev Upgrade the implementation of the proxy.
     *
     * NOTE: Only the admin can call this function. See {ProxyAdmin-upgrade}.
     */
    function upgradeTo(address newImplementation) external virtual ifAdmin {
        _upgradeTo(newImplementation);
    }

    /**
     * @dev Upgrade the implementation of the proxy, and then call a function from the new implementa
     * by `data`, which should be an encoded function call. This is useful to initialize new storage
     * proxied contract.
     *
     * NOTE: Only the admin can call this function. See {ProxyAdmin-upgradeAndCall}.
     */
    function upgradeToAndCall(address newImplementation, bytes calldata data) external payable virtua
        _upgradeTo(newImplementation);
        Address.functionDelegateCall(newImplementation, data);
    }

    /**
     * @dev Returns the current admin.
     */
    function _admin() internal view virtual returns (address adm) {
        bytes32 slot = _ADMIN_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            adm := sload(slot)
        }
    }
```

```
    /**
     * @dev Stores a new address in the EIP1967 admin slot.
     */
    function _setAdmin(address newAdmin) private {
        bytes32 slot = _ADMIN_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, newAdmin)
        }
    }

    /**
     * @dev Makes sure the admin cannot access the fallback function. See {Proxy-_beforeFallback}.
     */
    function _beforeFallback() internal virtual override {
        require(msg.sender != _admin(), "TransparentUpgradeableProxy: admin cannot fallback to proxy
        super._beforeFallback();
    }
}
```

SafeERC20.sol

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

pragma solidity >=0.6.0 <0.8.0;

import "./IERC20.sol";
import "../../math/SafeMath.sol";
import "../../utils/Address.sol";

/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

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

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

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
```

```
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
        // the target address contains contract code and also asserts for success in the low-level ca

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

SafeMath.sol

```
// 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);
    }

    /**
     * @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).
     *
     * 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).
     *
     * 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;
    }
}
```

IBrightStaking.sol

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

pragma solidity ^0.7.4;
```

```
import "./token/ISTKBrightToken.sol";
import "./IAbstractCooldownStaking.sol";

interface IBrightStaking is IAbstractCooldownStaking{
       event StakedBright(
        uint256 stakedBright,
        uint256 mintedStkBright,
        address indexed recipient
    );

    event WithdrawnBright(
        uint256 withdrawnBright,
        uint256 burnedStkBright,
        address indexed recipient
    );

    event SweepedUnusedRewards(address recipient, uint256 amount);

       function stkBrightToken() external returns (ISTKBrightToken);

       function stake(uint256 _amountBright) external;

    function stakeFor(address _user, uint256 _amountBright) external;

    function stakeWithPermit(uint256 _amountBright, uint8 _v, bytes32 _r, bytes32 _s) external;

    function callWithdraw(uint256 _amountStkBrightUnlock) external;

       function withdraw() external;

       function stakingReward(uint256 _amount) external view returns (uint256);

       function getStakedBright(address _address) external view returns (uint256);

       function setRewardPerBlock(uint256 _amount) external;

    function sweepUnusedRewards() external;
}
```

Address.sol

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

pragma solidity >=0.6.2 <0.8.0;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
```

```
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

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

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
```

```
    function functionCall(address target, bytes memory data, string memory errorMessage) internal ret
        return functionCallWithValue(target, data, 0, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValu
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory er
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: value }(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memor
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data, string memory errorMessage) intern
        require(isContract(target), "Address: static call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.staticcall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
```

```
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) inte
        require(isContract(target), "Address: delegate call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _verifyCallResult(bool success, bytes memory returndata, string memory errorMessage) pri
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}
```

AccessControlUpgradeable.sol

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

pragma solidity >=0.6.0 <0.8.0;

import "../utils/EnumerableSetUpgradeable.sol";
import "../utils/AddressUpgradeable.sol";
import "../utils/ContextUpgradeable.sol";
import "../proxy/Initializable.sol";

/**
 * @dev Contract module that allows children to implement role-based access
 * control mechanisms.
 *
 * Roles are referred to by their `bytes32` identifier. These should be exposed
 * in the external API and be unique. The best way to achieve this is by
 * using `public constant` hash digests:
 *
 *
```

- bytes32 public constant MY_ROLE = keccak256("MY_ROLE");
- `*
- Roles can be used to represent a set of permissions. To restrict access to a

