Polygon

Yield Exposed Token

Security Assessment Report

Version: 2.0

April, 2025

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Polygon's Yield Exposed Token components. The review focused solely on the security aspects of the Solidity implementation, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the components in scope. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Polygon components contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Polygon components in scope.

Overview

Yield Exposed Tokens are an innovation on the Polygon ZkEVM that allow for expanded DeFi opportunities on its layer Y's.

Tokens on layer X are deposited into yield farms and earn yield. This yield is then transmitted to layer Y DeFi protocols who then distribute it as additional rewards to their users. To be eligible for this extra yield, users need to use the special "YE" (Yield Exposed) tokens that are minted by depositing the original assets into Polygon's `YieldExposedToken` layer X contract.

The system also allows for asset tokens to be deposited directly on layer Y for YE tokens, in which case the system later migrates the asset tokens back to layer X for deposit into the yield vaults.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the Polygon Yield Exposed Token repository.

The scope of this time-boxed review was strictly limited to files at commit d282975.

Note: third party libraries and dependencies were excluded from the scope of this assessment.

Approach

The security assessment covered components written in Solidity.

For the Solidity components, the manual review focused on identifying issues associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout).

Additionally, the manual review process focused on identifying vulnerabilities related to known Solidity anti-patterns and attack vectors, such as re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers.

For a more detailed, but non-exhaustive list of examined vectors, see [1, 2].

To support the Solidity components of the review, the testing team also utilised the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Aderyn: `https://github.com/Cyfrin/aderyn`

Output for these automated tools is available upon request.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 14 issues during this assessment. Categorised by their severity:

- Critical: 1 issue.

- High: 1 issue.
- Medium: 4 issues.
- Low: 4 issues.
- Informational: 4 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Polygon components in scope. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
YE-01	Cross-Chain Call Execution Failure Due to Data Layout Mismatch	Critical	Resolved
YE-02	Reentrancy During Token Transfer Mints Extra Tokens	High	Resolved
YE-03	Reserve Replenishment With Fee Paying Tokens Could Consume Yield	Medium	Resolved
YE-04	Unnecessary Vault Deposits During Deposits And Migration	Medium	Resolved
YE-05	All Layer Y's Are Trusted For Valid Execution	Medium	Resolved
YE-06	Reentrancy During Yield Vault Deposit Can Cause Accounting Errors	Medium	Resolved
YE-07	Claim And Withdraw Process Claims Tokens To The `YieldExposedToken` Contract	Low	Resolved
YE-08	Unnecessary Refund During Deposit	Low	Closed
YE-09	Accounting Errors In Collected Yield Can Be Produced By Burning Tokens From Other Sources	Low	Resolved
YE-10	Any Token Can Be Claimed From The Bridge	Low	Resolved
YE-11	Insufficient Value Check in Function Logic	Informational	Resolved
YE-12	Inconsistent Use Of Event	Informational	Closed
YE-13	Event ReserveRebalanced() Does Not Indicate A Rebalanced Reserve	Informational	Resolved
YE-14	Miscellaneous General Comments	Informational	Resolved

YE-01	Cross-Chain Call Execution Failure Due to Data Layout Mismatch		
Asset	`NativeConverter.sol, YieldExposedToken.sol, WETHNativeConverter.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

The encoding and decoding logic for the cross-chain message data is mismatched. As a result, the native converter's migration process cannot be fully executed on layer X.

In the function `NativeConverter.migrateBackingToLayerX()`, the data sent through the bridge is encoded as follows:

```
$.lxlyBridge.bridgeMessage(
    $.layerXLxlyId, $.yeToken, true, abi.encode(CrossNetworkInstruction.COMPLETE_MIGRATION, shares, assets));
```

However, in the function `YieldExposedToken.onMessageReceived()`, the data is decoded using the following:

```
(CrossNetworkInstruction instruction, bytes memory instructionData) = abi.decode(data, (CrossNetworkInstruction, bytes));
```

As a result, a call from the bridge to `onMessageReceived()` function will revert due to this data layout mismatch.

Consequently, migration of underlying tokens from layer Y's to layer X would not be fully completed. This means that the migrated underlying assets would not be deposited into the yield generating vaults, resulting in a loss of generated revenue.

The process of minting `yeTokens` and migrating them to the layer Y's zero address would also not take place, meaning that the Agglayer accounting would be distorted, blocking bridging of these tokens.

