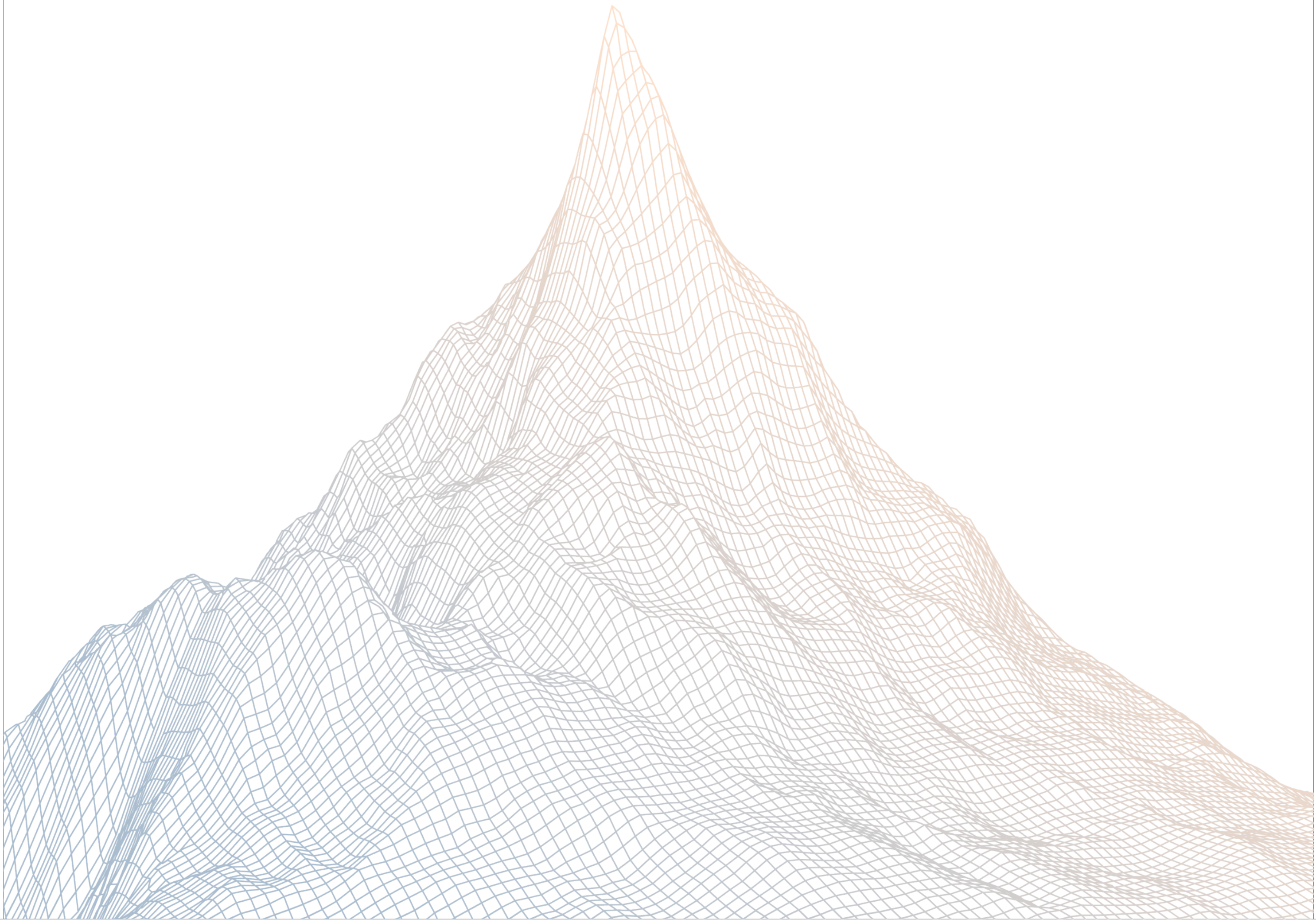
Zenith

Valantis

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: August 6th to August 7th, 2025

AUDITED BY: cccz
said

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2 Executive Summary

2.1 About Valantis

Valantis is a novel approach to representing Value-Exchange Logic, Valantis Core Pools can recover the entire DEX space using composable modules. It attempts to be a full generalization of smart-contract DEXes, enforcing strict security assumptions over the interactions between modules and users to create the most secure, composable, and developer friendly environment for DEX development. The Valantis Sovereign Pools integrate various components responsible for functions such as pricing logic, fee calculation, oracle services, reserve vaults, and liquidity management strategies. Sovereign Pools natively support rebase tokens.