- function call, use {hasRole}: *
- `
- function foo() public {
- require(hasRole(MY_ROLE, msg.sender));
- ...
- }
- `*
- Roles can be granted and revoked dynamically via the {grantRole} and
- {revokeRole} functions. Each role has an associated admin role, and only
- accounts that have a role's admin role can call {grantRole} and {revokeRole}. *
- By default, the admin role for all roles is `DEFAULT_ADMIN_ROLE`, which means
- that only accounts with this role will be able to grant or revoke other
- roles. More complex role relationships can be created by using
- {_setRoleAdmin}. *
- WARNING: The `DEFAULT_ADMIN_ROLE` is also its own admin: it has permission to
- grant and revoke this role. Extra precautions should be taken to secure
- accounts that have been granted it. */ abstract contract AccessControlUpgradeable is Initializable, ContextUpgradeable { function _*AccessControl*init() internal initializer {

```
__Context_init_unchained();
__AccessControl_init_unchained();
```

}

function _*AccessControl*init_unchained() internal initializer { } using EnumerableSetUpgradeable for EnumerableSetUpgradeable.AddressSet; using AddressUpgradeable for address;

struct RoleData {

```
EnumerableSetUpgradeable.AddressSet members;
bytes32 adminRole;
```

}

mapping (bytes32 => RoleData) private _roles;

bytes32 public constant DEFAULT_ADMIN_ROLE = 0x00;

/**

- @dev Emitted when `newAdminRole` is set as `role`'s admin role, replacing `previousAdminRole` *

- `DEFAULT_ADMIN_ROLE` is the starting admin for all roles, despite
- {RoleAdminChanged} not being emitted signaling this. *
- *Available since v3.1.* */ event RoleAdminChanged(bytes32 indexed role, bytes32 indexed previousAdminRole, bytes32 indexed newAdminRole);

  /**
- @dev Emitted when `account` is granted `role`. *
- `sender` is the account that originated the contract call, an admin role
- bearer except when using {_setupRole}. */ event RoleGranted(bytes32 indexed role, address indexed account, address indexed sender);

  /**
- @dev Emitted when `account` is revoked `role`. *
- `sender` is the account that originated the contract call:
- - if using `revokeRole`, it is the admin role bearer
- - if using `renounceRole`, it is the role bearer (i.e. `account`) */ event RoleRevoked(bytes32 indexed role, address indexed account, address indexed sender);

  /**
- @dev Returns `true` if `account` has been granted `role`. */ function hasRole(bytes32 role, address account) public view returns (bool) { return _roles[role].members.contains(account); }

  /**
- @dev Returns the number of accounts that have `role`. Can be used
- together with {getRoleMember} to enumerate all bearers of a role. */ function getRoleMemberCount(bytes32 role) public view returns (uint256) { return _roles[role].members.length(); }

  /**
- @dev Returns one of the accounts that have `role`. `index` must be a
- value between 0 and {getRoleMemberCount}, non-inclusive. *
- Role bearers are not sorted in any particular way, and their ordering may
- change at any point. *
- WARNING: When using {getRoleMember} and {getRoleMemberCount}, make sure
- you perform all queries on the same block. See the following
- https://forum.openzeppelin.com/t/iterating-over-elements-on-enumerableset-in-openzeppelin-contracts/2296 [forum post]
- for more information. */ function getRoleMember(bytes32 role, uint256 index) public view returns (address) { return _roles[role].members.at(index); }

  /**

- @dev Returns the admin role that controls `role`. See {grantRole} and
- {revokeRole}. *
- To change a role's admin, use {_setRoleAdmin}. */ function getRoleAdmin(bytes32 role) public view returns (bytes32) { return _roles[role].adminRole; }

  /**
- @dev Grants `role` to `account`. *
- If `account` had not been already granted `role`, emits a {RoleGranted}
- event. *
- Requirements: *
- - the caller must have `role`'s admin role. */ function grantRole(bytes32 role, address account) public virtual { require(hasRole(_roles[role].adminRole, _msgSender()), "AccessControl: sender must be an admin to grant");

    _grantRole(role, account); }

  /**
- @dev Revokes `role` from `account`. *
- If `account` had been granted `role`, emits a {RoleRevoked} event. *
- Requirements: *
- - the caller must have `role`'s admin role. */ function revokeRole(bytes32 role, address account) public virtual { require(hasRole(_roles[role].adminRole, _msgSender()), "AccessControl: sender must be an admin to revoke");

    _revokeRole(role, account); }

  /**
- @dev Revokes `role` from the calling account. *
- Roles are often managed via {grantRole} and {revokeRole}: this function's
- purpose is to provide a mechanism for accounts to lose their privileges
- if they are compromised (such as when a trusted device is misplaced). *
- If the calling account had been granted `role`, emits a {RoleRevoked}
- event. *
- Requirements: *
- - the caller must be `account`. */ function renounceRole(bytes32 role, address account) public virtual { require(account == _msgSender(), "AccessControl: can only renounce roles for self");

