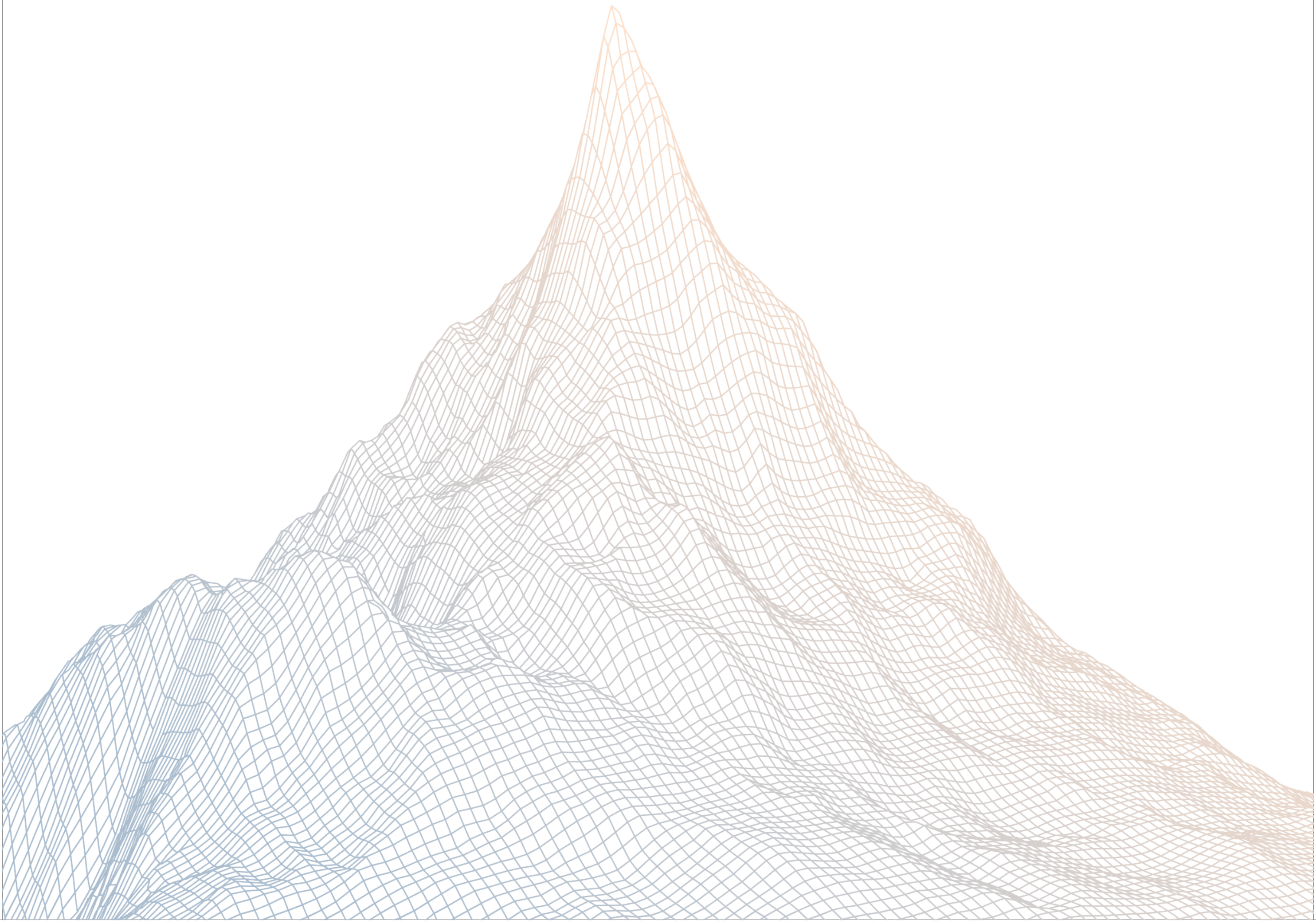
Zenith

Theo

Smart Contract
Security Assessment

VERSION 1.1



AUDIT DATES: July 14th to July 18th, 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 Theo

Theo is building the infrastructure for a new kind of financial system: one that makes high-quality assets accessible to anyone, anywhere. In today's world, moving money globally is instant, but flexible access to global markets remains gated behind institutional barriers and legacy systems. We believe that must evolve.

Theo bridges this gap by bringing high-quality real-world assets onchain in a way that is truly compelling to retail and institutional investors.