Note, this issue is also present in `WETHNativeConverter.migrateGasBackingToLayerX()`:

```
// Bridge a message to Migration Manager on Layer X to complete the migration.
lxlyBridge().bridgeMessage(
    layerXLxlyId(),
    address(yeToken()),
    true,
    abi.encode(
        CrossNetworkInstruction.CUSTOM,
        CustomCrossNetworkInstruction.WRAP_COIN_AND_COMPLETE_MIGRATION,
        amountOfCustomToken,
        amount
    )
);
```

Recommendations

Use the same data layout for encoding and decoding the bridged message data.

Resolution

This issue was resolved in commits 8ee6ff3, d64f31b and 083c64e.

YE-02	Reentrancy During Token Transfer Mints Extra Tokens		
Asset	`YieldExposedToken.sol, NativeConverter.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

If an underlying token permits the token owner to run arbitrary code, it could be exploited to inflate the number of tokens the contract thinks it has received during deposits and conversions.

The issue originates from the function `_receiveUnderlyingToken()`, which has similar logic in both `NativeConverter` and `YieldExposedToken`:

```
// Cache the balance.
uint256 balanceBefore = $.underlyingToken.balanceOf(address(this));

// Transfer.
$.underlyingToken.safeTransferFrom(from, address(this), value);

// Calculate the received amount.
receivedValue = $.underlyingToken.balanceOf(address(this)) - balanceBefore;
```

In the middle step `safeTransferFrom()`, any action that would increase the underlying token balance of the contract will be interpreted as being part of the deposit that is currently being processed.

Malicious actors could exploit this as follows:

1. During a call to `NativeConverter.convert()`, make another call to `NativeConverter.convert()`.
2. During a call to `YieldExposedToken.deposit()`, call the LXLY bridge's function `claimAsset()` to process a migration from a native converter.

In each case, the value `receivedValue` in the last step of the quoted code will be inflated, returning value of the `_receiveUnderlyingToken()` function and resulting in more custom tokens or `yeTokens` being minted to the caller.

Note, likelihood of this occurring is reduced as successful exploitation requires code execution during token transfer, which is a feature that most well known and established token assets do not have.

Recommendations

Add a reentrancy guard to all deposit, withdraw, convert and deconvert functions.

Resolution

This issue was resolved in commit 9b72563.

YE-03	Reserve Replenishment With Fee Paying Tokens Could Consume Yield		
Asset	`YieldExposedToken.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

Whenever a fee paying underlying token is deposited into or withdrawn from a yield vault, the token will charge a fee. It is therefore desirable, for these tokens, to minimise the number of transfers they make as all shortfalls have to be made up from yield.

The function `replenishReserve()` can be called by any user to withdraw tokens from the yield vault whenever the reserve falls below its minimum level. `rebalanceReserve()` works in the opposite direction, but is only called by the owner.

However, if a token experiences a large amount of deposits and withdrawals into the `YieldExposedToken` contract, and this causes the reserve levels to fluctuate a lot, this could create an unnecessarily large number of withdrawals from the yield vault, possibly outstripping the earned yield if the levels are high enough.

Recommendations

Consider replenishing beyond the minimum reserve level, ideally in a function controlled by the owner, so that it is not possible to cause excessive transfers of the underlying token with calls to `replenishReserve()`.

Resolution

This issue was resolved in commit 688487c.

YE-04	Unnecessary Vault Deposits During Deposits And Migration		
Asset	`YieldExposedToken.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

When calculating the amount of assets to deposit into the yield vault during a deposit or migration, the amount received is simply split in the ratio of the desired reserve to deposit ratio. This calculation does not take account of the current state of the reserve. In the case where the reserve is low, it might be that it can be replenished by keeping a higher proportion of the received amount. In the case of fee paying tokens, the unnecessary extra deposits and withdrawals to and from the yield vault will cost extra fees, depleting the protocol's yield.

For example, consider a scenario where the USDT reserve rate is 10%, and the total supply is 10 000 USDT, resulting in a reserve size of 1 000 USDT. If the reserve is empty and a deposit of 1 000 USDT is received, the current implementation would reserve 100 USDT and deposit 900 into the yield vault. If this is followed by a call to `replenishReserve()`, this 900 USDT would then immediately be withdrawn from the yield vault.

