

hexens x yelay

Security Review Report for Yelay

October 2025



Table of Contents

1. About Hexens
2. Executive summary
3. Security Review Details
 - Security Review Lead
 - Scope
 - Changelog
4. Severity Structure
 - Severity characteristics
 - Issue symbolic codes
5. Findings Summary
6. Weaknesses
 - ERC4626 accounting and execution mismatch between totalAssets and redemption paths
 - Improper use of approve instead of forceApprove from SafeERC20
 - EIP-4626 Spec Mismatch: maxWithdraw underflow reverts for small balances

1. About Hexens

Hexens is a pioneering cybersecurity firm dedicated to establishing robust security standards for Web3 infrastructure, driving secure mass adoption through innovative protection technology and frameworks. As an industry elite experts in blockchain security, we deliver comprehensive audit solutions across specialized domains, including infrastructure security, Zero Knowledge Proof, novel cryptography, DeFi protocols, and NFTs.

Our methodology combines industry-standard security practices combined with unique methodology of two teams per audit, continuously advancing the field of Web3 security. This innovative approach has earned us recognition from industry leaders.

Since our founding in 2021, we have built an exceptional portfolio of enterprise clients, including major blockchain ecosystems and Web3 platforms.

2. Executive Summary

This report covers the security review for Yelay. The review included a pull request that introduced ERC4626 compatibility to Yelay Lite vaults.

Our security assessment was a full review of the code, spanning a total of 3 days.

During our review, we did not identify any major severity vulnerabilities.

We did identify several minor severity vulnerabilities and code optimisations.

All of our reported issues were fixed or acknowledged by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

3. Security Review Details

- **Review Led by**

Jahyun Koo, Lead Security Researcher

- **Scope**

The analyzed resources are located on:

🔗 <https://github.com/YieldLayer/yelay-lite/pull/27>

The issues described in this report were fixed in the following commit:

🔗 <https://github.com/YieldLayer/yelay-lite/pull/27>

📌 Commit: 9c3682a8f7159f35abc55ade8353bd3fd74f1326

- **Changelog**

27 October 2025	Audit start
30 October 2025	Initial report
10 November 2025	Revision received
13 November 2025	Final report

4. Severity Structure

The vulnerability severity is calculated based on two components:

1. Impact of the vulnerability
2. Probability of the vulnerability

Impact	Probability			
	Rare	Unlikely	Likely	Very likely
Low	Low	Low	Medium	Medium
Medium	Low	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

▪ Severity Characteristics

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities that are highly likely to be exploited and can lead to catastrophic outcomes, such as total loss of protocol funds, unauthorized governance control, or permanent disruption of contract functionality.

High

Vulnerabilities that are likely to be exploited and can cause significant financial losses or severe operational disruptions, such as partial fund theft or temporary asset freezing.

Medium

Vulnerabilities that may be exploited under specific conditions and result in moderate harm, such as operational disruptions or limited financial impact without direct profit to the attacker.

Low

Vulnerabilities with low exploitation likelihood or minimal impact, affecting usability or efficiency but posing no significant security risk.

Informational

Issues that do not pose an immediate security risk but are relevant to best practices, code quality, or potential optimizations.

▪ Issue Symbolic Codes

Each identified and validated issue is assigned a unique symbolic code during the security research stage.

Due to the structure of the vulnerability reporting flow, some rejected issues may be missing.

5. Findings Summary

Severity	Number of findings
Critical	0
High	0
Medium	1
Low	2
Informational	0
Total:	3



■ Medium

■ Low



■ Fixed

6. Weaknesses

This section contains the list of discovered weaknesses.

YELAY6-3 | ERC4626 accounting and execution mismatch between totalAssets and redemption paths

Fixed ✓

Severity:

Medium

Probability:

Likely

Impact:

Medium

Path:

src/plugins/ERC4626Plugin.sol

Description:

The contract exhibits an inconsistency between its accounting layer and execution layer that affects share valuations and withdrawal operations. The `totalAssets()` function includes both the idle token balance held directly by the contract and the assets represented by the contract's `YelayLiteVault` position. This combined value is used in the base ERC4626 implementation's `convertToShares()` and `convertToAssets()` functions to determine share exchange rates during deposits and mints.

However, the `redeem()` and `withdraw()` functions only access funds from the `YelayLiteVault` position and do not utilize the idle balance when fulfilling withdrawal requests. The withdrawal amount calculation is based solely on the vault shares, effectively treating the idle balance as if it does not exist for redemption purposes.

```
function totalAssets() public view override returns (uint256) {
    uint256 shares = yelayLiteVault.balanceOf(address(this), projectId);
    return yelayLiteVault.convertToAssets(shares) + IERC20(asset()).balanceOf(address(this));
}

function redeem(uint256 shares, address receiver, address owner) public override returns (uint256) {
    uint256 maxShares = maxRedeem(owner);
    if (shares > maxShares) {
        revert ERC4626ExceededMaxRedeem(owner, shares, maxShares);
    }

    uint256 yelayLiteShares =
        shares.mulDiv(yelayLiteVault.balanceOf(address(this), projectId), totalSupply(), Math.Rounding.Floor);
```

```
uint256 assets = yelayLiteVault.redem(yelayLiteShares, projectId, address(this));  
  
_withdraw(_msgSender(), receiver, owner, assets, shares);  
  
return assets;  
}
```

Remediation:

Align accounting with execution. Either exclude idle balances from totalAssets and ensure conversions and limits reflect only vault-redeemable assets, or modify redeem and withdraw to first consume any idle asset balance and redeem only the shortfall from YelayLiteVault. In either approach, make previewRedeem and previewWithdraw mirror the exact execution path and use consistent rounding so previews and actual outcomes match.

YELAY6-1 | Improper use of approve instead of forceApprove from SafeERC20

Fixed ✓

Severity:

Low

Probability:

Rare

Impact:

Medium

Path:

src/plugins/ERC4626Plugin.sol#L76

Description:

The contract **ERC4626Plugin** directly invokes the token's native **approve()** function within the constructor instead of using the safer **forceApprove** method from the **SafeERC20** library. This approach can be unsafe when interacting with tokens that deviate from the ERC20 standard or implement non-standard approval logic. Certain tokens, such as USDT, do not return a boolean value from **approve**, which can cause these direct approval calls to revert or behave unpredictably.

```
IERC20(asset).approve(address(yelayLiteVault), type(uint256).max);
```

Remediation:

Consider using the function **forceApprove** provided by **SafeERC20**.

YELAY6-4 | EIP-4626 Spec Mismatch: maxWithdraw underflow reverts for small balances

Fixed ✓

Severity:

Low

Probability:

Rare

Impact:

Medium

Path:

src/plugins/ERC4626Plugin.sol#L234-L237

Description:

The contract overrides **maxWithdraw** to subtract a fixed margin from the withdrawable amount. When a user's convertible assets are less than this margin, the subtraction underflows and the call reverts. EIP-4626 specifies that **max*** view functions must not revert and should reflect the maximum amount available under current conditions. This behavior breaks standards compliance and can disrupt integrators or UIs that rely on **maxWithdraw** for parameter selection.

```
function maxWithdraw(address owner) public view override returns (uint256) {  
    return super.maxWithdraw(owner) - WITHDRAW_MARGIN;  
}
```

Remediation:

Ensure **maxWithdraw** never reverts and accurately reflects the maximum.

hexens x **yyelay**