2.2 Scope

The engagement involved a review of the following targets:

Target	contracts-v2
Repository	https://github.com/theo-network/contracts-v2
Commit Hash	`d70c820f7f5a5f965e3061aa9545132328c85b5c`

Files	`vaults/*`
	`BaseUpgradeable.sol`
	`TheoWhitelist.sol`
	`IToken.sol`
	`TToken.sol`
	`TTokenEscrow.sol`
	`TTokenRouter.sol`

2.3 Audit Timeline

July 14, 2025	Audit start
July 18, 2025	Audit end
July 22, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	8
Low Risk	3
Informational	2
Total Issues	**13**

3 Findings Summary

ID	Description	Status
M-1	The last withdrawer of TToken can be DoSed.	Acknowledged
M-2	checkRatio can be DoSed by donating one of the deposit tokens.	Resolved
M-3	IToken does not implement the inherited pause capability	Resolved
M-4	Blacklisted users can renounce their blacklist role.	Resolved
M-5	Attacker can mint any shares without providing any assets	Resolved
M-6	TToken.seize() should not fail due to insufficient assets	Resolved
M-7	Attacker can exploit the divide-by-0 error in _convertToAssets() to cause DOS	Resolved
M-8	IToken.checkAssetList() should require that the TToken's escrowAsset must be asset()	Resolved
L-1	Issues with updateDepositAssets	Resolved
L-2	ERC4626UpgradeableMultiAsset's _depositAssets should not be duplicated	Resolved
L-3	ERC4626UpgradeableMultiAsset.withdraw() should require shares > 0	Resolved
I-1	TTokenRouter could potentially accumulate assets due to bulkCompletePending	Resolved
I-2	Lack of a limit on deposit assets could cause an OOG issue	Acknowledged

4 Findings

4.1 Medium Risk

A total of 8 medium risk findings were identified.

[M-1] The last withdrawer of `TToken` can be DoSed.

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

Target

- TToken.sol#L95-L99

Description:

Due to the `minShares` check in `_withdraw`, the last user in `TToken` may be unable to withdraw their assets.

```
function _withdraw(address _caller, address _receiver, address _owner,
    uint256 _assets, uint256 _shares) internal override checkMinShares {
    super._withdraw(_caller, _receiver, _owner, _assets, _shares);
}
```

```
modifier checkMinShares() {
    _;
    uint256 _totalSupply = totalSupply();
    if (_totalSupply > 0 && _totalSupply < tTokenParams.minShares)
    revert MinSharesError();
}
```

If a malicious user mints a dust amount of shares (less than `minShares`) right before a withdrawal, the operation will always revert due to the `checkMinShares` modifier.

Recommendations:

Consider also enforcing a minimum minted share amount equal to `tTokenParams.minShares`.

Theo: Theo will also be depositing first to ensure `minShares` will be always be met for any future withdrawals.

Zenith: Acknowledged.

[M-2] checkRatio can be DoSed by donating one of the deposit tokens.

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

Target

- ITToken.sol#L43-L60

Description:

When calculating and checking `currentRatioBps`, it uses `depositAssetValues` and `totalValue`, both of which depend on the deposit asset balances held by the contract.

```
    modifier checkRatio() {
        _;
        // done after function is run
        if (config.enforcedRatio) {
            // return (depositAssets, amounts, values, totalUnderlyingValue);
>>>         (, , uint256[] memory depositAssetValues, uint256 totalValue)
    = totalDepositAssets();
            // go through each asset and check ratio
            for (uint256 i = 0; i < config.assetRatiosBps.length; i++) {
                // get current ratio in basis points
                uint256 currentRatioBps
    = depositAssetValues[i].mulDiv(10000, totalValue + 1);
                // get target ratio in basis points
                uint256 targetRatioBps = uint256(config.assetRatiosBps[i]);

                if (currentRatioBps > targetRatioBps + config.maxDeviationBps
    || currentRatioBps < targetRatioBps - config.maxDeviationBps) {
                    revert ErrorBadAssetRatio();
                }
            }
        }
    }
```

This can cause a DoS, as an attacker can intentionally donate assets to the contract and prevent `deposit` or `withdraw` operations from being performed.

Attack path: The attacker mints a dust amount of `IToken`, then transfers assets to the `IToken`,preferably the asset with the lower ratio. This way, the attacker doesn't lose any assets, while the next depositor will be blocked from depositing unless they provide a large enough amount to rebalance the ratio.

```
function testDoSRatio() public {
    uint32[] memory assetRatiosBps = new uint32[](2);
    assetRatiosBps[0] = 9000; // 90%
    assetRatiosBps[1] = 1000; // 10%
    vm.startPrank(owner);
    IIToken.ITokenParams memory config = IIToken.ITokenParams({
                    assetRatiosBps: assetRatiosBps,
                    enforcedRatio: true,
                    maxDeviationBps: 100 // 1%
                });
    iToken.setConfig(config);
    vm.stopPrank();

    vm.startPrank(user1);
    address[] memory assets = new address[](2);
    assets[0] = address(tToken1);
    assets[1] = address(tToken2);
    uint256[] memory amounts = new uint256[](2);
    amounts[0] = 90; // 90% of the total
    amounts[1] = 10; // 10% of the total

    iToken.deposit(assets, amounts, user1);

    tToken2.transfer(address(iToken), 50 ether);
    vm.stopPrank();
    // console.log(tToken2.balanceOf(address(iToken)));
    // console.logUint(iToken.convertToAssets(iToken.balanceOf(user1)));


    address user2 = address(0x1234);
    // need to mint tTokens to user2 for depositing
    vm.startPrank(minterRole);
    tToken1.depositOptimistic(300 ether, user2);
    // tToken2.depositOptimistic(10 ether, user2);
    vm.stopPrank();

    // approve iToken for both tokens
    vm.startPrank(user2);
    tToken1.approve(address(iToken), 300 ether);
    // tToken2.approve(address(iToken), 10 ether);
```

```
        uint256[] memory amounts2 = new uint256[](2);
        amounts[0] = 300 ether; // try to rebalance
        amounts[1] = 0 ether; // try to rebalance
        vm.expectRevert(IIToken.ErrorBadAssetRatio.selector);
        iToken.deposit(assets, amounts, user2);
        vm.stopPrank();
    }
```

Recommendations:

Consider tracking total values using internal balances, or adding a rescue function to withdraw donated assets.

