Brief audit of CedarVault.sol and DLPS.sol source code.

Summary in brief

Question	Answer
Withdraw USDC/WETH (principal deposit) without burning DLPS?	No — there is no such path for users
Withdraw accrued trading fees without burning DLPS?	Yes — this is by design (getUserFeeUSDC, getUserFeeETH)
Move the active Uniswap position to another address?	No — the position NFT belongs to the vault; transfer is not provided
Can the owner take assets bypassing DLPS?	Only 20% of trading fees, not the full principal

1. How USDC/WETH leave the vault

Users (without owner)

The only principal (deposit) withdrawal path: burningUserDLPS — DLPS must be burned.

```
151:194:CedarVault/contracts/contracts/CedarVault.sol
    function burningUserDLPS(uint256 amount) external nonReentrant onlyEoa {
        require(burningAllActive, "stopBurningAllDLPS");
        // ...
        if (_hasActiveUniswapPosition()) {
            revert("uniswapPositionIsActive");
        }
        // ...
        dlps.burnFrom(msg.sender, amount);
        // ...
        IERC20Minimal(weth).transfer(msg.sender, wethOut);
        IERC20Minimal(usdcToken).transfer(msg.sender, usdcOut);
    }
```

Requirements:

- burning is enabled (burningAllActive)
- no active Uniswap position (otherwise burn is blocked)
- no active staking lock (timeBurningMyDLPS)
- payout is a share from _usableWethBalance() / _usableUsdcBalance() — amounts not reserved for fees

Fee withdrawal (not deposit):

```
210:232:CedarVault/contracts/contracts/CedarVault.sol
    function getUserFeeUSDC() external nonReentrant onlyEoa { ... }
    function getUserFeeETH() external nonReentrant onlyEoa { ... }
```

This is only userFeeUsdc / userFeeEth — a share of Uniswap trading fees, not principal. DLPS are not burned — this is expected behavior.

Owner

There is no direct withdraw, rescue, or sweep.

The owner receives tokens only via _distributeFees — 20% of collected trading fees:

```
548:575:CedarVault/contracts/contracts/CedarVault.sol
        if (ownerEth > 0) {
            require(IERC20Minimal(weth).transfer(owner(), ownerEth), "owner WETH failed");
        }
        if (ownerUsdc > 0) {
```

```
            require(IERC20Minimal(usdcToken).transfer(owner(), ownerUsdc), "owner USDC failed");
        }
```

Triggered on collectFeesCedar and on removeLiquidityCedar (only on tokensOwed, not on position principal).

reinvestFeeEth() — clears user fee accounting and returns WETH to the vault liquidity pool; does not send principal to the owner.

2. Active Uniswap position

- The position NFT is always minted to address(this) (the vault):

```
342:358:CedarVault/contracts/contracts/CedarVault.sol
            (uint256 tokenId,,,) = INonfungiblePositionManager(nfpm).mint(
                // ...
                    recipient: address(this),
```

- There is no transfer / safeTransferFrom function for the NFT to another address.

The owner can only:

- addLiquidityCedar — deploy assets into the position
- collectFeesCedar — collect fees (20% → owner, 80% → DLPS holder accounting)
- removeLiquidityCedar — withdraw liquidity back to the vault, not to an arbitrary address

While the position is active (liquidity > 0):

- users cannot burn DLPS
- principal is effectively inside NFPM, on the vault balance as an NFT

Moving the position to another wallet via the contract is not possible.

3. DLPS — burn bypass protection

```
33:37:CedarVault/contracts/contracts/DLPS.sol
    function _update(address from, address to, uint256 value) internal override {
        if (from != address(0) && to != address(0) && to != vault) {
            revert NonTransferable();
        }
```

DLPS cannot be transferred to another address or sold — only the vault can mint and burn. Bypassing burn via DLPS transfer is not possible.

4. Separation of deposit vs. fees

```
630:637:CedarVault/contracts/contracts/CedarVault.sol
    function _usableUsdcBalance() internal view returns (uint256) {
        uint256 bal = IERC20Minimal(usdcToken).balanceOf(address(this));
        return bal > uniswapFeeUsdc ? bal - uniswapFeeUsdc : 0;
    }
    function _usableWethBalance() internal view returns (uint256) {
        uint256 bal = IERC20Minimal(weth).balanceOf(address(this));
        return bal > uniswapFeeEth ? bal - uniswapFeeEth : 0;
    }
```

- Burn uses only _usable* (principal)
- Claim fees uses the reserved uniswapFee*

There is no direct path to withdraw deposit via fee claims in the contract logic.

5. Centralization risks (not a bug, but important to know)

The owner cannot steal principal in a single transaction, but can restrict exits:

Owner action	Effect
isBurningAllDLPS(false)	users cannot exit via burn
isSendUSDC(false)	new deposits are disabled
isGetUserFeeETH(false)	WETH fee claims are disabled
reinvestFeeEth()	resets unclaimed user fees (not principal)
removeLiquidityCedar + keep burning disabled	principal sits on the vault, but exit is blocked

This is trust in the owner, not a hidden backdoor for withdrawing funds.

6. Other observations

- ReentrancyGuard on deposit/burn/claim/uniswap — good
- onlyEoa — blocks calls via other contracts (protection, but no smart wallet composability)
- No proxy/upgrade — logic cannot be replaced after deployment
- nfpm, pool, weth, swapRouter — immutable
- Direct USDC/WETH donations to the vault without minting DLPS increase NAV for DLPS holders, not the owner

Conclusion for investor/user

1. Withdrawing deposit (USDC/WETH principal) without burning DLPS — not possible (neither user nor owner via the contract).
2. Withdrawing trading fees without burn — possible via getUserFeeUSDC / getUserFeeETH.
3. Moving the Uniswap position to another address — not possible; vault-only management, funds return to the vault.
4. Owner receives only 20% of trading fees; full control over user exits — via admin flags (centralization risk).