Note the relevant code from the `_completeMigration()` function:

```
// Calculate the amount to reserve and the amount to deposit into the yield vault.
uint256 assetsToReserve = Math.mulDiv(requiredAssets, $.minimumReservePercentage, 1e18);
assetsToReserve = assetsToReserve > assets ? assets : assetsToReserve;

// Calculate the amount to deposit into the yield vault.
uint256 assetsToDeposit = assets - assetsToReserve;
```

And the equivalent section in `_deposit()`:

```
// Calculate the amount to reserve.
uint256 assetsToReserve = Math.mulDiv(assets, $.minimumReservePercentage, 1e18);

// Calculate the amount to deposit into the yield vault.
uint256 assetsToDeposit = assets - assetsToReserve;
// @note Yield vault usage.
uint256 maxDeposit_ = $.yieldVault.maxDeposit(address(this));
assetsToDeposit = assetsToDeposit > maxDeposit_ ? maxDeposit_ : assetsToDeposit;

// Cache the balance.
uint256 balanceBefore = $.underlyingToken.balanceOf(address(this));

// Deposit into the yield vault.
if (assetsToDeposit > 0) {
    $.yieldVault.deposit(assetsToDeposit, address(this));
}
```

Recommendations

Check the current status of the reserve when new underlying tokens are received, and keep more tokens as necessary to replenish the reserve.

Resolution

This issue was resolved in commit 53fb988.

YE-05	All Layer Y's Are Trusted For Valid Execution		
Asset	`YieldExposedToken.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The Agglayer system does not guarantee valid execution of all layer Y's, it only guarantees that a layer Y cannot transfer more tokens to another layer than it has secured on the LXLY bridge.

However, certain parts of this system place a greater degree of trust in all layer Y's:

1. `YieldExposedToken.onMessageReceived()` trusts the `originAddress` on the layer Y is a valid native converter.
2. `YieldExposedToken._completeMigration()` bridges to the zero address on any layer Y, declaring that the bridged yeTokens will not be claimable.

After discussion with the development team, it was confirmed that not all layer Y's would have full execution proofs at all times. Therefore it may not be desirable for the system to equally trust all layer Y's to always execute a correct state transition. For example, those with full execution proofs may want to be isolated from the risk of trusting messages bridged from those without.

Recommendations

Consider whitelisting the layer Y's that can use the yield exposed token system to allow tighter control of risk assumptions.

Resolution

This issue was resolved by adding a mapping of layer Ys to native converter addresses in commit 3221f98.

YE-06	Reentrancy During Yield Vault Deposit Can Cause Accounting Errors		
Asset	`YieldExposedToken.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

If a deposit into a yield vault permits arbitrary code execution, it might be exploited to inflate the number of tokens the contract assumes it has expended during a yield vault deposit. This could skew the `$.reservedAssets` value and lead to accounting discrepancies.

The issue originates from the function `_deposit()`, when it deposits into the yield vault:

```
// Cache the balance.
uint256 balanceBefore = $.underlyingToken.balanceOf(address(this));

// Deposit into the yield vault.
if (assetsToDeposit > 0) {
    $.yieldVault.deposit(assetsToDeposit, address(this));
}

// Recalculate the amount to reserve.
assetsToReserve = assets - (balanceBefore - $.underlyingToken.balanceOf(address(this)));

// Update the reserve.
$.reservedAssets += assetsToReserve;
```

In the middle step, any action that would increase the underlying token balance of the contract will ultimately increase the value of `$.reservedAssets`. This could be achieved via a call to `YieldExposedToken.deposit()`, or a call to the LXLY bridge's function `claimAsset()` to process a migration from a native converter.

Note, the accounting distortion is limited to the value of `assets`, as larger distortions would cause overflow errors.

An inaccuracy in `$.reservedAssets` impacts the amount of tokens held in reserve, the quantity deposited into the yield vault, and other accounting matters, potentially resulting in function reverts.

Recommendations

Add a reentrancy guard to all deposit functions.

Resolution

This issue was resolved in commit 9b72563.