2.2 Scope

The engagement involved a review of the following targets:

Target	valantis-stex-khype
Repository	https://github.com/ValantisLabs/valantis-stex-khype
Commit Hash	`aa748defdf55107659cecfab383cbbbd8a41a53c`
Files	`MultiMarketLendingModule.sol` `ERC4626LendingModule.sol`

2.3 Audit Timeline

August 6th, 2025	Audit start
August 7th, 2025	Audit end
August 11th, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	1
Low Risk	2
Informational	1
Total Issues	**5**

3

Findings Summary

ID	Description	Status
H-1	ManagerFee will be incorrect if the withdrawal exceeds totalPrincipal	Resolved
M-1	withdraw of all assets in MultiMarketLendingModule will frequently revert	Resolved
L-1	deposit does not consider the vault's max deposit limit	Acknowledged
L-2	Add duplicate checks to _lendingModules	Resolved
I-1	Lack of asset validation for lendingModule in initialize	Acknowledged

4 Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] ManagerFee will be incorrect if the withdrawal exceeds `totalPrincipal`

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- MultiMarketLendingModule.sol#L565
- MultiMarketLendingModule.sol#L504

Description:

The protocol uses the total balance minus the total principal as the yield to calculate the ManagerFee.

```
function _getManagerFee(uint256 totalBalance)
 private view returns (uint256) {
    uint256 totalNetYield =
    totalBalance > totalPrincipal ? totalBalance - totalPrincipal : 0;

    uint256 totalManagerClaimable =
     Math.mulDiv(totalNetYield, managerFeeBips, BIPS);

    return totalManagerClaimable >
     totalManagerClaimed ? totalManagerClaimable - totalManagerClaimed : 0;
}
```

However, the problem here is that if the withdrawal exceeds the `totalPrincipal`, the yield calculation will be incorrect, resulting in the incorrect ManagerFee.

```
totalPrincipal =
totalPrincipal > amountReceived ? totalPrincipal - amountReceived : 0;
```

Consider a 10% Manager Fee.

1. The manager deposits 1000 tokens, and the `totalPrincipal` is 1000.
2. 100 yield is generated, and the Manager Fee should be (1100 - 1000) * 10% = 10, with the maxWithdraw of 1090.
3. The manager withdraws 1050, and the totalPrincipal is set to 0, leaving the protocol with 50 tokens. The calculated Manager Fee is now (50 - 0) * 10% = 5.