    _revokeRole(role, account); }

  /**

- @dev Grants `role` to `account`. *
- If `account` had not been already granted `role`, emits a {RoleGranted}
- event. Note that unlike {grantRole}, this function doesn't perform any
- checks on the calling account. *
- [WARNING]
- ====
- This function should only be called from the constructor when setting
- up the initial roles for the system. *
- Using this function in any other way is effectively circumventing the admin
- system imposed by {AccessControl}.
- ==== */ function _setupRole(bytes32 role, address account) internal virtual { _grantRole(role, account); }

  /**
- @dev Sets `adminRole` as `role`'s admin role. *
- Emits a {RoleAdminChanged} event. */ function _setRoleAdmin(bytes32 role, bytes32 adminRole) internal virtual { emit RoleAdminChanged(role, _roles[role].adminRole, adminRole); _roles[role].adminRole = adminRole; }

  function *grantRole(bytes32 role, address account) private { if (*roles[role].members.add(account)) {

  ```
  emit RoleGranted(role, account, _msgSender());
  ```

  } }

  function *revokeRole(bytes32 role, address account) private { if (*roles[role].members.remove(account)) {

  ```
  emit RoleRevoked(role, account, _msgSender());
  ```

  } } uint256[49] private __gap; } `OwnableUpgradeable.sol` javascript // SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

import "../utils/ContextUpgradeable.sol"; import "../proxy/Initializable.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 OwnableUpgradeable is Initializable, ContextUpgradeable { address private _owner;

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

  /**

  - @dev Initializes the contract setting the deployer as the initial owner. */ function _*Ownable*init() internal initializer { _*Context*init_unchained(); _*Ownable*init_unchained(); }

    function _*Ownable*init_unchained() internal initializer { address msgSender = _msgSender(); _owner = msgSender; emit OwnershipTransferred(address(0), msgSender); }

    /**

  - @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. */ 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; } uint256[49] private* _gap; } `ContextUpgradeable.sol` javascript // SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0; import "../proxy/Initializable.sol";

/*

- @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 ContextUpgradeable is Initializable { function _*Context*init() internal initializer {

```
__Context_init_unchained();
```

}

function _*Context*init_unchained() internal initializer { } 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.com/e
return msg.data;
```

} uint256[50] private __gap; } `ISTKBrightToken.sol` javascript // SPDX-License-Identifier: MIT

pragma solidity ^0.7.4;

import "@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol";

interface ISTKBrightToken is IERC20Upgradeable { function mint(address account, uint256 amount) external;

```
function burn(address account, uint256 amount) external;
```

}

```
Proxy.sol
```javascript
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev This abstract contract provides a fallback function that delegates all calls to another contr
 * instruction `delegatecall`. We refer to the second contract as the _implementation_ behind the pro
 * be specified by overriding the virtual {_implementation} function.
 *
 * Additionally, delegation to the implementation can be triggered manually through the {_fallback} f
 * different contract through the {_delegate} function.
 *
 * The success and return data of the delegated call will be returned back to the caller of the proxy
 */
abstract contract Proxy {
    /**
     * @dev Delegates the current call to `implementation`.
     *
     * This function does not return to its internall call site, it will return directly to the exter
     */
    function _delegate(address implementation) internal virtual {
```
```

```
        // solhint-disable-next-line no-inline-assembly
        assembly {
            // Copy msg.data. We take full control of memory in this inline assembly
            // block because it will not return to Solidity code. We overwrite the
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize())

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
            let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)

            // Copy the returned data.
            returndatacopy(0, 0, returndatasize())

            switch result
            // delegatecall returns 0 on error.
            case 0 { revert(0, returndatasize()) }
            default { return(0, returndatasize()) }
        }
    }

    /**
     * @dev This is a virtual function that should be overriden so it returns the address to which th
     * and {_fallback} should delegate.
     */
    function _implementation() internal view virtual returns (address);

    /**
     * @dev Delegates the current call to the address returned by `_implementation()`.
     *
     * This function does not return to its internall call site, it will return directly to the exter
     */
    function _fallback() internal virtual {
        _beforeFallback();
        _delegate(_implementation());
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_implementation()`. Wi
     * function in the contract matches the call data.
     */
    fallback () external payable virtual {
        _fallback();
    }

    /**
     * @dev Fallback function that delegates calls to the address returned by `_implementation()`. Wi
     * is empty.
     */
    receive () external payable virtual {
        _fallback();
    }

