Pashov Audit Group

Euler
Security Review

November 6th 2025 - November 8th 2025

Contents

1. About Pashov Audit Group 3
2. Disclaimer 3
3. Risk Classification 3
4. About Euler Vault Kit Periphery 4
5. Executive Summary 4
6. Findings 5
Medium findings 6
[M-01] Vault status check missing in `ERC4626EVCCollateralFreezable` withdraw 6
Low findings 8
[L-01] ERC4626 view functions ignore frozen status breaking compliancy 8
[L-02] Enforce freeze status of initiator in `deposit()` and `mint()` 8
[L-03] `maxDeposit()` and `maxMint()` return `uint256.max` ignoring cap 9
[L-04] Can not liquidate when account is frozen or contract is paused 9
[L-05] Transfer bypass through dummy controller 10

1. About Pashov Audit Group

Pashov Audit Group consists of 40+ freelance security researchers, who are well proven in the space - most have earned over $100k in public contest rewards, are multi-time champions or have truly excelled in audits with us. We only work with proven and motivated talent.

With over 300 security audits completed — uncovering and helping patch thousands of vulnerabilities — the group strives to create the absolute very best audit journey possible. While 100% security is never possible to guarantee, we do guarantee you our team's best efforts for your project.

Check out our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

• **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users
• **Medium** - leads to a moderate material loss of assets in the protocol or moderately harms a group of users
• **Low** - leads to a minor material loss of assets in the protocol or harms a small group of users

Likelihood

• **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost
• **Medium** - only a conditionally incentivized attack vector, but still relatively likely
• **Low** - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive

4. About Euler Vault Kit Periphery

Euler Vault Kit Periphery is a set of on-chain and off-chain components that support the Euler Vault Kit’s system of ERC-4626-based credit vaults, which function as passive lending pools with borrowing functionality.

5. Executive Summary

A time-boxed security review of the **euler-xyz/evk-periphery** and **euler-xyz/evk-periphery** repositories was done by Pashov Audit Group, during which **zark**, **Ch_301**, **unforgiven**, **Nyx** engaged to review **Euler Vault Kit Periphery**. A total of **6** issues were uncovered.

Protocol Summary	
Project Name	Euler Vault Kit Periphery
Protocol Type	Periphery contracts
Timeline	November 6th 2025 - November 8th 2025

Review commit hashes:
- 3d70bf28e13625d99bb3b90ab5b3d28ec2b7f264 (euler-xyz/evk-periphery)
- 0467fb77dcf269be6d572bcd6427a3626a3a4e14 (euler-xyz/evk-periphery)

Fixes review commit hash:
- 06c0817d1fe4dcd9f420c28085204024fdf8188d (euler-xyz/evk-periphery)

Scope

`ERC4626EVCCollateralSecuritize.sol` `ERC4626EVC.sol`
`ERC4626EVCCollateral.solERC4626EVCCollateralCapped.sol`
`ERC4626EVCCollateralFreezable.sol` `ERC4626EVCCollateralSecuritizeFactory.sol`
`IERC4626EVCCollateralSecuritizeFactory.sol` `SwapVerifier.sol`
`TransferFromSender.sol`

6. Findings

Findings count

Severity	Amount
Medium	1
Low	5
Total findings	**6**

Summary of findings

ID	Title	Severity	Status
[M-01]	Vault status check missing in `ERC4626EVCCollateralFreezable` withdraw	Medium	**Resolved**
[L-01]	ERC4626 view functions ignore frozen status breaking compliancy	Low	**Resolved**
[L-02]	Enforce freeze status of initiator in `deposit()` and `mint()`	Low	Acknowledged
[L-03]	`maxDeposit()` and `maxMint()` return `uint256.max` ignoring cap	Low	**Resolved**
[L-04]	Can not liquidate when account is frozen or contract is paused	Low	Acknowledged
[L-05]	Transfer bypass through dummy controller	Low	**Resolved**