YE-07	Claim And Withdraw Process Claims Tokens To The `YieldExposedToken` Contract		
Asset	`YieldExposedToken.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

When claiming and withdrawing from the LXLY bridge via the `claimAndWithdraw()` function, the tokens are withdrawn directly to the `YieldExposedToken` contract, followed by a withdrawal that burns tokens from `msg.sender`. This process undermines the function's primary purpose, rendering it largely ineffective.

```
function claimAndWithdraw(
    bytes32[32] calldata smtProofLocalExitRoot,
    bytes32[32] calldata smtProofRollupExitRoot,
    uint256 globalIndex,
    bytes32 mainnetExitRoot,
    bytes32 rollupExitRoot,
    uint32 originNetwork,
    address originTokenAddress,
    uint32 destinationNetwork,
    address destinationAddress,
    uint256 amount,
    bytes calldata metadata,
    address receiver
) external whenNotPaused returns (uint256 shares) {
    // Claim yeToken from LxLy Bridge.
    lxlyBridge().claimAsset(
        smtProofLocalExitRoot,
        smtProofRollupExitRoot,
        globalIndex,
        mainnetExitRoot,
        rollupExitRoot,
        originNetwork,
        originTokenAddress,
        destinationNetwork,
        address(this),
        amount,
        metadata
    );

    // Withdraw the underlying token to the receiver.
    return _withdraw(amount, receiver, destinationAddress);
}
```

And then within the `_withdraw()` function:

```
// Check the input.
if (msg.sender != owner) _spendAllowance(owner, msg.sender, shares);

// The amount that cannot be withdrawn at the moment.
uint256 remainingAssets = assets;

// Calculate the amount to withdraw from the reserve.
uint256 amountToWithdraw = $.reservedAssets > remainingAssets ? remainingAssets : $.reservedAssets;

// Withdraw the underlying token from the reserve.
if (amountToWithdraw > 0) {
    // Burn yeToken.
    _burn(owner, convertToShares(amountToWithdraw));
```

Despite this flaw, the core functionality remains intact - users can still claim `yeTokens` directly from the LXLY bridge to their address and subsequently withdraw them to the underlying asset, albeit through two separate transactions rather than a streamlined single step.

Recommendations

Modify the `claimAndWithdraw()` function to claim to and withdraw from the same address.

Resolution

This issue was resolved in commit b9321a3.

YE-08	Unnecessary Refund During Deposit		
Asset	`YieldExposedToken.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

In the deposit process, the maximum required asset amount is determined. If the transferred assets exceed this limit, the excess is returned to the sender. However, this calculation could occur prior to the initial transfer, allowing the transfer amount to be adjusted downward if needed.

This approach would eliminate the gas expense of an additional refund and, for fee-bearing tokens, prevent yield losses due to extra fees from the surplus transfer.

```
// Transfer the underlying token from the sender to self.
assets = _receiveUnderlyingToken(msg.sender, assets);

// Check for a refund.
if (maxShares > 0) {
    // Calculate the required amount of the underlying token.
    uint256 requiredAssets = convertToAssets(maxShares);

    if (assets > requiredAssets) {
        // Calculate the difference.
        uint256 refund = assets - requiredAssets;

        // Refund the difference.
        _sendUnderlyingToken(msg.sender, refund);

        // Update the `assets`.
        assets = requiredAssets;
    }
}
```

This issue is partially mitigated by the UI interface of the system, which will presumably attempt to provide inputs that avoid these refunds.

Recommendations

Consider calculating the maximum required assets before the initial transfer and then transfer in this amount.

Resolution

The development team reviewed the issue and considered that no action was required at this time.

YE-09	Accounting Errors In Collected Yield Can Be Produced By Burning Tokens From Other Sources		
Asset	`YieldExposedToken.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `burn()` function requires that the caller be the yield recipient, under the assumption that only tokens generated from yield would ever be burned. It is possible, however, for the yield recipient to receive tokens from an origin other than yield, and then to burn them. This would artificially reduce the value of `$.totalCollectedYield`.

```
function burn(uint256 shares) external {
    YieldExposedTokenStorage storage $ = _getYieldExposedTokenStorage();

    // Check the input.
    require(msg.sender == $.yieldRecipient, Unauthorized());
    require(shares > 0, InvalidShares());

    // Update the total collected yield.
    $.totalCollectedYield -= shares;

    // Burn yeToken.
    _burn(msg.sender, shares);
```

The main effect of this is to reduce the value returned by `totalCollectedYield()`, giving a misleading picture of how the system has performed.

Given that the yield recipient is a trusted role, this seems unlikely to occur. It is also a very expensive form of attack with a low impact.

