SHERLOCK

Security Review For HaHa Technologies

Collaborative Audit Prepared For: **HaHa Technologies**
Lead Security Expert(s): **PeterSR**
T1MOH
Date Audited: **May 28 - May 31, 2026**

Introduction

HaHa Technologies Inc (Permutize) is a developer of numerous Web3 products. This vault smart contract is for a delta-neutral stablecoin vault. It deploys to lending protocols and auto rebalances to achieve better than average market returns. Can support both USDC and AUSDC on Monad (EVM).

Scope

Repository: Permutize/smart-vault-contracts

Audited Commit: abbff910871e5c55a512349c093453ffd6eda1b9

Final Commit: 03b3a95b553d07721faa7909ecc1e11022f2bda2

Files:

- src/adapters/AdapterHarness.sol
- src/adapters/curvance/CurvanceCollateralAdapter.sol
- src/adapters/curvance/CurvanceDebtAdapter.sol
- src/adapters/euler/EulerCollateralAdapter.sol
- src/adapters/euler/EulerDebtAdapter.sol
- src/adapters/interfaces/ICollateralAdapter.sol
- src/adapters/interfaces/ICurvanceWrappedMON.sol
- src/adapters/interfaces/IDebtAdapter.sol
- src/adapters/interfaces/IHooks.sol
- src/adapters/interfaces/IPoolManager.sol
- src/adapters/interfaces/IUniswapV3SwapRouter.sol
- src/adapters/interfaces/IUniswapV4.sol
- src/adapters/merkl/MerklAdapter.sol
- src/adapters/rewards/RewardsAdapter.sol
- src/adapters/rewards/Swapper.sol
- src/common/ShMonSwapAdapter.sol
- src/common/SwapAdapter.sol
- src/common/WMonUnwrapper.sol
- src/fee/VaultFeeAllocator.sol
- src/GatedVaultImpl.sol

- src/interface/IBaseStrategy.sol
- src/interface/IChainlink.sol
- src/interface/ICurvance.sol
- src/interface/IEuler.sol
- src/interface/IGatedVault.sol
- src/interface/IPyth.sol
- src/interface/IShMon.sol
- src/interface/IWMonUnwrapper.sol
- src/interface/IWrappedToken.sol
- src/libraries/LTVCalculations.sol
- src/libraries/OracleHelpers.sol
- src/libraries/StrategyCommandSelectors.sol
- src/libraries/StrategyRolesLib.sol
- src/libraries/StrategyStorageLib.sol
- src/oracle/ShMonImpliedOracle.sol
- src/strategies/AusdStrategy.sol
- src/strategies/AusdStrategyUnstakeWorker.sol
- src/strategies/BaseStrategy.sol

Final Commit Hash

03b3a95b553d07721faa7909ecc1e11022f2bda2

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	0	2

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue L-1: `EulerDebtAdapter.getMaxBorrowAmount()` doesn't account for Euler limits [RESOLVED]

Source: https://github.com/sherlock-audit/2026-05-permutize-haha-update-may-28th-2026/issues/46

Summary

Euler has 2 unaccounted limits: 1) max debt; 2) available cash.

Vulnerability Detail

Currently `EulerDebtAdapter.getMaxBorrowAmount()` takes into consideration only LTV:

```
function getMaxBorrowAmount(address account, uint256 targetLtv, uint256 maxDebt)
↪  external view returns (uint256) {
    // Read collateral from the paired Euler vault and normalize it into underlying
    ↪  asset units
    // before passing the balance into the generic LTV borrow-cap helper.
    uint256 collateralShares = COLLATERAL_TOKEN.balanceOf(account);
    uint256 collateral = collateralShares == 0 ? 0 :
    ↪  COLLATERAL_TOKEN.convertToAssets(collateralShares);
    uint256 debt = DEBT_TOKEN.debtOf(account);

    // Euler does not impose the Curvance-style minimum loan-size rule here, so the
    ↪  generic
    // oracle-aware LTV calculation is sufficient for maximum borrow sizing.
    return LTVCalculations.calculateMaxBorrow(
        collateral,
        debt,
        COLLATERAL_ORACLE,
        DEBT_ORACLE,
        COLLATERAL_DECIMALS,
        DEBT_DECIMALS,
        targetLtv,
        maxDebt,
        COLLATERAL_ORACLE_HEARTBEAT,
        DEBT_ORACLE_HEARTBEAT
    );
}
```