    /**
     * @dev Hook that is called before falling back to the implementation. Can happen as part of a ma
     * call, or as part of the Solidity `fallback` or `receive` functions.
     *
     * If overriden should call `super._beforeFallback()`.
     */
    function _beforeFallback() internal virtual {
    }
}
```



ContractsRegistry.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.4;
pragma experimental ABIEncoderV2;

import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import "@openzeppelin/contracts/proxy/TransparentUpgradeableProxy.sol";

import "./interfaces/IContractsRegistry.sol";

contract ContractsRegistry is IContractsRegistry, AccessControlUpgradeable {
    mapping (bytes32 => address) private _contracts;
    mapping (address => bool) private _isProxy;

    bytes32 constant public REGISTRY_ADMIN_ROLE = keccak256("REGISTRY_ADMIN_ROLE");

    bytes32 public constant UNISWAP_BRIGHT_TO_ETH_PAIR_NAME = keccak256("UNI_BRIGHT_ETH_PAIR");
    bytes32 public constant LIQUIDITY_MINING_STAKING_NAME = keccak256("LIQ_MINING_STAKING");

    bytes32 constant public BRIGHT_STAKING_NAME = keccak256("BRIGHT_STAKING_NAME");
    bytes32 constant public BRIGHT_NAME = keccak256("BRIGHT");
    bytes32 constant public STKBRIGHT_NAME = keccak256("STK_BRIGHT");

    modifier onlyAdmin() {
        require(hasRole(REGISTRY_ADMIN_ROLE, msg.sender), "ContractsRegistry: Caller is not an admin"
        _;
    }

    function __ContractsRegistry_init() external initializer {
        __AccessControl_init();

        _setupRole(REGISTRY_ADMIN_ROLE, msg.sender);
        _setRoleAdmin(REGISTRY_ADMIN_ROLE, REGISTRY_ADMIN_ROLE);
    }

    function getUniswapBrightToETHPairContract() external view override returns (address) {
        return getContract(UNISWAP_BRIGHT_TO_ETH_PAIR_NAME);
    }

    function getBrightContract() external view override returns (address) {
        return getContract(BRIGHT_NAME);
    }

    function getBrightStakingContract() external view override returns (address) {
        return getContract(BRIGHT_STAKING_NAME);
    }

    function getSTKBrightContract() external view override returns (address) {
        return getContract(STKBRIGHT_NAME);
    }

    function getLiquidityMiningStakingContract() external view override returns (address) {
        return getContract(LIQUIDITY_MINING_STAKING_NAME);
    }

    function getContract(bytes32 name) public view returns (address) {
        require(_contracts[name] != address(0), "ContractsRegistry: This mapping doesn't exist");

        return _contracts[name];
    }

    function upgradeContract(bytes32 name, address newImplementation) external onlyAdmin {
        require(_contracts[name] != address(0), "ContractsRegistry: This mapping doesn't exist");

        TransparentUpgradeableProxy proxy = TransparentUpgradeableProxy(payable(_contracts[name]));
```

```
        require(_isProxy[address(proxy)], "ContractsRegistry: Can't upgrade not a proxy contract");

        proxy.upgradeTo(newImplementation);
    }

    function addContract(bytes32 name, address contractAddress) external onlyAdmin {
        require(contractAddress != address(0), "ContractsRegistry: Null address is forbidden");

        _contracts[name] = contractAddress;
    }

    function addProxyContract(bytes32 name, address contractAddress) external onlyAdmin {
        require(contractAddress != address(0), "ContractsRegistry: Null address is forbidden");

        TransparentUpgradeableProxy proxy = new TransparentUpgradeableProxy(
            contractAddress, address(this), ""
        );

        _contracts[name] = address(proxy);
        _isProxy[address(proxy)] = true;
    }

    function deleteContract(bytes32 name) external onlyAdmin {
        require(_contracts[name] != address(0), "ContractsRegistry: This mapping doesn't exist");

        delete _isProxy[_contracts[name]];
        delete _contracts[name];
    }
}
```

Globals.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.4;
pragma experimental ABIEncoderV2;

uint256 constant SECONDS_IN_THE_YEAR = 365 * 24 * 60 * 60; // 365 days * 24 hours * 60 minutes * 60 s
uint256 constant MAX_INT = 2**256 - 1;

uint256 constant DECIMALS18 = 10**18;

uint256 constant PRECISION = 10**25;
uint256 constant PERCENTAGE_100 = 100 * PRECISION;

uint256 constant BLOCKS_PER_DAY = 6450;
uint256 constant BLOCKS_PER_YEAR = BLOCKS_PER_DAY * 365;

uint256 constant APY_TOKENS = DECIMALS18;
```

SafeMathUpgradeable.sol

```
// 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 SafeMathUpgradeable {
    /**
     * @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);
    }

    /**
     * @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).
     *
     * 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).
     *
     * 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;
    }
}
```

Math.sol

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

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Standard math utilities missing in the Solidity language.
 */
library Math {
    /**
     * @dev Returns the largest of two numbers.
     */
    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }

    /**
     * @dev Returns the smallest of two numbers.
     */
    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a < b ? a : b;
    }

    /**
     * @dev Returns the average of two numbers. The result is rounded towards
     * zero.
     */
    function average(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b) / 2 can overflow, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}
```