Recommendations

If output of `totalCollectedYield()` is considered a vital function, prevent transfers to the yield recipient address other than from the `YieldExposedToken` contract.

Resolution

This issue was resolved in commit cd60520.

YE-10	Any Token Can Be Claimed From The Bridge		
Asset	`YieldExposedToken.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

When claiming and withdrawing from the LXLY bridge via the function `claimAndWithdraw()`, it is possible to enter any token as the token to be claimed.

This token will then be claimed on to the `YieldExposedToken` contract and the withdrawal process would proceed as normal, as this process does not depend on the token address argument.

```
function claimAndWithdraw(
    bytes32[32] calldata smtProofLocalExitRoot,
    bytes32[32] calldata smtProofRollupExitRoot,
    uint256 globalIndex,
    bytes32 mainnetExitRoot,
    bytes32 rollupExitRoot,
    uint32 originNetwork,
    address originTokenAddress,
    uint32 destinationNetwork,
    address destinationAddress,
    uint256 amount,
    bytes calldata metadata,
    address receiver
) external whenNotPaused returns (uint256 shares) {
    // Claim yeToken from LxLy Bridge.
    lxlyBridge().claimAsset(
        smtProofLocalExitRoot,
        smtProofRollupExitRoot,
        globalIndex,
        mainnetExitRoot,
        rollupExitRoot,
        originNetwork,
        originTokenAddress,
        destinationNetwork,
        address(this),
        amount,
        metadata
    );

    // Withdraw the underlying token to the receiver.
    return _withdraw(amount, receiver, destinationAddress);
}
```

This is an issue with limited impact. It requires an asset to have been bridged to the `YieldExposedToken` contract. That asset can then be claimed on to the `YieldExposedToken` contract, after which a withdrawal for the same amount would need to be validly processed by the caller to avoid the transaction reverting.

Recommendations

Validate the `originTokenAddress` argument, or remove the parameter and use a state variable to supply the token address.

Resolution

This issue was resolved in commit b9321a3.

YE-11	Insufficient Value Check in Function Logic
Asset	`YeETH.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The internal function `_receiveUnderlyingToken()` could behave erratically if inherited by another contract or used in an upgraded version of `YeETH`.

The potential issue is due to the logic of the function, which does not account for a scenario where `msg.value` is greater than zero but less than `assets`.

```
if (msg.value >= assets) {
    // deposit everything, excess funds will be refunded in WETH
    weth.deposit{value: msg.value}();
} else {
    weth.safeTransferFrom(msg.sender, address(this), assets);
}
return assets;
```

If that were the case, the function would not deposit the received ETH into WETH, but would still transfer WETH from the `msg.sender`.

However, in the current implementation, this internal function is not used in any scenario where that could happen: either the functions calling it are not `payable`, or they use `msg.value` as the value for `assets`. Therefore, this is only a potential issue in future code.

Recommendations

Consider modifying the `if` test to check whether any ETH value is present at all, for example:

```
if (msg.value > 0) {
    // deposit everything, excess funds will be refunded in WETH
    weth.deposit{value: msg.value}();
    return msg.value;
} else {
    weth.safeTransferFrom(msg.sender, address(this), assets);
    return assets;
}
```

Resolution

The suggested modifications were implemented in commit 2d14cc1.

YE-12	Inconsistent Use Of Event
Asset	`NativeConverter.sol, WETHNativeConverter.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The event `AssetsTooLarge()` is used to refer to two different types of balance, and this may be misleading, especially to automated systems monitoring this event.

In `NativeConverter.migrateBackingToLayerX()`, the error refers to a lack of underlying tokens.

```
require(assets <= $.backingOnLayerY, AssetsTooLarge($.backingOnLayerY, assets));
```

In `WETHNativeConverter.migrateGasBackingToLayerX()`, the same event refers to the ETH balance of the `zETH` contract, a related but distinct asset store.

```
require(amount <= address(zETH).balance, AssetsTooLarge(address(zETH).balance, amount));
```

Recommendations

Consider creating a new event for `migrateGasBackingToLayerX()`, such as `GasAssetsTooLarge()`.

Resolution

The development team reviewed the issue and considered that no action was required at this time.