Theo: Resolved with @e270666a1b...

Zenith: Verified.

[M-3] `IToken` does not implement the inherited pause capability

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

Target

- IToken.sol

Description:

`IToken` inherents `BaseUpgradeable` which include `PausableUpgradeable`. However, this capability is not used inside contracts.

```
contract IToken is ERC4626UpgradeableMultiAsset, BaseUpgradeable, IIToken {
    using SafeERC20 for IERC20;
    using Math for uint256;

    /// @notice Config for IToken
    ITokenParams public config;
   // ...
}
```

Recommendations:

Consider implementing the pause capability and restricting critical functions using the `whenNotPaused` modifier.

Theo: Resolved with @ad40409156...

Zenith: Verified.

[M-4] Blacklisted users can renounce their blacklist role.

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

Target

- TheoWhitelist.sol#L43-L45

Description:

TheoWhitelist implements blacklist functionality by assigning the BLACKLIST_ROLE to blacklisted users.

```
function grantBlacklist(address _account)
    external onlyRole(WHITELIST_MANAGER_ROLE) {
    _grantRole(BLACKLIST_ROLE, _account);
    emit GrantBlacklist(_account);
}
```

However, users can call renounceRole to remove the BLACKLIST_ROLE assigned to them.

```
function renounceRole(bytes32 role, address callerConfirmation)
    public virtual {
    if (callerConfirmation ≠ _msgSender()) {
        revert AccessControlBadConfirmation();
    }

    _revokeRole(role, callerConfirmation);
}
```

Recommendations:

Override renounceRole and disable it.

Theo: Resolved with @1b64055033...

Zenith: Verified.

[M-5] Attacker can mint any shares without providing any assets

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

Target

- ERC4626UpgradeableMultiAsset.sol#L205-L211

Description:

In `ERC4626UpgradeableMultiAsset.maxMint()`, when `totalSupply() == 0`, the protocol does not allow `mint()` to be called.

```
function maxMint(address) public view virtual returns (uint256) {
    if (totalSupply() == 0) {
        // If no shares exist, mint cannot be called first
        return 0;
    }
    return type(uint256).max;
}
```

This is because `totalVaultValue` is also 0 at this time, users can mint any shares without providing any assets.

```
function _convertToAssets(uint256 shares, Math.Rounding rounding)
    internal view virtual returns (address[] memory, uint256[] memory) {
    (address[] memory assets, uint256[] memory vaultBalances, uint256[]
    memory assetValues, uint256 totalVaultValue) = totalDepositAssets();
    if (assets.length == 0 || shares == 0) {
        return (assets, new uint256[](assets.length));
    }

    // convert shares to total value in underlying asset terms
    uint256 totalShareValue = shares.mulDiv(totalUnderlyingAssets() + 1,
    totalSupply() + 10 ** _decimalsOffset(), rounding);

    // distribute total value proportianlly based on current valut
    composition
    uint256[] memory amounts = new uint256[](assets.length);
    for (uint i = 0; i < assets.length; i++) {
        if (totalVaultValue == 0 || vaultBalances[i] == 0) {
            amounts[i] = 0;
```

But attacker can construct a situation where `totalSupply()` is not 0 and `totalVaultValue` is 0 to bypass the check in `maxMint()`.

1. Consider IToken1 is composed of IToken2, and IToken2 is composed of TToken1 and TToken2.
2. Alice deposits 2 wei IToken2 into IToken1. 2 wei IToken2 is 1 wei TToken1 and 1 wei TToken2, totalVaultValue is 2, Alice get 2 shares.
3. Then Alice calls `IToken1.withdraw()` to withdraw 1 wei IToken2, 1 wei IToken2 is 0 wei TToken1(0.5 rounding down to 0) and 0 wei TToken2(0.5 rounding down to 0), totalValue is 0. The protocol will burn 0 shares, but transfer 1 wei IToken2 to Alice.
4. Then Alice calls `IToken1.mint()`, at this time `totalSupply() == 2`, but there is only 1 wei IToken2, `totalVaultValue` is 0. So no matter how many shares the user mints, no assets need to be provided.
5. Then Bob calls `IToken1.deposit()` to deposit assets, Bob will get `totalValue * totalSupply()` shares, which may fail due to overflow.

Recommendations:

1. It is recommended to return 0 when `totalVaultValue == 0` in `maxMint()`.
2. It is recommended to apply `checkMinShares()` check for IToken.

Theo: Resolved with @431f0680fd...