LiquidityMiningStaking.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.4;

import "@openzeppelin/contracts/math/Math.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts-upgradeable/token/ERC20/SafeERC20Upgradeable.sol";
import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

import "@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol";

import "./interfaces/IContractsRegistry.sol";
import "./interfaces/IBrightStaking.sol";
import "./interfaces/ILiquidityMiningStaking.sol";
import "./interfaces/token/IERC20Permit.sol";

import "./Globals.sol";
import "./AbstractCooldownStaking.sol";
```

```
contract LiquidityMiningStaking is
    ILiquidityMiningStaking,
    AbstractCooldownStaking,
    OwnableUpgradeable,
    ReentrancyGuard
{
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IERC20 public rewardsToken;
    address public stakingToken;
    IBrightStaking public brightStaking;

    uint256 public rewardPerBlock;
    uint256 public lastUpdateBlock;
    uint256 public rewardPerTokenStored;

    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) internal _rewards;

    uint256 public totalStaked;
    mapping(address => uint256) public staked;

    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);
    event RewardRestaked(address indexed user, uint256 reward);
    event RewardTokensRecovered(uint256 amount);

    modifier updateReward(address account) {
        _updateReward(account);
        _;
    }

    function __LiquidityMiningStaking_init(uint256 _rewardPerBlock, IContractsRegistry _contractsRegi
        __Ownable_init();
        rewardPerBlock = _rewardPerBlock;
        rewardsToken = IERC20(_contractsRegistry.getBrightContract());
        brightStaking = IBrightStaking(_contractsRegistry.getBrightStakingContract());
        stakingToken = _contractsRegistry.getUniswapBrightToETHPairContract();
    }

    function stake(uint256 _amount) external override nonReentrant updateReward(_msgSender()) {
        require(_amount > 0, "LMS: Amount should be greater than 0");

        IERC20(stakingToken).safeTransferFrom(_msgSender(), address(this), _amount);
        _stake(_msgSender(), _amount);
    }

    function stakeFor(address _user, uint256 _amount) external override {
        require(_amount > 0, "LMS: Amount should be greater than 0");

        IERC20(stakingToken).safeTransferFrom(_msgSender(), address(this), _amount);
        _stake(_user, _amount);
    }

    function stakeWithPermit(uint256 _stakingAmount, uint8 _v, bytes32 _r, bytes32 _s) external overr
        IERC20Permit(address(stakingToken)).permit(
            _msgSender(),
            address(this),
            _stakingAmount,
            MAX_INT,
            _v,
            _r,
            _s
```

```
        );

        IERC20(stakingToken).safeTransferFrom(_msgSender(), address(this), _stakingAmount);
        _stake(_msgSender(), _stakingAmount);
    }

    function _stake(address _user, uint256 _amount) internal {
        totalStaked = totalStaked.add(_amount);
        staked[_user] = staked[_user].add(_amount);

        emit Staked(_user, _amount);
    }

    function withdraw(uint256 _amount) public override nonReentrant updateReward(_msgSender()) {
        require(_amount > 0, "LMS: Amount should be greater than 0");
        uint256 userStaked = staked[_msgSender()];
        require(userStaked >= _amount, "LMS: Insufficient staked amount");

        totalStaked = totalStaked.sub(_amount);
        staked[_msgSender()] = userStaked.sub(_amount);
        IERC20(stakingToken).safeTransfer(_msgSender(), _amount);

        emit Withdrawn(_msgSender(), _amount);
    }


    /**
     * @dev Available after cooldown on callGetReward()
     */
    function exit() external override {
        withdraw(staked[_msgSender()]);
        getReward();
    }

    /**
     * @dev Available without cooldown
     */
    function restake() external override nonReentrant updateReward(_msgSender()) {
        uint256 _reward = _rewards[_msgSender()];

        if (_reward > 0) {
            delete _rewards[_msgSender()];
            rewardsToken.approve(address(brightStaking), _reward);
            brightStaking.stakeFor(_msgSender(), _reward);
            emit RewardRestaked(_msgSender(), _reward);
        }
    }