YE-13	Event ReserveRebalanced() Does Not Indicate A Rebalanced Reserve
Asset	`YieldExposedToken.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The event `ReserveRebalanced()` is emitted whenever an adjustment is made to the reserve, no matter how small. This may be misleading.

The event is emitted in the following code so long as the `assetsToWithdraw` value is greater than zero:

```
if ($.reservedAssets < minimumReserve) {
    // Calculate how much to withdraw.
    uint256 shortfall = minimumReserve - $.reservedAssets;
    // @note Yield vault usage.
    uint256 maxWithdraw_ = $.yieldVault.maxWithdraw(address(this));
    uint256 assetsToWithdraw = shortfall > maxWithdraw_ ? maxWithdraw_ : shortfall;

    // Withdraw from the yield vault.
    if (assetsToWithdraw > 0) {
        // Cache the balance.
        uint256 balanceBefore = $.underlyingToken.balanceOf(address(this));

        // Withdraw.
        $.yieldVault.withdraw(assetsToWithdraw, address(this), address(this));

        // Update the reserve.
        $.reservedAssets += $.underlyingToken.balanceOf(address(this)) - balanceBefore;

        // Emit the event.
        emit ReserveRebalanced($.reservedAssets);
    } else if (force) {
        revert CannotRebalanceReserve();
    }
}
```

There is a following block with similar logic for deposits from the reserve into the yield vault.

In either case, the event it emitted with no indication of how large an adjustment was made, or whether the reserve is fully balanced.

Recommendations

Consider adding information to this event to indicate the size of the rebalancing, the type of the rebalancing and whether the reserve is now fully balanced.

Resolution

The suggested modifications were implemented in commit 0be49b1.

YE-14	Miscellaneous General Comments
Asset	All contracts
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Lack Of Named Errors**

   Related Asset(s): zETH.sol

   When withdrawing from the `zETH` contract, if the user's `zETH` balance is too low, the function will revert, but without a custom named error.

   Similarly, if the ETH balance of the `zETH` contract is too low for the withdrawal, the transfer on line [**51**] will fail, but also without a named error.

   ```
   function withdraw(uint256 value) public {
       require(balanceOf(msg.sender) >= value);
       _burn(msg.sender, value);
       payable(msg.sender).transfer(value);
       emit Withdrawal(msg.sender, value);
   }
   ```

   As the code generally uses named custom errors, consider adding them for these situations.

2. **Unclear Variable Names**

   Related Asset(s): YieldExposedToken.sol

   In the following section of code, the value contained in `maxShares` is converted into assets and stored in a variable named `requiredAssets`. This change of name is slightly unclear.

   ```
   // Check for a refund.
   if (maxShares > 0) {
       // Calculate the required amount of the underlying token.
       uint256 requiredAssets = convertToAssets(maxShares);

       if (assets > requiredAssets) {
           // Calculate the difference.
           uint256 refund = assets - requiredAssets;

           // Refund the difference.
           _sendUnderlyingToken(msg.sender, refund);

           // Update the `assets`.
           assets = requiredAssets;
       }
   }
   ```

   Consider renaming the variable `requiredAssets` to `maxAssets`.

3. **Unnecessary Cast**

   Related Asset(s): ERC20PermitUser.sol

   On line [**52**], the `token` address argument of the function `_permit()` is unnecessarily cast to an `address`.

   Remove the unnecessary cast.

4. **Extra Storage Reads**
   Related Asset(s): YieldExposedToken.sol
   There may be a possible gas optimisation in this code on line [722]:

```
function stakedAssets() public view returns (uint256) {
    YieldExposedTokenStorage storage $ = _getYieldExposedTokenStorage();
    return $.yieldVault.convertToAssets($.yieldVault.balanceOf(address(this)));
}
```

   The variable `$.yieldVault` is loaded from storage twice here. It could be cached to potentially save gas. Consider caching the variable.

5. **Typos**
   Related Asset(s): YieldExposedToken.sol, NativeConverter.sol
   On `YieldExposedToken.sol` line [231], the word *"Converter"* is misspelt.
   On `YieldExposedToken.sol` line [232], *"the migrate"* should be *"then migrate"*.

```
/// @notice The address of Native Conveter of this yeToken.
/// @notice Native Converter on Layer Ys can mint Custom Token independently of yeToken, and the migrate backing to Layer X.
      ↪  Please refer to `completeMigration` for more information.
```

   On `NativeConverter.sol` line [450], *"transfered"* should be *"transferred"*.
   Modify the comments as suggested.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team's responses to the raised issues above are as follows.

