hexens x yelay

Security Review Report for Yelay

April 2025

Table of Contents

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 audit covered the security assessment of Yelay Lite, a lightweight version of the Yelay yield protocol. Yelay Lite is a vault designed to manage a single underlying asset that is deployed to various different integrated DeFi protocols to generate yield.

Our security assessment was a full review of the smart contracts spanning a total of 1 week.

During our audit, we did not identify any security vulnerabilities. We did report two informational findings.

All of our reported issues were fixed 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**

SoonChan Hwang, Lead Security Researcher

- **Scope**

The analyzed resources are located on:

🔗 GitHub Repository

📌 **Commit:** `04312ce2935f66fef88f2c94f53164225027e5ba`

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

🔗 GitHub Repository

📌 **Commit:** `a19144e7ea03b2dc0ff9451c54cf4c8edd428b84`

- **Changelog**

31 March, 2025	Audit Start
07 April, 2025	Initial Report Delivered
08 April, 2025	Revision Received
11 April, 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.

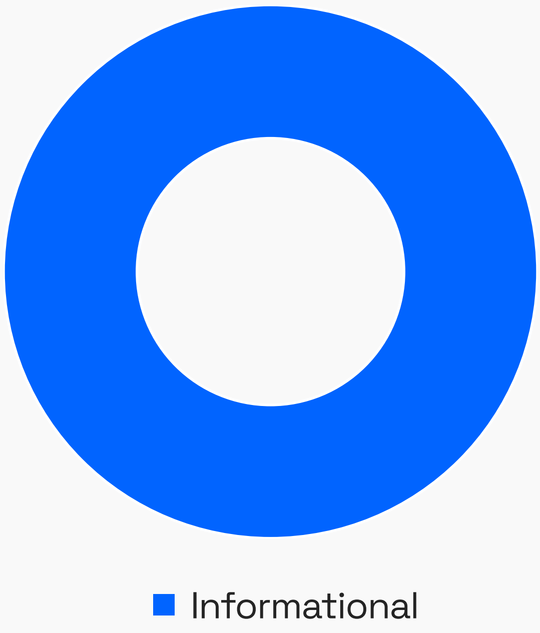
▪ Issue Symbolic Codes

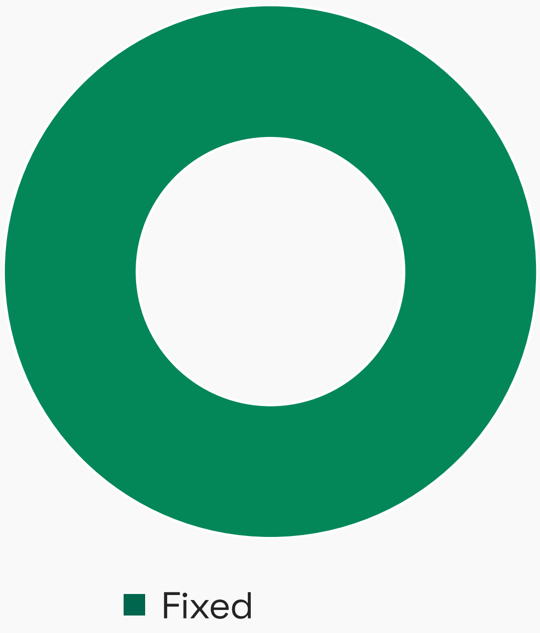
Each identified and validated issue is assigned a unique symbolic code during the security research stage.

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	0
🟩 Low	0
🟦 Informational	2
Total:	**2**





6. Weaknesses

This section contains the list of discovered weaknesses.

YELAY5-6 | Users can migrate into a global lock project that is unlocked

Fixed ✓

Severity:	Informational	Probability:	Unlikely	Impact:	Informational

Path:

src/plugins/DepositLockPlugin.sol#L144-L157

Description:

The deposit function checks if the **globalUnlockTime** is later than the current **block.timestamp** to ensure its not already unlocked.

```
function depositLocked(address vault, uint256 projectId, uint256 assets)
external returns (uint256 shares) {
    uint256 globalUnlockTime = projectGlobalUnlockTime[vault][projectId];
    if (globalUnlockTime > 0) {
        require(block.timestamp < globalUnlockTime,
LibErrors.GlobalUnlockTimeReached(globalUnlockTime));
    }
}
```

However its possible to migrate to a project that is globally unlocked, since that check is missing.

Remediation:

Move the following code to the internal function **_addLockedDeposit**.

```
uint256 globalUnlockTime = projectGlobalUnlockTime[vault][projectId];
    if (globalUnlockTime > 0) {
        require(block.timestamp < globalUnlockTime,
LibErrors.GlobalUnlockTimeReached(globalUnlockTime));
    }
```

YELAY5-7 | FundsFacet::migratePosition does not prevent same-project migration

Fixed ✓

Severity:	Informational	Probability:	Very likely	Impact:	Informational

Path:

src/facets/FundsFacet.sol#L224-L234

Description:

FundsFacet::migratePosition does not prevent migration when **fromProject** and **toProject** are identical. This allows an invalid migration between the same project to occur at no cost to the caller, which may result in unintended effects.

```
//src/facets/FundsFacet.sol
...
    function migratePosition(uint256 fromProjectId, uint256 toProjectId,
uint256 amount) external notPaused {
        require(
            LibClients._isProjectActive(fromProjectId) &&
LibClients._isProjectActive(toProjectId)
                && LibClients._sameClient(fromProjectId, toProjectId),
            LibErrors.PositionMigrationForbidden()
        );
        _accrueFee();
        _burn(msg.sender, fromProjectId, amount);
        _mint(msg.sender, toProjectId, amount, "");
        emit LibEvents.PositionMigrated(msg.sender, fromProjectId, toProjectId,
amount);
    }
...
```

Remediation:

Consider preventing migration if **fromProjectId** and **toProjectId** are identical in **FundsFacet::migratePosition**, similar to below:

```
//src/facets/FundsFacet.sol
..
    function migratePosition(uint256 fromProjectId, uint256 toProjectId, 
uint256 amount) external notPaused {
        require(
            LibClients._isProjectActive(fromProjectId) && 
LibClients._isProjectActive(toProjectId)
                && LibClients._sameClient(fromProjectId, toProjectId)
++              && (fromProjectId != toProjectId),
            LibErrors.PositionMigrationForbidden()
        );
        _accrueFee();
        ...
    }
...
```

hexens x yelay