Zenith: Verified.

[M-6] `TToken.seize()` should not fail due to insufficient assets

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

Target

- TToken.sol#L372-L389

Description:

`TToken.seize()` allows admin to seize TTokens of blacklisted users, but the problem here is that it uses `maxRedeem()` as the limit on the shares to be seized.

```
function seize(address _from, address _to, uint256 _shares,
    string memory _reason) external onlyRole(DEFAULT_ADMIN_ROLE)
    onlyCanTransfer(_to) {
    if (_shares == 0) {
        revert ZeroAssetsOrShares();
    }
    if (_from == address(0) || _to == address(0)) {
        revert ErrorZeroAddress();
    }
    if (!whitelistContract.isBlacklisted(_from)) {
        revert SeizeNotAllowed(_from);
    }
    uint256 maxShares = maxRedeem(_from);
    if (_shares > maxShares) {
        revert ERC4626ExceededMaxRedeem(_from, _shares, maxShares);
    }
    _updateUnchecked(_from, _to, _shares);

    emit Seize(_from, _to, _shares, _reason);
}
```

Since the protocol supports pending deposits, that is, minting TTokens for users before assets are transferred to the protocol, which causes the actual assets held by the protocol to be less than `totalAssets()`, so in `maxWithdraw()/maxRedeem()`, the protocol will use the actual balance to limit the assets that users can withdraw

```
/** @dev See {ITToken-maxWithdraw} */
function maxWithdraw(address _owner)
    public view override(ERC4626Upgradeable, ITToken, IERC4626)
    returns (uint256) {
    uint256 contractAssets = IERC20(asset()).balanceOf(address(this));
    uint256 maxAssets = super.maxWithdraw(_owner);
    if (maxAssets > contractAssets) {
        return contractAssets;
    }
    return maxAssets;
}

/** @dev See {ITToken-maxRedeem} */
function maxRedeem(address _owner) public view override(ERC4626Upgradeable,
    ITToken, IERC4626) returns (uint256) {
    // convert max withdraw to shares
    uint256 maxAssets = maxWithdraw(_owner);
    return convertToShares(maxAssets);
}
```

However, `TToken.seize()` is different from withdrawals. It does not convert shares into assets to withdraw, so its limit should be the user's share balance rather than the asset balance.

Recommendations:

Change to

```
function seize(address _from, address _to, uint256 _shares,
    string memory _reason) external onlyRole(DEFAULT_ADMIN_ROLE)
    onlyCanTransfer(_to) {
    if (_shares == 0) {
        revert ZeroAssetsOrShares();
    }
    if (_from == address(0) || _to == address(0)) {
        revert ErrorZeroAddress();
    }
    if (!whitelistContract.isBlacklisted(_from)) {
        revert SeizeNotAllowed(_from);
    }
-   uint256 maxShares = maxRedeem(_from);
+   uint256 maxShares = balanceOf(_from);
    if (_shares > maxShares) {
        revert ERC4626ExceededMaxRedeem(_from, _shares, maxShares);
    }
    _updateUnchecked(_from, _to, _shares);

    emit Seize(_from, _to, _shares, _reason);
}
```

Theo: Resolved with @1495b38142...

Zenith: Verified.

[M-7] Attacker can exploit the divide-by-0 error in `_convertToAssets()` to cause DOS

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

Target

- ERC4626UpgradeableMultiAsset.sol#L347-L354

Description:

In `_convertToAssets()`, if `totalVaultValue == 0` or `vaultBalances[i]` == 0, `amounts[i]` is set to 0, otherwise divisions are performed to calculate `amounts[i]`.

```
function _convertToAssets(uint256 shares, Math.Rounding rounding)
    internal view virtual returns (address[] memory, uint256[] memory) {
    (address[] memory assets, uint256[] memory vaultBalances, uint256[]
    memory assetValues, uint256 totalVaultValue) = totalDepositAssets();
    if (assets.length == 0 || shares == 0) {
        return (assets, new uint256[](assets.length));
    }

    // convert shares to total value in underlying asset terms
    uint256 totalShareValue = shares.mulDiv(totalUnderlyingAssets() + 1,
    totalSupply() + 10 ** _decimalsOffset(), rounding);

    // distribute total value proportianlly based on current valut
    composition
    uint256[] memory amounts = new uint256[](assets.length);
    for (uint i = 0; i < assets.length; i++) {
        if (totalVaultValue == 0 || vaultBalances[i] == 0) {
            amounts[i] = 0;
        } else {
            uint256 assetValueFromShares
    = totalShareValue.mulDiv(assetValues[i], totalVaultValue, rounding);
            amounts[i] = assetValueFromShares.mulDiv(vaultBalances[i],
    assetValues[i], rounding);
        }
    }
    return (assets, amounts);
}
```

The calculation is divided by `assetValues[i]`, there is an implicit assumption that when `vaultBalances[i]` is not 0, `assetValues[i]` must not be 0. However, this is not always true, especially when the asset is an IToken, since it is composed of multiple assets, 1 share may get a value of 0 due to rounding down.

```
if (totalVaultValue == 0 || vaultBalances[i] == 0) {
    amounts[i] = 0;
} else {
    uint256 assetValueFromShares = totalShareValue.mulDiv(assetValues[i],
    totalVaultValue, rounding);
    amounts[i] = assetValueFromShares.mulDiv(vaultBalances[i],
    assetValues[i], rounding);
}
```