1. **Lack Of Named Errors** This issue was resolved in commit 048c7bc.
2. **Unclear Variable Names**
   The development team reviewed the issue and considered that no action was required at this time.
3. **Unnecessary Cast** This issue was resolved in commit 048c7bc.
4. **Extra Storage Reads**
   The development team reviewed the issue and considered that no action was required at this time.
5. **Typos** This issue was resolved in commit 048c7bc.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `forge` framework was used to perform these tests and the output is given below.

```
Ran 2 tests for test/tests-local/CustomToken.t.sol:CustomTokenTest
[PASS] test_CustomToken_init() (gas: 66821)
[PASS] test_onlyMinterBurner() (gas: 248127)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 5.25ms (460.86µs CPU time)

Ran 5 tests for test/tests-local/yeETHTest.t.sol:yeETHTest
[PASS] test_depositGasToken() (gas: 292320)
[PASS] test_depositGasTokenAndBridge() (gas: 295753)
[PASS] test_initialize() (gas: 108230)
[PASS] test_mintWithGasToken() (gas: 292347)
[PASS] test_onMessageReceived__dispatchCustomCrossNetworkInstruction() (gas: 323275)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 6.61ms (4.10ms CPU time)

Ran 7 tests for test/tests-local/WETH.t.sol:WETHTest
[PASS] test_bridgeBackingToLayerX() (gas: 166170)
[PASS] test_deposit() (gas: 109025)
[PASS] test_initialize() (gas: 65512)
[PASS] test_receive() (gas: 107741)
[PASS] test_version() (gas: 21611)
[PASS] test_withdraw() (gas: 138181)
[PASS] test_withdraw_insufficientBacking() (gas: 199368)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 11.08ms (1.73ms CPU time)

Ran 4 tests for test/tests-local/WETHNativeConverter.t.sol:WETHNativeConverterTest
[PASS] test_initialize() (gas: 74220)
[PASS] test_migrateGasBackingToLayerX() (gas: 236708)
[PASS] test_migrateGasBackingToLayerX_moreAssetsThanBacking() (gas: 128812)
[PASS] test_migrateGasBackingToLayerX_zeroAmount() (gas: 157106)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 11.56ms (1.41ms CPU time)

Ran 9 tests for test/tests-local/ERC20PermitUser.t.sol:ERC20PermitUserTest
[PASS] test_convertWithPermit_amountTooGreat() (gas: 73872)
[PASS] test_convertWithPermit_daiPermit_notAllowed() (gas: 75006)
[PASS] test_convertWithPermit_daiPermit_success() (gas: 323032)
[PASS] test_convertWithPermit_daiPermit_wrongSender() (gas: 77020)
[PASS] test_convertWithPermit_daiPermit_wrongSpender() (gas: 77337)
[PASS] test_convertWithPermit_permitDataMismatch() (gas: 118012)
[PASS] test_convertWithPermit_wrongSelector() (gas: 71531)
[PASS] test_convertWithPermit_wrongSender() (gas: 76414)
[PASS] test_convertWithPermit_wrongSpender() (gas: 76198)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 11.67ms (5.00ms CPU time)