POC:

```
diff --git a/src/mocks/MockERC4626LendingPool.sol
b/src/mocks/MockERC4626LendingPool.sol
index 05ddc92..0ee4452 100644
-- a/src/mocks/MockERC4626LendingPool.sol
++ b/src/mocks/MockERC4626LendingPool.sol
@@ -116,4 +116,7
@@ contract MockERC4626LendingPool {
     function removeToken(uint256 amount, address recipient) external {
         ERC20Mock(underlyingAsset).safeTransfer(recipient, amount);
     }
    function addTotalAssets(uint256 amount) external {
        _totalAssets += amount;
    }
 }
diff --git
a/test/MultiMarketLendingModule.t.sol
b/test/MultiMarketLendingModule.t.sol
index ca141ae..13a9743 100644
-- a/test/MultiMarketLendingModule.t.sol
++ b/test/MultiMarketLendingModule.t.sol
@@ -1202,4 +1202,93
@@ contract MultiMarketLendingModuleTest is Test {
         vm.prank(owner);
         multiLendingModule.initialize(lendingModules, configs);
     }
    function testIncorrectManagerFee() public {
        // Deploy with manager fees
        vm.prank(owner);
        MultiMarketLendingModule multiLendingModuleWithFees =
            new MultiMarketLendingModule
 (address(asset), owner, manager, tokenSweepManager, 1000); // 10% fee
```

```
    ERC4626LendingModule newLendingModule1 =

new ERC4626LendingModule(address(mockPool1), address(multiLendingM
           oduleWithFees), tokenSweepManager);
    ERC4626LendingModule newLendingModule2 =

new ERC4626LendingModule(address(mockPool2), address(multiLendingM
           oduleWithFees), tokenSweepManager);

    address[] memory lendingModules = new address[](2);
    lendingModules[0] = address(newLendingModule1);
    lendingModules[1] = address(newLendingModule2);

    MultiMarketLendingModule.LendingModuleConfig[] memory configs =
        new MultiMarketLendingModule.LendingModuleConfig[](2);
    configs[0] = MultiMarketLendingModule.LendingModuleConfig({
        depositWeightBips: 5000, // 50%
        withdrawWeightBips: 5000 // 50%
    });
    configs[1] = MultiMarketLendingModule.LendingModuleConfig({
        depositWeightBips: 5000, // 50%
        withdrawWeightBips: 5000 // 50%
    });

    vm.prank(owner);
    multiLendingModuleWithFees.initialize(lendingModules, configs);

    // Setup balances and approvals
    asset.mint(manager, INITIAL_BALANCE);
    vm.prank(manager);

asset.approve(address(multiLendingModuleWithFees), type(uint256).max);

    // Deposit
    vm.prank(manager);
    multiLendingModuleWithFees.deposit(1000e18);

    // Simulate yield
```

```
        asset.mint(address(mockPool1), 50e18);
        mockPool1.addTotalAssets(50e18);
        asset.mint(address(mockPool2), 50e18);
        mockPool2.addTotalAssets(50e18);

        uint256 snapshotId = vm.snapshot();

        vm.prank(owner);
        multiLendingModuleWithFees.claimManagerFee(recipient);
        console.log(multiLendingModuleWithFees.totalManagerClaimed()); // 10e18

        vm.revertTo(snapshotId);
        vm.prank(manager);
        multiLendingModuleWithFees.withdraw(1000e18, recipient);
        vm.prank(owner);
        multiLendingModuleWithFees.claimManagerFee(recipient);
        console.log(multiLendingModuleWithFees.totalManagerClaimed()); // 10e18

        vm.revertTo(snapshotId);
        vm.prank(manager);
        multiLendingModuleWithFees.withdraw(1030e18, recipient);
        vm.prank(owner);
        multiLendingModuleWithFees.claimManagerFee(recipient);
        console.log(multiLendingModuleWithFees.totalManagerClaimed()); // 7e18

        vm.revertTo(snapshotId);
        vm.prank(manager);
        multiLendingModuleWithFees.withdraw(1090e18, recipient);
        vm.prank(owner);
        multiLendingModuleWithFees.claimManagerFee(recipient);

    console.log(multiLendingModuleWithFees.totalManagerClaimed()); // 1e18

    }
}
```

Recommendations:

It is recommended to add a `virtualYeild` variable to record the yield that has been withdrawn.

```diff
diff --git a/src/lending-modules/MultiMarketLendingModule.sol
b/src/lending-modules/MultiMarketLendingModule.sol
index 14f8511..b1a2487 100644
-- a/src/lending-modules/MultiMarketLendingModule.sol
++ b/src/lending-modules/MultiMarketLendingModule.sol
@@ -120,6 +120,9
@@ contract MultiMarketLendingModule is ILendingModule, Ownable2Step {
      */
     uint256 public totalPrincipal;


    uint256 public virtualYeild;

     /**
      * @notice Total amount of `asset` token manager fees
      * already claimed by the manager.
      */
@@ -503,6 +506,8
@@ contract MultiMarketLendingModule is ILendingModule,
     Ownable2Step {
     function _getManagerFee(uint256 totalBalance)
     private view returns (uint256) {
         uint256 totalNetYield =
         totalBalance > totalPrincipal ?
          totalBalance - totalPrincipal : 0;

        totalNetYield += virtualYeild;

         uint256 totalManagerClaimable =
         Math.mulDiv(totalNetYield, managerFeeBips, BIPS);

         return totalManagerClaimable >
         totalManagerClaimed ? totalManagerClaimable - totalManagerClaimed :
    0;
@@ -562,6 +567,12
@@ contract MultiMarketLendingModule is ILendingModule,
    Ownable2Step {
         }

         // Update total principal
        totalPrincipal = totalPrincipal > amountReceived ?
```

```
         totalPrincipal - amountReceived : 0;
        // totalPrincipal = totalPrincipal >
        amountReceived ? totalPrincipal - amountReceived : 0;
        if (totalPrincipal > amountReceived){
            totalPrincipal -= amountReceived;
        } else {
            virtualYeild += amountReceived - totalPrincipal;
            totalPrincipal = 0;
        }
    }
}
```

Valantis: Resolved with PR-23

Zenith: Verified.

4.2 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] withdraw of all assets in `MultiMarketLendingModule` will frequently revert

SEVERITY: Medium	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- MultiMarketLendingModule.sol#L528-L536

Description:

When `withdraw` is called by the manager, it iterates through the `_lendingModules` and calculates the `amountToWithdraw` from each module using the configured `withdrawWeightBips`, then withdraws the assets from each module using the calculated `amountToWithdraw`.

- MultiMarketLendingModule.sol#L528-L536

```
function _withdraw(uint256 amount, uint256 maxWithdraw,
address recipient)
    private
    returns (uint256 amountReceived)
{
    // Cannot withdraw in excess
    if (amount > maxWithdraw) {
        revert
        MultiMarketLendingModule__withdraw_InsufficientBalance();
    }

    IERC20 token = IERC20(asset);

    uint256 amountRemaining = amount;
    uint256 recipientPreBalance = token.balanceOf(recipient);
    for (uint256 i = 0; i < _lendingModules.length(); i++) {
            address lendingModule = _lendingModules.at(i);
            LendingModuleConfig memory
            lendingModuleConfig = _lendingModuleConfigs[lendingModule];
            uint256 amountToWithdraw = Math.min(
>>>             Math.mulDiv(
                    amount,
                    lendingModuleConfig.withdrawWeightBips,
                    BIPS,
                    Math.Rounding.Ceil // rounding up to avoid dust left-over
    Ceil
                ),
                amountRemaining
            );

            if (amountToWithdraw == 0) {
                continue;
            }

            amountRemaining -= amountToWithdraw;

        // WARNING: This might temporarily revert if:
        // 1) Lending Module has insufficient liqudity due to high
    utilization
        // 2) `amountWithdraw` is greater than the Lending Module's available
    liquidity.
        // If 2), `owner` is required to adjust withdrawal weights
    proportionally.
        // If 1), `manager` needs to wait for the Lending Module to have
    sufficient liquidity.
        ILendingModule(lendingModule).withdraw(amountToWithdraw, recipient);
        }

        uint256 recipientPostBalance = token.balanceOf(recipient);
        if (recipientPostBalance < recipientPreBalance + amount) {
            revert
            MultiMarketLendingModule__withdraw_InsufficientAmountReceived();
        }

        amountReceived = recipientPostBalance - recipientPreBalance;

        // Cannot withdraw in excess
        if (amountReceived > maxWithdraw) {
            revert
             MultiMarketLendingModule__withdraw_ExcessiveAmountReceived();
        }

        // Update total principal
        totalPrincipal = totalPrincipal > amountReceived ? totalPrincipal
    - amountReceived : 0;
    }
```

And when replacing the `lendingModule` in `setProposedLendingModule()`, all the assets in the old lendingModule are required to be withdrawn.