Medium findings

[M-01] Vault status check missing in `ERC4626EVCCollateralFreezable` withdraw

Severity

Impact: Medium

Likelihood: Medium

Description

In the `ERC4626EVCCollateralFreezable` contract, the `withdraw` and `redeem` functions use the `takeSnapshot` modifier and call the parent functions from `ERC4626EVCCollateral`:

```
    function withdraw(uint256 assets, address receiver, address owner)
        public
        virtual
        override
        callThroughEVC
        nonReentrant
        whenNotPaused
        whenNotFrozen(owner)
        whenNotFrozen(receiver)
        takeSnapshot
        returns (uint256 shares)
    {
        shares = ERC4626EVCCollateral.withdraw(assets, receiver, owner);
    }
```

However, these functions do not call `evc.requireVaultStatusCheck()` afterward to reset snapshot. The `evc.requireVaultStatusCheck()` function triggers `checkVaultStatus()`, which: - Validates that the vault's total assets do not exceed the supply cap. - Disables the `SNAPSHOT` feature so future snapshots can be safely taken.

```
    /// @notice Checks the status of the vault and validates supply cap constraints.
    /// @return magicValue The selector of checkVaultStatus if the vault is in a valid state.
    function checkVaultStatus() external virtual onlyEVCWithChecksInProgress returns (bytes4
magicValue) {
        if (_isEnabled(SNAPSHOT)) {
            uint256 finalTotalAssets = totalAssets();

            if (finalTotalAssets > _supplyCap.resolve() && finalTotalAssets >
_snapshotTotalAssets) {
                revert SupplyCapExceeded();
            }

            // Disable the snapshot feature to allow future snapshots to be taken. There's no
need to clear the total
            // assets snapshot itself.
            _disableFeature(SNAPSHOT);
        }

        return IVault.checkVaultStatus.selector;
    }
```

Without calling `evc.requireVaultStatusCheck()`, the snapshot remains active, and the supply cap validation logic is skipped.
This can cause incorrect behavior in later deposits or withdrawals, including wrong total asset calculations, or potential bypass of supply cap constraint.

Note: `ERC4626EVCCollateralSecuritize.sol` also inherits the same contract and functions with the same issue.

Recommendations

Add a call to `evc.requireVaultStatusCheck()` at the end of both `withdraw` and `redeem` functions to ensure the snapshot is properly cleared and vault status stays consistent.

Low findings

[L-01] ERC4626 view functions ignore frozen status breaking compliancy

`ERC4626EVCCollateralCapped::{maxDeposit,maxMint,maxWithdraw,maxRedeem}` report limits without checking whether the caller/owner is frozen, even though every state-changing path is guarded by `whenNotFrozen`. As a result, integrators relying on ERC-4626 view helpers are told they can deposit or redeem positive amounts, while the subsequent transaction will revert due to the freeze, breaking ERC-4626 compliance expectations and causing confusion.

```
/// @notice Calculate the amount of assets that will be transferred when redeeming requested
amount of shares
/// @param shares Amount of shares redeemed
/// @return Amount of assets transferred
function previewRedeem(uint256 shares)
    public
    view
    virtual
    override
    nonReentrantView(this.previewRedeem.selector)
    returns (uint256)
{
    return super.previewRedeem(shares);
}
```

It is recommended to mirror the freeze logic in these view helpers (check `isFrozen(owner)` / `isFrozen(receiver)` and return zero when frozen) so the limits match what the stateful functions will actually allow.

[L-02] Enforce freeze status of initiator in `deposit()` and `mint()`

The `deposit()` and `mint()` functions in `ERC4626EVCCollateralFreezable.sol` fail to apply the `whenNotFrozen(_msgSender())` modifier to the transaction initiator. This will create a loophole where a frozen account, which should be completely immobilized, can still initiate deposits as long as the shares are sent to an unfrozen recipient, thus violating the security invariant of a total account lock.

While the specific `ERC4626EVCCollateralSecuritize.sol` implementation is coincidentally protected (because its `isCommonOwner` check forces the receiver to have the same owner). Any future vault that inherits from `ERC4626EVCCollateralFreezable` without this same restrictive logic will be immediately vulnerable.

