```solidity
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (proxy/transparent/TransparentUpgradeableProxy.sol)

pragma solidity ^0.8.0;

import "../ERC1967/ERC1967Proxy.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[proxy selector
 * 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 pattern implies two
 * things that go hand in hand:
 *
 * 1. If any account other than the admin calls the proxy, the call will be forwarded to the implementation, even if
 * 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 forwarded to the
 * implementation. If the admin tries to call a function on the implementation it will fail with an error that says
 * "admin cannot fallback to proxy target".
 *
 * These properties mean that the admin account can only be used for admin actions like upgrading the proxy or changing
 * the admin, so it's best if it's a dedicated account that is not used for anything else. This will avoid headaches due
 * 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 set up this way,
 * you should think of the `ProxyAdmin` instance as the real administrative interface of your proxy.
 */
contract TransparentUpgradeableProxy is ERC1967Proxy {
    /**
     * @dev Initializes an upgradeable proxy managed by `_admin`, backed by the implementation at `_logic`, and
     * optionally initialized with `_data` as explained in {ERC1967Proxy-constructor}.
     */
    constructor(address _logic, address admin_, bytes memory _data) payable ERC1967Proxy(_logic, _data) {
        _changeAdmin(admin_);
    }

    /**
     * @dev Modifier used internally that will delegate the call to the implementation unless the sender is the admin.
     */
    modifier ifAdmin() {
        if (msg.sender == _getAdmin()) {
            _;
        } 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 by EIP1967) using the
     * https://eth.wiki/json-rpc/API#eth_getstorageat[`eth_getStorageAt`] RPC call.
     * `0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103`
     */
    function admin() external ifAdmin returns (address admin_) {
        admin_ = _getAdmin();
    }

    /**
     * @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 by EIP1967) using the
     * 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 {
        _changeAdmin(newAdmin);
    }

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

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

    /**
     * @dev Returns the current admin.
     */
    function _admin() internal view virtual returns (address) {
        return _getAdmin();
    }

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