1. Consider IToken1 is composed of TToken1 and IToken2, and IToken2 is composed of TToken2 and TToken3.
2. Alice deposits 1 wei TToken1 and 1 wei IToken2 into IToken1. 1 wei TToken1 is 1 value, 1 wei IToken2 is 0 value (see below), totalValue is 1, and 1 share is minted.
3. Then Bob calls `IToken1.mint()`, and `previewMint()` calls `_convertToAssets()`.
4. In `_convertToAssets()`, vaultBalances[TToken1] == 1, assetValues[TToken1] == 1, and vaultBalances[IToken2] == 1, but in `IToken2.convertToAssets(1)`, due to rounding down, vaultBalances[TToken2] == 0(0.5 rounding down to 0) and vaultBalances[TToken3] == 0(0.5 rounding down to 0) , that is, assetValues[IToken2] == 0, which causes revert due to a division by 0 error when calculating `amounts[i]`.

Recommendations:

It is recommended to set amount[i] to 0 when assetValues[0] == 0.

```
function _convertToAssets(uint256 shares, Math.Rounding rounding)
    internal view virtual returns (address[] memory, uint256[] memory) {
    (address[] memory assets, uint256[] memory vaultBalances, uint256[]
    memory assetValues, uint256 totalVaultValue) = totalDepositAssets();
    if (assets.length == 0 || shares == 0) {
        return (assets, new uint256[](assets.length));
    }

    // convert shares to total value in underlying asset terms
    uint256 totalShareValue = shares.mulDiv(totalUnderlyingAssets() + 1,
    totalSupply() + 10 ** _decimalsOffset(), rounding);
    // distribute total value proportianlly based on current valut
    composition
    uint256[] memory amounts = new uint256[](assets.length);
    for (uint i = 0; i < assets.length; i++) {
      if (totalVaultValue == 0 || vaultBalances[i] == 0) {

    if (totalVaultValue == 0 || vaultBalances[i] == 0 || assetValues[i] ==
          0) {
          amounts[i] = 0;
       } else {
          uint256 assetValueFromShares
    = totalShareValue.mulDiv(assetValues[i], totalVaultValue, rounding);
          amounts[i] = assetValueFromShares.mulDiv(vaultBalances[i],
    assetValues[i], rounding);
       }
    }
    return (assets, amounts);
}
```

or omit assetValues[i] when calculating amount[i].

```
    function _convertToAssets(uint256 shares, Math.Rounding rounding)
    internal view virtual returns (address[] memory, uint256[] memory) {
        (address[] memory assets, uint256[] memory vaultBalances, uint256[]
    memory assetValues, uint256 totalVaultValue) = totalDepositAssets();
        if (assets.length == 0 || shares == 0) {
            return (assets, new uint256[](assets.length));
        }

        // convert shares to total value in underlying asset terms
        uint256 totalShareValue = shares.mulDiv(totalUnderlyingAssets() + 1,
    totalSupply() + 10 ** _decimalsOffset(), rounding);

        // distribute total value proportianlly based on current valut
    composition
        uint256[] memory amounts = new uint256[](assets.length);
        for (uint i = 0; i < assets.length; i++) {
            if (totalVaultValue == 0 || vaultBalances[i] == 0) {
                amounts[i] = 0;
            } else {
              uint256 assetValueFromShares = totalShareValue.mulDiv(
                  assetValues[i], totalVaultValue, rounding);
              amounts[i] = assetValueFromShares.mulDiv(vaultBalances[i],
                    assetValues[i], rounding);
                amounts[i] = totalShareValue.mulDiv(vaultBalances[i],
                    totalVaultValue, rounding);
            }
        }
        return (assets, amounts);
    }
```

Theo: Resolved with @054a95e911...

Zenith: Verified.

[M-8] `IToken.checkAssetList()` should require that the TToken's `escrowAsset` must be `asset()`

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

Target

- ITToken.sol#L76-L82

Description:

When configuring `depositAssets` of IToken, `checkAssetList()` will check `depositAssets`. If `depositAsset` is `TToken`, it requires that the `asset` **or** `escrowAsset` of TToken must be the `asset` of IToken.

```
modifier checkAssetList(address[] calldata assetList) {
    // allow for initialize to be setup before checking list so "asset()" is
    set
    _;
    for (uint256 i = 0; i < assetList.length; i++) {
        address depositAsset = assetList[i];
        if (depositAsset == address(0) || depositAsset == address(this)) {
            revert ErrorInvalidDepositAsset(depositAsset);
        }
        if (ERC165Checker.supportsInterface(depositAsset,
    type(IIToken).interfaceId)) {
            // if IToken, ensure underlying asset is the same
            if (IIToken(depositAsset).asset() != asset()) {
                revert ErrorInvalidDepositAsset(depositAsset);
            }
        } else if (ERC165Checker.supportsInterface(depositAsset,
    type(ITToken).interfaceId)) {
            // if TToken, ensure asset or escrow asset is the same as this
    contract's asset
            ITToken.TTokenParams memory tTokenParams
    = ITToken(depositAsset).getTTokenParams();
            if (tTokenParams.asset != asset() && tTokenParams.escrowAsset !=
    asset()) {
                revert ErrorInvalidDepositAsset(depositAsset);
            }
        } else {
            // doesn't support either interface
            revert ErrorInvalidDepositAsset(depositAsset);
        }
    }
}
```