Ran 25 tests for test/tests-local/NativeConverter.t.sol:NativeConverterTest
[PASS] test_NativeConverter_init() (gas: 63637)
[PASS] test_convert() (gas: 242654)
[PASS] test_convertWithPermit() (gas: 303585)
[PASS] test_convertWithPermit_noPermitData() (gas: 40855)
[PASS] test_convert_zeroAssets() (gas: 72460)
[PASS] test_convert_zeroReveiver() (gas: 73143)
[PASS] test_deconvert() (gas: 343904)
[PASS] test_deconvertAndBridge() (gas: 344044)
[PASS] test_deconvertAndBridge_sameNetwork() (gas: 290547)
[PASS] test_deconvertWithPermit() (gas: 401058)
[PASS] test_deconvertWithPermitAndBridge() (gas: 400985)
[PASS] test_deconvertWithPermitAndBridge_sameNetwork() (gas: 291160)
[PASS] test_deconvertWithPermit_noPermitData() (gas: 281737)
[PASS] test_deconvert_insufficientBacking() (gas: 291471)
[PASS] test_deconvert_zeroReceiver() (gas: 282368)
[PASS] test_deconvert_zeroShares() (gas: 282456)
[PASS] test_maxDeconvert_fullyBacked() (gas: 280643)
[PASS] test_maxDeconvert_partiallyBacked() (gas: 260760)
```

```
[PASS] test_maxDeconvert_paused() (gas: 306806)
[PASS] test_migrateBackingToLayerX() (gas: 300115)
[PASS] test_migrateBackingToLayerX_moreAssetsThanBacking() (gas: 232226)
[PASS] test_migrateBackingToLayerX_zeroAssets() (gas: 225492)
[PASS] test_pause() (gas: 64290)
[PASS] test_receiveUnderlyingToken() (gas: 312098)
[PASS] test_unpause() (gas: 61675)
Suite result: ok. 25 passed; 0 failed; 0 skipped; finished in 11.73ms (25.51ms CPU time)

Ran 12 tests for test/tests-local/yeTokenTest.t.sol:yeTokenTest
[PASS] test_collectYield() (gas: 378383)
[PASS] test_deposit() (gas: 322965)
[PASS] test_depositAndBridge() (gas: 327159)
[PASS] test_depositWithPermit() (gas: 395640)
[PASS] test_depositWithPermitAndBridge() (gas: 400547)
[PASS] test_donateAsYield() (gas: 126432)
[PASS] test_initialize() (gas: 108120)
[PASS] test_mint() (gas: 340934)
[PASS] test_onMessageReceived_completeMigration() (gas: 325119)
[PASS] test_rebalanceReserve_case1() (gas: 414349)
[PASS] test_rebalanceReserve_case2() (gas: 231952)
[PASS] test_withdraw() (gas: 371336)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 11.72ms (12.65ms CPU time)

Ran 4 tests for test/tests-local/YeUSDT.t.sol:YeUSDTTest
[PASS] test_YeUSDTinitialize() (gas: 61642)
[PASS] test_recacheUsdtTransferFeeParameters() (gas: 169993)
[PASS] test_transferFeePredictions(uint96,uint256) (runs: 10004, μ: 222562, ~: 224095)
[PASS] test_version() (gas: 21963)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 1.92s (1.91s CPU time)

Ran 8 test suites in 1.92s (1.99s CPU time): 68 tests passed, 0 failed, 0 skipped (68 total tests)
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

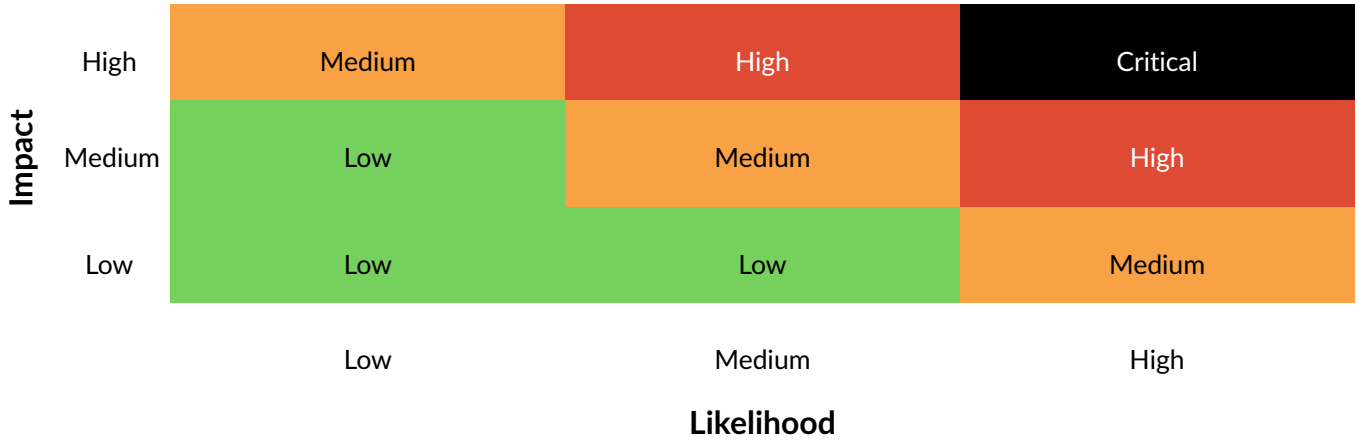

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].