It is recommended to add the `whenNotFrozen(_msgSender())` modifier to both functions in the base class to ensure the initiator's status is always checked.

Euler comments:

We acknowledge the finding but will keep as is. We will define freezing as preventing the vault balance of an account to change. This definition is compatible with Securitize requirements and doesn't include the sender of the deposit. Crucially we can avoid this way an issue of not being able to communicate the freeze through maxMint / maxDeposit functions, which only check the receiver.

We have added a comment with definition for the freeze function: euler-xyz/evk-periphery@13aa1e74

[L-03] `maxDeposit()` and `maxMint()` return `uint256.max` ignoring cap

The `ERC4626EVCCollateralCapped.sol` contract implements a supply cap `_supplyCap` to limit the total assets in the vault. However, it fails to override the standard EIP-4626 functions `maxDeposit()` and `maxMint()`. As a result, these functions inherit the default implementation from the base OpenZeppelin `ERC4626.sol` contract, which is to simply return `type(uint256).max`.

This creates a discrepancy: the vault enforces a supply cap on `deposit()` and `mint()` transactions, but the view functions designed to report this limit misleadingly signal that there is no cap.

This issue breaks the contract's compliance with the EIP-4626 standard.

[L-04] Can not liquidate when account is frozen or contract is paused

Right now, if an account is frozen or the contract is paused, you can’t liquidate it.
This happens because both `transfer()` and `transferFrom()` in `ERC4626EVCCollateralSecuritize` use `whenNotFrozen(from)`, which makes the call revert, even when it should be allowed (for example, when `evc.isControlCollateralInProgress() && isTransferCompliant(to, amount)` is true).

```
function transfer(address to, uint256 amount)
    public
    virtual
    override
    callThroughEVC
    nonReentrant
    whenNotPaused
    whenNotFrozen(_msgSender())
    whenNotFrozen(to)
    returns (bool result)
{
    if (
        !isCommonOwner(_msgSender(), to)
            && !(evc.isControlCollateralInProgress() && isTransferCompliant(to, amount))
    ) {
        revert NotAuthorized();
    }
    result = ERC4626EVCCollateral.transfer(to, amount);
}
```

This can cause problems since the system won't be able to liquidate the bad position, which could lead to bad debt.

Recommendation:

Update the `whenNotFrozen(from)` check inside `transfer()` and `transferFrom()` so it allows transfers when liquidation is happening.

Euler comments:

We acknowledge the finding but will keep as is. The definition of how the contract should behave during a freeze is arbitrary. While it may make sense for other contracts which will use the freeze functionality, for Securitize the requirement is to block liquidations as well. We prefer to keep as is and let future contracts override this behavior.

[L-05] Transfer bypass through dummy controller

The documentation states that transfers of shares should not be allowed. Though, `ERC4626EVCCollateralSecuritize::transfer` and `ERC4626EVCCollateralSecuritize::transferFrom` allow vault shares to move between different owners whenever `evc.isControlCollateralInProgress()` is true. That flag is set exclusively by `EVC.controlCollateral`, but registering a controller is under the account owner's control: `EVC.enableController` accepts any address provided by the owner/operator and only checks that it returns the expected magic value from `IVault.checkAccountStatus`. An account owner can therefore deploy a trivial "controller" contract, enable it, and call `controlCollateral` to simulate a liquidation. During that call the vault sees `isControlCollateralInProgress()` as true and `_msgSender()` resolves to the borrower account, so the transfer guard passes and the caller can route shares to any compliant*third party address. In this way, the same-owner transfer restriction can be bypassed.

```
// ERC4626EVCCollateralSecuritize.sol
if (!isCommonOwner(_msgSender(), to) && !(evc.isControlCollateralInProgress() &&
isTransferCompliant(to, amount))) {
    revert NotAuthorized();
}
```

Recommendations

Require additional checks in the vault so that liquidation-mode transfers are only honored when initiated by trusted controller vaults.