The problem here is that when getting the amount of underlying assets corresponding to depositAsset in `_convertDepositAssetToUnderlying()`, if depositAsset is TToken, the protocol will call `TToken.sharesToEscrowAssets()` to get the amount of `escrowAsset` corresponding to depositAsset. Once the `escrowAsset` of TToken is not the `asset` of IToken, this will cause `_convertDepositAssetToUnderlying()` to return the wrong amount of tokens, resulting in protocol accounting errors.

```
function _convertDepositAssetToUnderlying(address depositAsset,
    uint256 amount) internal view override returns (uint256) {
    if (ERC165Checker.supportsInterface(depositAsset,
    type(IIToken).interfaceId)) {
        // if IToken, convert to underlying asset
        return IIToken(depositAsset).convertToAssets(amount);
    }
    if (ERC165Checker.supportsInterface(depositAsset,
    type(ITToken).interfaceId)) {
        // if TToken, convert to escrow asset
        return ITToken(depositAsset).sharesToEscrowAssets(amount);
    }
    // revert since deposit asset is not a valid TToken or IToken
    revert ErrorInvalidDepositAsset(depositAsset);
}
```

Recommendations:

It is recommended to require that the TToken's `escrowAsset` must be `asset()` in `checkAssetList()`.

```
modifier checkAssetList(address[] calldata assetList) {
    // allow for initialize to be setup before checking list so "asset()" is
    set
    _;
    for (uint256 i = 0; i < assetList.length; i++) {
        address depositAsset = assetList[i];
        if (depositAsset == address(0) || depositAsset == address(this)) {
            revert ErrorInvalidDepositAsset(depositAsset);
        }
```

```
            if (ERC165Checker.supportsInterface(depositAsset,
        type(IIToken).interfaceId)) {
                // if IToken, ensure underlying asset is the same
                if (IIToken(depositAsset).asset() ≠ asset()) {
                    revert ErrorInvalidDepositAsset(depositAsset);
                }
            } else if (ERC165Checker.supportsInterface(depositAsset,
        type(ITToken).interfaceId)) {
                // if TToken, ensure asset or escrow asset is the same as this
        contract's asset
                ITToken.TTokenParams memory tTokenParams
        = ITToken(depositAsset).getTTokenParams();
-               if (tTokenParams.asset ≠ asset() && tTokenParams.escrowAsset ≠
                    asset()) {
+               if (tTokenParams.escrowAsset ≠ asset()) {
                    revert ErrorInvalidDepositAsset(depositAsset);
                }
            } else {
                // doesn't support either interface
                revert ErrorInvalidDepositAsset(depositAsset);
            }
        }
    }
```

Theo: Resolved with @4e310790cf...

Zenith: Verified.

4.2 Low Risk

A total of 3 low risk findings were identified.

[L-1] Issues with `updateDepositAssets`

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

Target

- ITоken.sol#L121-L136

Description:

There are several issues with `updateDepositAssets`. First, previously deposited assets are changed without being withdrawn or rescued, causing them to become stuck inside the `IToken`. Second, if `updateDepositAssets` is called and `totalValueAfter` is significantly larger than `totalValueBefore`, the operation becomes prone to a sandwich attack.

```
function updateDepositAssets(address[] calldata newAssets,
    ITokenParams calldata newConfig) external onlyRole(DEFAULT_ADMIN_ROLE)
    checkAssetList(newAssets) {
    _updateDepositAssets(newAssets);
    _setConfig(newConfig);
}

function _updateDepositAssets(address[] calldata newAssets)
    internal override {
    // get total value before update
    uint256 totalValueBefore = totalUnderlyingAssets();
    super._updateDepositAssets(newAssets);
    // get total value after update
    uint256 totalValueAfter = totalUnderlyingAssets();

    if (totalValueAfter < totalValueBefore) {
        revert ErrorDepositAssetsValueWentDown();
    }
}
```

Recommendations:

Consider rescuing the previous assets and pausing all deposit and withdraw operations before performing the `updateDepositAssets` operation.

Theo: Resolved with @4a54367cc6...

Zenith: Verified.