https://github.com/ValantisLabs/valantis-stex-khype/blob/main/src/withdrawal-modules/stHYPEWithdrawalModule.sol#L404-L420

```
    function setProposedLendingModule()
    external onlyOwner whenPoolNotLocked {
        if (lendingModuleProposal.startTimestamp > block.timestamp) {
            revert

    stHYPEWithdrawalModule__setProposedLendingModule_ProposalNotActive();
        }

        if (lendingModuleProposal.startTimestamp == 0) {
            revert

    stHYPEWithdrawalModule__setProposedLendingModule_InactiveProposal();
        }

        // Withdraw all token1 amount from lending module back into pool
        if (address(lendingModule) != address(0)) {
            uint256 amountToken1LendingModule
    = lendingModule.assetBalance();

            if (amountToken1LendingModule > 0) {
>>>           lendingModule.withdraw(amountToken1LendingModule, pool);
            }
        }
    }
```

The problem is that it calculates the `amountToWithdraw` using a ceiling operation without checking the actual maximum amount that can be withdrawn from each corresponding module.

In the case of withdrawing all principal and yield from the `MultiMarketLendingModule`, rounding up the `amountToWithdraw` can lead to an overestimation of the total principal and yield stored in the corresponding modules, causing the operation to revert, even if the withdraw weights have been properly adjusted.

PoC :

```
function testWithdrawFail() public {
    // Deploy with manager fees
    vm.prank(owner);
    MultiMarketLendingModule multiLendingModuleWithFees =
        new MultiMarketLendingModule(address(asset), owner, manager,
    tokenSweepManager, 1000); // 10% fee

    _initializeLendingModule(multiLendingModuleWithFees);

    // Setup balances and approvals
    asset.mint(manager, INITIAL_BALANCE);
    vm.prank(manager);
    asset.approve
    (address(multiLendingModuleWithFees), type(uint256).max);

    // Deposit
    vm.prank(manager);
    multiLendingModuleWithFees.deposit(1000e18);

    // Simulate yield
    asset.mint(address(mockPool1), 100e18);
    asset.mint(address(mockPool2), 50e18);

    uint256 recipientBalanceBefore = asset.balanceOf(recipient);
    uint256 expectedFee = (150e18 * 1000) / 10000; // 15e18

    // Claim fees
    vm.expectEmit(true, false, false, true);
    emit ManagerFeeClaimed(recipient, expectedFee);

    vm.prank(owner);
    multiLendingModuleWithFees.claimManagerFee(recipient);

    assertEq
    (asset.balanceOf(recipient), recipientBalanceBefore + expectedFee);
    assertEq
    (multiLendingModuleWithFees.totalManagerClaimed(), expectedFee);
    assertEq
    (multiLendingModuleWithFees.managerFeeClaimable(), 0);

    uint256 finalBalance = multiLendingModuleWithFees.assetBalance();
    uint256 balancePool1 = asset.balanceOf(address(mockPool1));
    uint256 balancePool2 = asset.balanceOf(address(mockPool2));
    console.log
    ("percentage pool1 :  ", balancePool1 * 10000 / finalBalance);
    console.log
    ("percentage pool2 : ", balancePool2 * 10000 / finalBalance);

    uint16[] memory newWeights = new uint16[](2);
    newWeights[0] = uint16(balancePool1 * 10000 / finalBalance);
    newWeights[1] = uint16(balancePool2 * 10000 / finalBalance) + 1;

    vm.expectEmit(true, true, true, true);
    emit WithdrawWeightsSet(newWeights);

    vm.prank(owner);
    multiLendingModuleWithFees.setWithdrawWeights(newWeights);

    vm.prank(manager);
    vm.expectRevert();
    multiLendingModuleWithFees.withdraw(finalBalance, recipient);
}
```

And the assets distributed in the `MultiMarketLendingModule` will most likely not be distributed with a precision of 1/10000(withdrawWeightBips precision) and thus cannot be completely withdrawn.