    /**
     * @dev Caller asks for rewards, which still will keep growing over the cooldown period
     */
    function callGetReward() external override nonReentrant updateReward(_msgSender()){
        require(_rewards[_msgSender()] > 0, "LMS: No rewards at stake");

        withdrawalsInfo[_msgSender()] = WithdrawalInfo(
            block.timestamp.add(WITHDRAWING_COOLDOWN_DURATION),
            0                                                   //not used
        );
    }

    function getReward() public override nonReentrant updateReward(_msgSender()) {
        uint256 _whenCanWithdrawBrightReward = whenCanWithdrawBrightReward(_msgSender());
        require(_whenCanWithdrawBrightReward != 0, "LMS: unlock not started/exp");
        require(_whenCanWithdrawBrightReward <= block.timestamp, "LMS: cooldown not reached");

        delete withdrawalsInfo[_msgSender()];
```

```
        uint256 _reward = _rewards[_msgSender()];

        if (_reward > 0) {
            delete _rewards[_msgSender()];
            rewardsToken.safeTransfer(_msgSender(), _reward);
            emit RewardPaid(_msgSender(), _reward);
        }
    }

    function setRewards(
        uint256 _rewardPerBlock
    ) external onlyOwner updateReward(address(0)) {
        rewardPerBlock = _rewardPerBlock;
    }

    function _updateReward(address account) internal {
        uint256 currentRewardPerToken = rewardPerToken();

        rewardPerTokenStored = currentRewardPerToken;
        lastUpdateBlock = block.number;

        if (account != address(0)) {
            _rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = currentRewardPerToken;
        }
    }

    function recoverRewards() external onlyOwner {
        uint256 _remaining = rewardsToken.balanceOf(address(this));
        rewardsToken.safeTransfer(owner(), _remaining);
        emit RewardTokensRecovered(_remaining);
    }

    /// @dev returns APY with 10**5 precision
    function getAPY() external view override returns (uint256) {
        uint256 totalSupply = IUniswapV2Pair(stakingToken).totalSupply();
        (uint256 reserveBright, , ) = IUniswapV2Pair(stakingToken).getReserves();

        if (totalSupply == 0 || reserveBright == 0) {
            return 0;
        }

        return rewardPerBlock.mul(BLOCKS_PER_YEAR).mul(PERCENTAGE_100).div(
            totalStaked.add(APY_TOKENS).mul(reserveBright.mul(2).mul(10**20).div(totalSupply))
        );
    }

    function rewardPerToken() public view override returns (uint256) {
        uint256 totalPoolStaked = totalStaked;

        if (totalPoolStaked == 0) {
            return rewardPerTokenStored;
        }
        uint256 _blocksPassed = lastUpdateBlock == 0 ? 0 : block.number.sub(lastUpdateBlock);
        uint256 accumulatedReward = _blocksPassed.mul(rewardPerBlock).mul(DECIMALS18).div(totalPoolSt

        return rewardPerTokenStored.add(accumulatedReward);
    }

    function earned(address _account) public view override returns (uint256) {
        uint256 rewardsDifference = rewardPerToken().sub(userRewardPerTokenPaid[_account]);
        uint256 newlyAccumulated = staked[_account].mul(rewardsDifference).div(DECIMALS18);

        return _rewards[_account].add(newlyAccumulated);
    }
```

```
}
```

EnumerableSetUpgradeable.sol

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

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
 * types.
 *
 * Sets have the following properties:
 *
 * - Elements are added, removed, and checked for existence in constant time
 * (O(1)).
 * - Elements are enumerated in O(n). No guarantees are made on the ordering.
 *
 *
```

- contract Example {
- // Add the library methods
- using EnumerableSet for EnumerableSet.AddressSet; *
- // Declare a set state variable
- EnumerableSet.AddressSet private mySet;
- }
- ` *
- As of v3.3.0, sets of type `bytes32` ( `Bytes32Set` ), `address` ( `AddressSet` )
- and `uint256` ( `UintSet` ) are supported. */ library EnumerableSetUpgradeable { // To implement this library for multiple types with as little code // repetition as possible, we write it in terms of a generic Set type with // bytes32 values. // The Set implementation uses private functions, and user-facing // implementations (such as AddressSet) are just wrappers around the // underlying Set. // This means that we can only create new EnumerableSets for types that fit // in bytes32.

  struct Set {

  ```
  // Storage of set values
  bytes32[] _values;

  // Position of the value in the `values` array, plus 1 because index 0
  // means a value is not in the set.
  mapping (bytes32 => uint256) _indexes;
  ```

  }

  /**

  - @dev Add a value to a set. O(1). *

- Returns true if the value was added to the set, that is if it was not
- already present. */ function *add(Set storage set, bytes32 value) private returns (bool) { if (!*contains(set, value)) {

```
set._values.push(value);
// The value is stored at length-1, but we add 1 to all indexes
// and use 0 as a sentinel value
set._indexes[value] = set._values.length;
return true;
```