[L-2] ERC4626UpgradeableMultiAsset's `_depositAssets` should not be duplicated

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

Target

- ERC4626UpgradeableMultiAsset.sol#L145-L174

Description:

If the `depositAssets` of the ERC4626UpgradeableMultiAsset contains duplicate assets, in `totalUnderlyingAssets()` and `totalDepositAssets()`, since it uses the token balance directly to calculate the total value of those assets, this could result in the asset values being double counted.

```
function totalUnderlyingAssets() public view virtual returns (uint256) {
    ERC4626MultiAssetStorage storage $ = _getERC4626MultiAssetStorage();
    uint256 totalUnderlying = 0;
    for (uint256 i = 0; i < $._depositAssets.length; i++) {
        address assetAddress = $._depositAssets[i];
        IERC20 assetContract = IERC20(assetAddress);
        totalUnderlying += _convertDepositAssetToUnderlying(assetAddress,
    assetContract.balanceOf(address(this)));
    }
    return totalUnderlying;
}

/** @dev See {IERC4626MultiAsset-totalDepositAssets}. */
function totalDepositAssets() public view virtual returns (address[] memory,
    uint256[] memory, uint256[] memory, uint256) {
    ERC4626MultiAssetStorage storage $ = _getERC4626MultiAssetStorage();
    address[] memory depositAssets = $._depositAssets;
    uint256[] memory amounts = new uint256[](depositAssets.length);
    uint256[] memory values = new uint256[](depositAssets.length);
    uint256 totalUnderlyingValue = 0;

    for (uint256 i = 0; i < depositAssets.length; i++) {
        address assetAddress = depositAssets[i];
        IERC20 assetContract = IERC20(assetAddress);
```

```
            uint256 assetBalance = assetContract.balanceOf(address(this));
            amounts[i] = assetBalance;
            uint256 underlyingValue
        = _convertDepositAssetToUnderlying(assetAddress, assetBalance);
            values[i] = underlyingValue;
            totalUnderlyingValue += underlyingValue;
        }
        return (depositAssets, amounts, values, totalUnderlyingValue);
    }
```

Recommendations:

It is recommended to require ERC4626UpgradeableMultiAsset's `_depositAssets` to be non-duplicate.

Theo: Resolved with @bdf3171945...

Zenith: Verified.

[L-3] `ERC4626UpgradeableMultiAsset.withdraw()` should require shares > 0

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

Target

- ERC4626UpgradeableMultiAsset.sol#L295

Description:

`ERC4626UpgradeableMultiAsset.withdraw()` does not require shares > 0, which may cause users to burn 0 shares to withdraw 1 wei assets.

1. Consider IToken1 is composed of IToken2, and IToken2 is composed of TToken1 and TToken2.
2. Alice calls `IToken1.withdraw()` to withdraw 1 wei IToken2, 1 wei IToken2 is 0 wei TToken1(0.5 rounding down to 0) and 0 wei TToken2(0.5 rounding down to 0) in `IToken2.convertToAssets()`, totalValue is 0. The protocol will burn 0 shares, but transfer 1 wei IToken2 to Alice.

```
function convertToAssets(uint256 shares)
    public view virtual returns (uint256) {
    // convert to deposit assets, then get total value in underlying asset
    terms
    (address[] memory assets, uint256[] memory amounts)
    = _convertToAssets(shares, Math.Rounding.Floor); // ← rounding down
    return _getAssetListValue(assets, amounts);
}
```

Recommendations:

```
function withdraw(address[] calldata withdrawAssets, uint256[]
    calldata assetAmounts, address receiver, address owner)
    public virtual returns (uint256) {
    if (withdrawAssets.length ≠ assetAmounts.length) {
        revert ERC4626MultiAssetArrayMismatch();
    }
    _checkArrayDuplicates(withdrawAssets);

    for (uint i = 0; i < withdrawAssets.length; i++) {
        if (!isSupportedDepositAsset(withdrawAssets[i])) {
            revert
    ERC4626MultiAssetUnsupportedDepositAsset(withdrawAssets[i]);
        }
        uint256 maxAssets
    = IERC20(withdrawAssets[i]).balanceOf(address(this));
        if (assetAmounts[i] > maxAssets) {
            revert ERC4626MultiAssetExceededMaxWithdraw(owner,
    withdrawAssets[i], assetAmounts[i], maxAssets);
        }
    }

    // convert assets to shares
    uint256 shares = previewWithdraw(withdrawAssets, assetAmounts);
   require(shares > 0);
```

Theo: Resolved with @431f0680fd...

Zenith: Verified.

4.3 Informational

A total of 2 informational findings were identified.

[I-1] `TTokenRouter` could potentially accumulate assets due to `bulkCompletePending`

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

Target

- TTokenRouter.sol#L36-L45

Description:

When `bulkCompletePending` is called, it processes each `CompletePending` request by pulling `complete.amountAsset` from the sender and calling `tToken.completePending` to complete the operation.

```
    function _completePending(CompletePending calldata complete) internal {

        // ensure asset is correct
        if (ITToken(complete.tToken).asset() ≠ complete.asset) {
            revert InvalidTTokenAsset(complete.asset,
    ITToken(complete.tToken).asset());
        }
        // transfer asset to this contract
        IERC20(complete.asset).safeTransferFrom(msg.sender, address(this),
    complete.amountAsset);

        IERC20(complete.asset).safeIncreaseAllowance(complete.tToken,
    complete.amountAsset);
>>>     ITToken(complete.tToken).completePending(complete.amountAsset);
    }
```

However, `completePending` only processes up to `totalAssetsPending` of the assets if `_assets` is greater than `totalAssetsPending`.

```
    function completePending(uint256 _assets) external onlyRole(MINTER_ROLE)
    {
        if (_assets > totalAssetsPending) {
>>>         _assets = totalAssetsPending;
        }
        SafeERC20.safeTransferFrom(IERC20(asset()), msg.sender,
    address(this), _assets);
        // update total assets
        totalAssetsPending -= _assets;
        emit CompletePending(msg.sender, _assets);
    }
```

The assets can be rescued through `recoverFunds`, but this issue demonstrates that one of the stated invariants can be broken: "any function called should not accumulate assets in this contract".