However Euler Vault has borrow cap checked in the end of operation:

```
uint256 prevBorrows = snapshotBorrows.toUint();
uint256 borrows = vaultCache.totalBorrows.toAssetsUp().toUint();
```

```
if (borrows > vaultCache.borrowCap && borrows > prevBorrows) revert
↪  E_BorrowCapExceeded();
```

Also user can't borrow more than available cash, so you should also account for it:

```
    function borrow(uint256 amount, address receiver) public virtual nonReentrant
    ↪  returns (uint256) {
        (VaultCache memory vaultCache, address account) = initOperation(OP_BORROW,
        ↪  CHECKACCOUNT_CALLER);

        Assets assets = amount == type(uint256).max ? vaultCache.cash :
        ↪  amount.toAssets();
        if (assets.isZero()) return 0;

@>      if (assets > vaultCache.cash) revert E_InsufficientCash();

        increaseBorrow(vaultCache, account, assets);

        pushAssets(vaultCache, receiver, assets);

        return assets.toUint();
    }
```

Impact

Actions close to Euler caps can unexpectedly revert.

Code Snippet

https://github.com/sherlock-audit/2026-05-permutize-haha-update-may-28th-2026/blob/main/smart-vault-contracts/src/adapters/euler/EulerDebtAdapter.sol#L122-L143

Tool Used

Manual Review

Recommendation

Fetch borrow cap via this getter and convert into amount via this library.

Also bound borrowable amount by available cash using this function.

Discussion

0xdavid7

Fixed it: https://github.com/Permutize/smart-vault-contracts/pull/200

T1MOH593

Fix confirmed.

Issue L-2: `setTargetLtvs` Accepts 100% LTV Due to Off-by-One in Validation [RESOLVED]

Source: https://github.com/sherlock-audit/2026-05-permutize-haha-update-may-28th-2026/issues/47

Summary

`setTargetLtvs` uses a strict greater-than check against `LTVCalculations.BPS` (10,000), which means passing exactly `10000` – representing 100% LTV – passes validation and is stored as the target.

Vulnerability Detail

The validation at `BaseStrategy.sol:1342` reads:

```
function setTargetLtvs(uint256 primaryLtv, uint256 secondaryLtv) external
↪  onlyStrategyAdmin {
    if (primaryLtv > LTVCalculations.BPS || secondaryLtv > LTVCalculations.BPS) {
        revert InvalidMaxLtv(primaryLtv, secondaryLtv);
    }
    $.maxPrimaryLtv = primaryLtv;
    $.maxSecondaryLtv = secondaryLtv;
}
```

`LTVCalculations.BPS = 1e4 = 10000`. The condition `primaryLtv > 10000` only rejects values strictly above 100%, not 100% itself. Calling `setTargetLtvs(10000, 10000)` succeeds and stores 100% as the target LTV for both legs.

The strategy's default values are 90% (`DEFAULT_MAX_PRIMARY_LTV_BPS = 9000`), making it clear the intent is to operate below 100%. External lending protocols Curvance and Euler enforce their own liquidation thresholds below 100%, so a strategy position pushed to 100% LTV is immediately liquidatable on any adverse price movement.

Impact

If the admin sets 100% LTV, the next leverage operation that reaches the target will push both legs to full collateralization. Any price movement in either direction triggers immediate liquidation on the lending protocol, with the resulting penalty (typically 5–15%) absorbed by vault depositors.

Code Snippet

```
// BaseStrategy.sol:1341-1344
function setTargetLtvs(uint256 primaryLtv, uint256 secondaryLtv) external
↪  onlyStrategyAdmin {
    if (primaryLtv > LTVCalculations.BPS || secondaryLtv > LTVCalculations.BPS) {
        revert InvalidMaxLtv(primaryLtv, secondaryLtv);
    }
```

```
// LTVCalculations.sol:21
uint256 public constant BPS = 1e4;
```

Tool Used

Manual Review

Recommendation

Change > to >= to reject 100% LTV, and add an explicit safe ceiling constant well below liquidation thresholds:

```
uint256 public constant MAX_SAFE_LTV_BPS = 9000; // 90%

if (primaryLtv >= LTVCalculations.BPS || secondaryLtv >= LTVCalculations.BPS) {
    revert InvalidMaxLtv(primaryLtv, secondaryLtv);
}
```

Discussion

0xdavid7

Fixed it: https://github.com/Permutize/smart-vault-contracts/pull/200

PeterSRWeb3

Fix confirmed.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.