```solidity
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.5.0) (proxy/ERC1967/ERC1967Upgrade.sol)

pragma solidity ^0.8.2;

import "../beacon/IBeacon.sol";
import "../../interfaces/draft-IERC1822.sol";
import "../../utils/Address.sol";
import "../../utils/StorageSlot.sol";

/**
 * @dev This abstract contract provides getters and event emitting update functions for
 * https://eips.ethereum.org/EIPS/eip-1967[EIP1967] slots.
 *
 * _Available since v4.1._
 *
 * @custom:oz-upgrades-unsafe-allow delegatecall
 */
abstract contract ERC1967Upgrade {
    // This is the keccak-256 hash of "eip1967.proxy.rollback" subtracted by 1
    bytes32 private constant _ROLLBACK_SLOT = 0x4910fdfa16fed3260ed0e7147f7cc6da11a60208b5b9406d12a635614ffd9143;

    /**
     * @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 internal constant _IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc;

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

    /**
     * @dev Returns the current implementation address.
     */
    function _getImplementation() internal view returns (address) {
        return StorageSlot.getAddressSlot(_IMPLEMENTATION_SLOT).value;
    }

    /**
     * @dev Stores a new address in the EIP1967 implementation slot.
     */
    function _setImplementation(address newImplementation) private {
        require(Address.isContract(newImplementation), "ERC1967: new implementation is not a contract");
        StorageSlot.getAddressSlot(_IMPLEMENTATION_SLOT).value = newImplementation;
    }

    /**
     * @dev Perform implementation upgrade
     *
     * Emits an {Upgraded} event.
     */
    function _upgradeTo(address newImplementation) internal {
        _setImplementation(newImplementation);
        emit Upgraded(newImplementation);
    }

    /**
     * @dev Perform implementation upgrade with additional setup call.
     *
     * Emits an {Upgraded} event.
     */
    function _upgradeToAndCall(
        address newImplementation,
        bytes memory data,
        bool forceCall
    ) internal {
        _upgradeTo(newImplementation);
        if (data.length > 0 || forceCall) {
            Address.functionDelegateCall(newImplementation, data);
        }
    }

    /**
     * @dev Perform implementation upgrade with security checks for UUPS proxies, and additional setup call.
     *
     * Emits an {Upgraded} event.
     */
    function _upgradeToAndCallUUPS(
        address newImplementation,
        bytes memory data,
        bool forceCall
    ) internal {
        // Upgrades from old implementations will perform a rollback test. This test requires the new
        // implementation to upgrade back to the old, non-ERC1822 compliant, implementation. Removing
        // this special case will break upgrade paths from old UUPS implementation to new ones.
        if (StorageSlot.getBooleanSlot(_ROLLBACK_SLOT).value) {
            _setImplementation(newImplementation);
        } else {
            try IERC1822Proxiable(newImplementation).proxiableUUID() returns (bytes32 slot) {
                require(slot == _IMPLEMENTATION_SLOT, "ERC1967Upgrade: unsupported proxiableUUID");
            } catch {
                revert("ERC1967Upgrade: new implementation is not UUPS");
            }
            _upgradeToAndCall(newImplementation, data, forceCall);
        }
    }

    /**
     * @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 internal constant _ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103;

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

    /**
     * @dev Returns the current admin.
     */
    function _getAdmin() internal view returns (address) {
        return StorageSlot.getAddressSlot(_ADMIN_SLOT).value;
    }

    /**
     * @dev Stores a new address in the EIP1967 admin slot.
     */
    function _setAdmin(address newAdmin) private {
        require(newAdmin != address(0), "ERC1967: new admin is the zero address");
        StorageSlot.getAddressSlot(_ADMIN_SLOT).value = newAdmin;
    }

    /**
     * @dev Changes the admin of the proxy.
     *
     * Emits an {AdminChanged} event.
     */
    function _changeAdmin(address newAdmin) internal {
        emit AdminChanged(_getAdmin(), newAdmin);
```

```solidity
        _setAdmin(newAdmin);
    }

    /**
     * @dev The storage slot of the UpgradeableBeacon contract which defines the implementation for this proxy.
     * This is bytes32(uint256(keccak256('eip1967.proxy.beacon')) - 1)) and is validated in the constructor.
     */
    bytes32 internal constant _BEACON_SLOT = 0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50;

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

    /**
     * @dev Returns the current beacon.
     */
    function _getBeacon() internal view returns (address) {
        return StorageSlot.getAddressSlot(_BEACON_SLOT).value;
    }

    /**
     * @dev Stores a new beacon in the EIP1967 beacon slot.
     */
    function _setBeacon(address newBeacon) private {
        require(Address.isContract(newBeacon), "ERC1967: new beacon is not a contract");
        require(
            Address.isContract(IBeacon(newBeacon).implementation()),
            "ERC1967: beacon implementation is not a contract"
        );
        StorageSlot.getAddressSlot(_BEACON_SLOT).value = newBeacon;
    }

    /**
     * @dev Perform beacon upgrade with additional setup call. Note: This upgrades the address of the beacon, it does
     * not upgrade the implementation contained in the beacon (see {UpgradeableBeacon-_setImplementation} for that).
     *
     * Emits a {BeaconUpgraded} event.
     */
    function _upgradeBeaconToAndCall(
        address newBeacon,
        bytes memory data,
        bool forceCall
    ) internal {
        _setBeacon(newBeacon);
        emit BeaconUpgraded(newBeacon);
        if (data.length > 0 || forceCall) {
            Address.functionDelegateCall(IBeacon(newBeacon).implementation(), data);
        }
    }
}
```