PoC:

```
    function testFullyWithdrawFail() public {
        // Deploy with manager fees
        vm.prank(owner);
        MultiMarketLendingModule multiLendingModuleWithFees =
            new MultiMarketLendingModule(address(asset), owner, manager,
    tokenSweepManager, 1000); // 10% fee

        ERC4626LendingModule newLendingModule1 =
            new ERC4626LendingModule(address(mockPool1),
    address(multiLendingModuleWithFees), tokenSweepManager);
        ERC4626LendingModule newLendingModule2 =
            new ERC4626LendingModule(address(mockPool2),
    address(multiLendingModuleWithFees), tokenSweepManager);

        address[] memory lendingModules = new address[](2);
        lendingModules[0] = address(newLendingModule1);
        lendingModules[1] = address(newLendingModule2);

        MultiMarketLendingModule.LendingModuleConfig[] memory configs =
            new MultiMarketLendingModule.LendingModuleConfig[](2);
        configs[0] = MultiMarketLendingModule.LendingModuleConfig({
            depositWeightBips: 5000, // 50%
            withdrawWeightBips: 5000 // 50%
        });
        configs[1] = MultiMarketLendingModule.LendingModuleConfig({
            depositWeightBips: 5000, // 50%
            withdrawWeightBips: 5000 // 50%
        });

        vm.prank(owner);
        multiLendingModuleWithFees.initialize(lendingModules, configs);

        // Setup balances and approvals
        asset.mint(manager, INITIAL_BALANCE);
        vm.prank(manager);
        asset.approve(address(multiLendingModuleWithFees),
    type(uint256).max);

        // Deposit
        vm.prank(manager);
        multiLendingModuleWithFees.deposit(1000e18);

        // Simulate yield
        asset.mint(address(mockPool1), 50e18 + 1000); // and 1000 dust
        mockPool1.addTotalAssets(50e18 + 1000);
        asset.mint(address(mockPool2), 50e18);
        mockPool2.addTotalAssets(50e18);
        vm.prank(owner);
        multiLendingModuleWithFees.claimManagerFee(recipient);
        console.log(multiLendingModuleWithFees.totalManagerClaimed()); //
    10e18
        uint256 assetBalance = multiLendingModuleWithFees.assetBalance();
        console.log(assetBalance);

        vm.prank(manager);
        multiLendingModuleWithFees.withdraw(assetBalance, recipient); //
    revert
}
```

Recommendations:

Consider also checking the maximum withdrawable amount from each module when calculating `amountToWithdraw`.

```
// ...
            uint256 amountToWithdraw = Math.min(
                Math.mulDiv(
                    amount,
                    lendingModuleConfig.withdrawWeightBips,
                    BIPS,
                    Math.Rounding.Ceil
                    // rounding up to avoid dust left-over Ceil
                ),
                amountRemaining
            );
      amountToWithdraw =
       Math.min
       (amountToWithdraw, ILendingModule(lendingModule).assetBalance());
// ...
```

And consider implement new logic in `MultiMarketLendingModule.withdraw()` to withdraw all assets from _lendingModules when the withdrawal amount is equal to the `assetBalance()`.

Valantis: Resolved with PR-25.

Zenith: Verified.

4.3 Low Risk

A total of 2 low risk findings were identified.

[L-1] `deposit` does not consider the vault's max deposit limit

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- [MultiMarketLendingModule.sol#L418-L426(https://gitea.zellic.io/mirror-zenith/ValantisLabs-valantis-stex-khype/src/branch/main/src/lending-modules/MultiMarketLendingModule.sol#L418-L426)

Description:

When `deposit` is performed, it doesn't check `lendingModules's` vault max deposit limit.

```
    function deposit(uint256 _amount)
    external override onlyManager onlyWhenInitialized {
        if (_amount == 0) {
            revert MultiMarketLendingModule__deposit_InvalidAmount();
        }

        IERC20 token = IERC20(asset);
        token.safeTransferFrom(msg.sender, address(this), _amount);

        uint256 amountRemaining = _amount;
        uint256 preBalance = token.balanceOf(address(this));
        for (uint256 i = 0; i < _lendingModules.length(); i++) {
            address lendingModule = _lendingModules.at(i);
            LendingModuleConfig memory lendingModuleConfig
    = _lendingModuleConfigs[lendingModule];
>>>         uint256 amountToDeposit = Math.min(
                Math.mulDiv(
                    _amount,
                    lendingModuleConfig.depositWeightBips,
                    BIPS,
                    Math.Rounding.Ceil // rounding up to avoid dust left-over
                ),
                amountRemaining
            );

            if (amountToDeposit == 0) {
                continue;
            }

            amountRemaining -= amountToDeposit;

            token.forceApprove(lendingModule, amountToDeposit);
            ILendingModule(lendingModule).deposit(amountToDeposit);
        }

        uint256 amountDeposited = preBalance
    - token.balanceOf(address(this));
        if (amountDeposited != _amount || amountRemaining > 0) {
            revert
    MultiMarketLendingModule__deposit_PartialDepositNotAllowed();
        }

        // Update total principal
        totalPrincipal += _amount;
    }
```

Standard `ERC4626` vaults, especially the Felix Hype vault, which is planned to be integrated into the `MultiMarketLendingModule` have a maximum deposit limit.

```
function _maxDeposit() internal view returns (uint256 totalSuppliable) {
    for (uint256 i; i < supplyQueue.length; ++i) {
        Id id = supplyQueue[i];

        uint256 supplyCap = config[id].cap;
        if (supplyCap == 0) continue;

        uint256 supplyShares = MORPHO.supplyShares(id, address(this));
        (uint256 totalSupplyAssets, uint256 totalSupplyShares,,)
    = MORPHO.expectedMarketBalances(_marketParams(id));
        // `supplyAssets` needs to be rounded up for `totalSuppliable` to be
    rounded down.
        uint256 supplyAssets = supplyShares.toAssetsUp(totalSupplyAssets,
    totalSupplyShares);

        totalSuppliable += supplyCap.zeroFloorSub(supplyAssets);
    }
}
```

Not considering `_maxDeposit` can potentially cause a revert when deposit is performed and may require frequent management of deposit weights.

Recommendations:

When performing `deposit`, consider skipping vaults that have reached their maximum deposit limit and reallocating the amount to other `lendingModules`.

Valantis: Acknowledged.

[L-2] Add duplicate checks to `_lendingModules`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- MultiMarketLendingModule.sol#L286

Description:

When `_lendingModules` contains duplicate elements, it passes the check in `initialize()`, and the later element's `lendingModuleConfig` overrides the earlier element's `lendingModuleConfig`.

```
ILendingModule(lendingModule).assetBalance();

_lendingModules.add(lendingModule);

LendingModuleConfig memory lendingModuleConfig
    = _lendingModuleConfigArray[i];

_lendingModuleConfigs[lendingModule] = lendingModuleConfig;

totalDepositWeightBips += lendingModuleConfig.depositWeightBips;
totalWithdrawWeightBips += lendingModuleConfig.withdrawWeightBips;
```

And it fails during deposit and withdraw because `totalDepositWeightBips/totalWithdrawWeightBips` is not equal to BIPS .

```
        if (amountDeposited ≠ _amount || amountRemaining > 0) {
            revert
    MultiMarketLendingModule__deposit_PartialDepositNotAllowed();
        }
...
        if (recipientPostBalance < recipientPreBalance + amount) {
            revert
    MultiMarketLendingModule__withdraw_InsufficientAmountReceived();
        }
```

A bad case is that if a misconfigured `MultiMarketLendingModule` is used, since its deposit/withdraw always fails, a malicious user can donate to it so that `assetBalance()` is not 0, and thus it cannot be replaced in the `setProposedLendingModule()` due to withdraw failures.

```
// Withdraw all token1 amount from lending module back into pool
if (address(lendingModule) ≠ address(0)) {
    uint256 amountToken1LendingModule = lendingModule.assetBalance();

    if (amountToken1LendingModule > 0) {
        lendingModule.withdraw(amountToken1LendingModule, pool);
    }
}
```