} else {

```
return false;
```

} }

/**

- @dev Removes a value from a set. O(1). *
- Returns true if the value was removed from the set, that is if it was
- present. */ function *remove(Set storage set, bytes32 value) private returns (bool) { // We read and store the value's index to prevent multiple reads from the same storage slot uint256 valueIndex = set.*indexes[value];

if (valueIndex != 0) { // Equivalent to contains(set, value)

```
// To delete an element from the _values array in O(1), we swap the element to delete with t
// the array, and then remove the last element (sometimes called as 'swap and pop').
// This modifies the order of the array, as noted in {at}.

uint256 toDeleteIndex = valueIndex - 1;
uint256 lastIndex = set._values.length - 1;

// When the value to delete is the last one, the swap operation is unnecessary. However, sin
// so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' statement.

bytes32 lastvalue = set._values[lastIndex];

// Move the last value to the index where the value to delete is
set._values[toDeleteIndex] = lastvalue;
// Update the index for the moved value
set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

// Delete the slot where the moved value was stored
set._values.pop();

// Delete the index for the deleted slot
delete set._indexes[value];

return true;
```

} else {

```
return false;
```

} }

/**

- @dev Returns true if the value is in the set. O(1). */ function *contains(Set storage set, bytes32 value) private view returns (bool) { return set.*indexes[value] != 0; }

/**

- @dev Returns the number of values on the set. O(1). */ function *length(Set storage set) private view returns (uint256) { return set.*values.length; }

/**

- @dev Returns the value stored at position `index` in the set. O(1). *
- Note that there are no guarantees on the ordering of values inside the
- array, and it may change when more values are added or removed. *
- Requirements: *
- - `index` must be strictly less than {length}. */ function *at(Set storage set, uint256 index) private view returns (bytes32) { require(set.*values.length > index, "EnumerableSet: index out of bounds"); return set._values[index]; }

// Bytes32Set

struct Bytes32Set { Set _inner; }

/**

- @dev Add a value to a set. O(1). *
- Returns true if the value was added to the set, that is if it was not
- already present. */ function add(Bytes32Set storage set, bytes32 value) internal returns (bool) { return *add(set.*inner, value); }

/**

- @dev Removes a value from a set. O(1). *
- Returns true if the value was removed from the set, that is if it was
- present. */ function remove(Bytes32Set storage set, bytes32 value) internal returns (bool) { return *remove(set.*inner, value); }

/**

- @dev Returns true if the value is in the set. O(1). */ function contains(Bytes32Set storage set, bytes32 value) internal view returns (bool) { return *contains(set.*inner, value); }

/**

- @dev Returns the number of values in the set. O(1). */ function length(Bytes32Set storage set) internal view returns (uint256) { return *length(set.*inner); }

/**

- @dev Returns the value stored at position `index` in the set. O(1). *

- Note that there are no guarantees on the ordering of values inside the
- array, and it may change when more values are added or removed. *
- Requirements: *
- `index` must be strictly less than {length}. */ function at(Bytes32Set storage set, uint256 index) internal view returns (bytes32) { return *at(set.*inner, index); }

// AddressSet

struct AddressSet { Set _inner; }

/**

- @dev Add a value to a set. O(1). *
- Returns true if the value was added to the set, that is if it was not
- already present. */ function add(AddressSet storage set, address value) internal returns (bool) { return *add(set.*inner, bytes32(uint256(uint160(value)))); }

/**

- @dev Removes a value from a set. O(1). *
- Returns true if the value was removed from the set, that is if it was
- present. */ function remove(AddressSet storage set, address value) internal returns (bool) { return *remove(set.*inner, bytes32(uint256(uint160(value)))); }

/**

- @dev Returns true if the value is in the set. O(1). */ function contains(AddressSet storage set, address value) internal view returns (bool) { return *contains(set.*inner, bytes32(uint256(uint160(value)))); }

/**

- @dev Returns the number of values in the set. O(1). */ function length(AddressSet storage set) internal view returns (uint256) { return *length(set.*inner); }

/**

- @dev Returns the value stored at position `index` in the set. O(1). *
- Note that there are no guarantees on the ordering of values inside the
- array, and it may change when more values are added or removed. *
- Requirements: *
- `index` must be strictly less than {length}. */ function at(AddressSet storage set, uint256 index) internal view returns (address) { return address(uint160(uint256(*at(set.*inner, index)))); }

```
// UintSet

struct UintSet {
    Set _inner;
}

/**
 * @dev Add a value to a set. O(1).
 *
 * Returns true if the value was added to the set, that is if it was not
 * already present.
 */
```

```
    function add(UintSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(UintSet storage set, uint256 value) internal view returns (bool) {
        return _contains(set._inner, bytes32(value));
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }
```

/**

```
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
        return uint256(_at(set._inner, index));
    }
```