Recommendations:

Consider to send back the leftover assets from `bulkCompletePending` to the sender.

Theo: Resolved with @114cf1ccc16...

Zenith: Verified.

[I-2] Lack of a limit on deposit assets could cause an OOG issue

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

Target

- ITToken.sol#L63-L87

Description:

The number of `_depositAssets` of `IToken` currently has no limit. This could cause an out-of-gas (OOG) issue, as several operations loop through the `_depositAssets` list.

For instance, functions such as `totalDepositAssets`, `totalUnderlyingAssets`, and `_convertToAssets` iterate through the assets to calculate their values.

```
    function totalDepositAssets() public view virtual returns (address[]
    memory, uint256[] memory, uint256[] memory, uint256) {
        ERC4626MultiAssetStorage storage $ = _getERC4626MultiAssetStorage();
        address[] memory depositAssets = $._depositAssets;
        uint256[] memory amounts = new uint256[](depositAssets.length);
        uint256[] memory values = new uint256[](depositAssets.length);
        uint256 totalUnderlyingValue = 0;

        for (uint256 i = 0; i < depositAssets.length; i++) {
            address assetAddress = depositAssets[i];
            IERC20 assetContract = IERC20(assetAddress);
            uint256 assetBalance = assetContract.balanceOf(address(this));
            amounts[i] = assetBalance;
            uint256 underlyingValue
    = _convertDepositAssetToUnderlying(assetAddress, assetBalance);
            values[i] = underlyingValue;
            totalUnderlyingValue += underlyingValue;
        }
        return (depositAssets, amounts, values, totalUnderlyingValue);
    }

    function totalUnderlyingAssets() public view virtual returns (uint256) {
        ERC4626MultiAssetStorage storage $ = _getERC4626MultiAssetStorage();
        uint256 totalUnderlying = 0;
        for (uint256 i = 0; i < $._depositAssets.length; i++) {
            address assetAddress = $._depositAssets[i];
            IERC20 assetContract = IERC20(assetAddress);
            totalUnderlying
    += _convertDepositAssetToUnderlying(assetAddress,
    assetContract.balanceOf(address(this)));
        }
        return totalUnderlying;
    }

    function _convertToAssets(uint256 shares, Math.Rounding rounding)
    internal view virtual returns (address[] memory, uint256[] memory) {
>>>     (address[] memory assets, uint256[] memory vaultBalances, uint256[]
    memory assetValues, uint256 totalVaultValue) = totalDepositAssets();
        if (assets.length == 0 || shares == 0) {
            return (assets, new uint256[](assets.length));
        }
        // convert shares to total value in underlying asset terms
>>>     uint256 totalShareValue = shares.mulDiv(totalUnderlyingAssets() + 1,
    totalSupply() + 10 ** _decimalsOffset(), rounding);

        // distribute total value proportianlly based on current valut
    composition
        uint256[] memory amounts = new uint256[](assets.length);
        for (uint i = 0; i < assets.length; i++) {
            if (totalVaultValue == 0 || vaultBalances[i] == 0) {
                amounts[i] = 0;
            } else {
                uint256 assetValueFromShares
    = totalShareValue.mulDiv(assetValues[i], totalVaultValue, rounding);
                amounts[i] = assetValueFromShares.mulDiv(vaultBalances[i],
    assetValues[i], rounding);
            }
        }
        return (assets, amounts);
    }
```

This is especially concerning for `_convertToAssets`, which also calls `totalDepositAssets` and `totalUnderlyingAssets` during execution. Additionally, since an `IToken` can contain another `IToken`, the likelihood of an OOG issue increases.

```
    function _convertDepositAssetToUnderlying(address depositAsset,
    uint256 amount) internal view override returns (uint256) {
        if (ERC165Checker.supportsInterface(depositAsset,
    type(IIToken).interfaceId)) {
            // if IToken, convert to underlying asset
>>>         return IIToken(depositAsset).convertToAssets(amount);
        }
        if (ERC165Checker.supportsInterface(depositAsset,
    type(ITToken).interfaceId)) {
            // if TToken, convert to escrow asset
            return ITToken(depositAsset).sharesToEscrowAssets(amount);
        }
        // revert since deposit asset is not a valid TToken or IToken
        revert ErrorInvalidDepositAsset(depositAsset);
    }
```

Recommendations:

Consider to limit the number of `_depositAssets`.

Theo: Acknowledged.