POC:

```
diff --git a/test/MultiMarketLendingModule.t.sol
    b/test/MultiMarketLendingModule.t.sol
index ca141ae..13a9743 100644
-- a/test/MultiMarketLendingModule.t.sol
++ b/test/MultiMarketLendingModule.t.sol
@@ -1202,4 +1202,93 @@ contract MultiMarketLendingModuleTest is Test {
         vm.prank(owner);
         multiLendingModule.initialize(lendingModules, configs);
     }
    function testInitializeDuplicatedLendingModules() public {
        address[] memory lendingModules = new address[](2);
        lendingModules[0] = address(lendingModule1);
        lendingModules[1] = address(lendingModule1);

        MultiMarketLendingModule.LendingModuleConfig[] memory configs =
            new MultiMarketLendingModule.LendingModuleConfig[](2);

    configs[0] = MultiMarketLendingModule.LendingModuleConfig({depositWeig
            htBips: 6000, withdrawWeightBips: 4000});

    configs[1] = MultiMarketLendingModule.LendingModuleConfig({depositWeig
            htBips: 4000, withdrawWeightBips: 6000});

        vm.prank(owner);
        multiLendingModule.initialize(lendingModules, configs);
```

```
         // Verify state

      address[] memory storedModules = multiLendingModule.lendingModules();
         assertEq(storedModules.length, 1);
     }
```

Recommendations:

It is recommended to require the return value of `_lendingModules.add()` to be true for duplicate checking.

```
diff --git a/src/lending-modules/MultiMarketLendingModule.sol
    b/src/lending-modules/MultiMarketLendingModule.sol
index 14f8511..9d99398 100644
-- a/src/lending-modules/MultiMarketLendingModule.sol
++ b/src/lending-modules/MultiMarketLendingModule.sol
@@ -43,6 +43,7 @@ contract MultiMarketLendingModule is ILendingModule,
    Ownable2Step {
     error MultiMarketLendingModule__deposit_PartialDepositNotAllowed();
     error MultiMarketLendingModule__initialize_AlreadyInitialized();
     error MultiMarketLendingModule__initialize_ExceededMaxLendingModules();
    error MultiMarketLendingModule__initialize_DuplicatedLendingModules();
@@ -283,7 +287,7 @@ contract MultiMarketLendingModule is ILendingModule,
    Ownable2Step {
             // Sanity check that it is possible to query `assetBalance`
             ILendingModule(lendingModule).assetBalance();

            _lendingModules.add(lendingModule);

    require(_lendingModules.add(lendingModule),MultiMarketLendingModul
               e__initialize_DuplicatedLendingModules());

             LendingModuleConfig memory lendingModuleConfig
    = _lendingModuleConfigArray[i];
```

Valantis: Resolved with PR-24

Zenith: Verified.

4.4 Informational

A total of 1 informational findings were identified.

[I-1] Lack of asset validation for `lendingModule` in `initialize`

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- MultiMarketLendingModule.sol#L250-L294

Description:

When `MultiMarketLendingModule.initialize` is called, it does not validate whether the provided lending module's asset matches the `MultiMarketLendingModule`'s configured asset.

```
function initialize(address[] memory _lendingModuleArray,
    LendingModuleConfig[] memory _lendingModuleConfigArray)
    external
    onlyOwner
{
    // ...

        // Sanity check that it is possible to query `assetBalance`

        ILendingModule(lendingModule).assetBalance();

        _lendingModules.add(lendingModule);

        LendingModuleConfig memory lendingModuleConfig
    = _lendingModuleConfigArray[i];

        _lendingModuleConfigs[lendingModule] = lendingModuleConfig;

        // ...
}
```

This can cause issues, such as being unable to perform deposit to the module or incorrect results when `assetBalance` is called, due to the inclusion of a lending module with a different underlying asset.

Recommendations:

Consider ensuring that the `lendingModule`'s `asset` is equal to the `MultiMarketLendingModule`'s `asset`.

Valantis: Acknowledged.