}

IUniswapV2Pair.sol

```
```javascript
pragma solidity >=0.5.0;

interface IUniswapV2Pair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);
```
```

```
    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

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

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

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTim
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

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

SafeERC20Upgradeable.sol

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

pragma solidity >=0.6.0 <0.8.0;

import "./IERC20Upgradeable.sol";
import "../../math/SafeMathUpgradeable.sol";
import "../../utils/AddressUpgradeable.sol";

/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20Upgradeable {
    using SafeMathUpgradeable for uint256;
    using AddressUpgradeable for address;

    function safeTransfer(IERC20Upgradeable token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }
```

```
    function safeTransferFrom(IERC20Upgradeable token, address from, address to, uint256 value) inter
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, valu
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20Upgradeable token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    function safeDecreaseAllowance(IERC20Upgradeable token, address spender, uint256 value) internal
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20Upgradeable token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
        // the target address contains contract code and also asserts for success in the low-level ca

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

AddressUpgradeable.sol

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

pragma solidity >=0.6.2 <0.8.0;

/**
 * @dev Collection of functions related to the address type
 */
library AddressUpgradeable {
    /**
```

```
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

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

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
```

```
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data, string memory errorMessage) internal ret
        return functionCallWithValue(target, data, 0, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValu
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory er
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: value }(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memor
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data, string memory errorMessage) intern
        require(isContract(target), "Address: static call to non-contract");
```

```
        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.staticcall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _verifyCallResult(bool success, bytes memory returndata, string memory errorMessage) pri
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}
```

IERC20Upgradeable.sol

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

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20Upgradeable {
    /**
     * @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.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
```



ILiquidityMiningStaking.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.4;

import "./IAbstractCooldownStaking.sol";

interface ILiquidityMiningStaking is IAbstractCooldownStaking{

    function rewardPerToken() external view returns (uint256);

    function earned(address _account) external view returns (uint256);

    function stakeFor(address _user, uint256 _amount) external;

    function stakeWithPermit(uint256 _stakingAmount, uint8 _v, bytes32 _r, bytes32 _s) external;

    function stake(uint256 _amount) external;

    function withdraw(uint256 _amount) external;
```

```
    function callGetReward() external;

    function getReward() external;

    function restake() external;

    function exit() external;

    function getAPY() external view returns (uint256);
}
```

IAbstractCooldownStaking.sol

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

pragma solidity ^0.7.4;

import "./token/ISTKBrightToken.sol";

interface IAbstractCooldownStaking {

    function getWithdrawalInfo(address _userAddr) external view
    returns (
        uint256 _amount,
        uint256 _unlockPeriod,
        uint256 _availableFor
    );

}
```

AbstractCooldownStaking.sol

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

pragma solidity ^0.7.4;

import "@openzeppelin/contracts/math/SafeMath.sol";
import "./interfaces/IAbstractCooldownStaking.sol";

abstract contract AbstractCooldownStaking is IAbstractCooldownStaking{
    using SafeMath for uint256;

    uint256 internal constant WITHDRAWING_COOLDOWN_DURATION = 7 days;
    uint256 internal constant WITHDRAWAL_PHASE_DURATION = 2 days;

    struct WithdrawalInfo {
        uint256 coolDownTimeEnd;
        uint256 amount;             //optional
    }

    mapping(address => WithdrawalInfo) internal withdrawalsInfo;

    // There is a second withdrawal phase of 48 hours to actually receive the rewards.
    // If a user misses this period, in order to withdraw he has to wait for 7 days again.
    // It will return:
    // 0 if cooldown time didn't start or if phase duration (48hs) has expired
    // #coolDownTimeEnd Time when user can withdraw.
    function whenCanWithdrawBrightReward(address _address) internal view returns (uint256) {
        return
        withdrawalsInfo[_address].coolDownTimeEnd.add(WITHDRAWAL_PHASE_DURATION) >=
        block.timestamp
        ? withdrawalsInfo[_address].coolDownTimeEnd
        : 0;
```

```
    }

    function getWithdrawalInfo(address _userAddr) external view override
    returns (
        uint256 _amount,
        uint256 _unlockPeriod,
        uint256 _availableFor
    )
    {
        _unlockPeriod = whenCanWithdrawBrightReward(_userAddr);
        if (_unlockPeriod > 0) {
            _amount = withdrawalsInfo[_userAddr].amount;

            uint256 endUnlockPeriod = _unlockPeriod.add(WITHDRAWAL_PHASE_DURATION);
            _availableFor = _unlockPeriod <= block.timestamp ? endUnlockPeriod : 0;
        }
    }

}
```

# Analysis of audit results

## Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.



## Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

## Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

## Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

## Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

## Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many

ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

## External Contract Referencing

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

```
PASSED!
```

- **Security suggestion:**
  no.

## Unsolved TODO comments

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

```
PASSED!
```

- **Security suggestion:**
  no.



## Short Address/Parameter Attack

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

```
PASSED!
```

- **Security suggestion:**
  no.

## Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

## Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

## Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

## Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

## Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

## Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.



## Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

## tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

## 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