



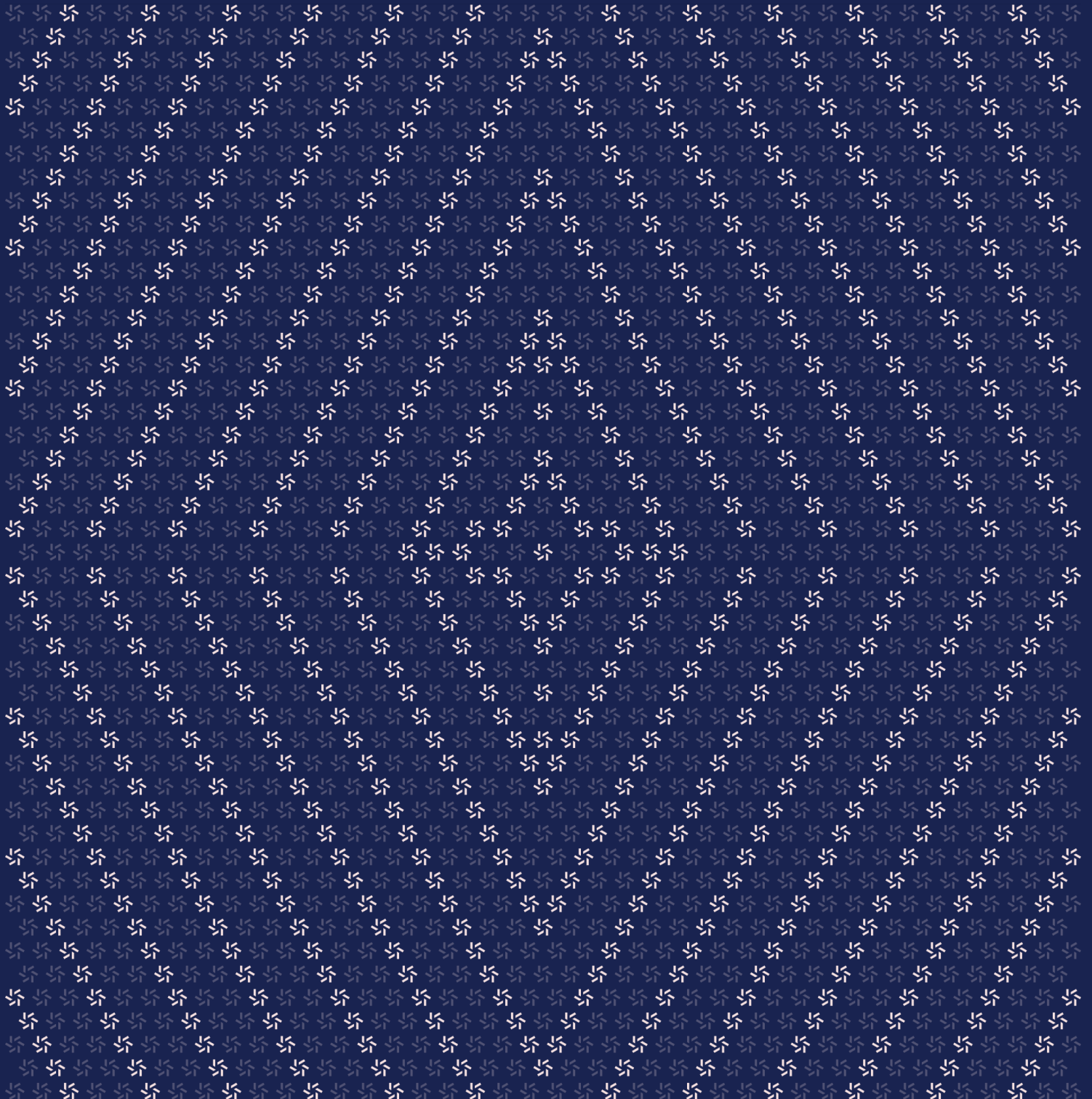
August 13, 2025

Prepared for
thai@mitosis.org
ray@mitosis.org
eddy@mitosis.org
Mitosis

Prepared by
Kritsada Dechawattana
Qingying Jie
Zellic

Extensible Vaults

Smart Contract Patch Review



Contents

About Zellic	3
<hr/>	
1. Overview	3
1.1. Executive Summary	4
1.2. Results	4
<hr/>	
2. Introduction	4
2.1. Scope	5
2.2. Disclaimer	6
<hr/>	
3. Detailed Findings	6
3.1. Inaccessible contract balance due to flawed withdrawal mechanism	7
3.2. The <code>claimLendingEmissions</code> function lacks proper batch-claim support	9
<hr/>	
4. Discussion	10
4.1. Privileged role operation	11
4.2. Updated business logic in the <code>isValidSignature</code> function now restricts calls to off-chain only	12
<hr/>	
5. Patch Review	12
5.1. Changes in contract structure	13
5.2. Introduction of new contracts	14

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the [#1 CTF \(competitive hacking\) team](#) worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io and follow [@zellic_io](#) on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io.



1. Overview

1.1. Executive Summary






Zellic conducted a security assessment for Mitosis on August 11th, 2025. During this engagement, Zellic reviewed Extensible Vaults's code for security vulnerabilities, design issues, and general weaknesses in security posture.

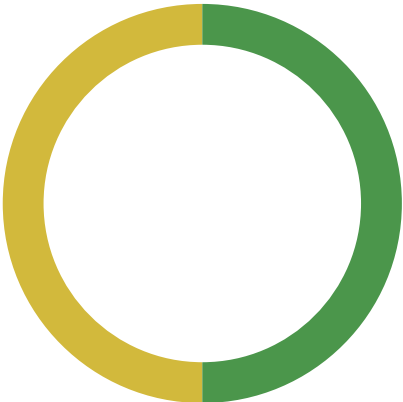
1.2. Results

During our assessment on the scoped Extensible Vaults contracts, we discovered two findings. No critical issues were found. One finding was of medium impact and one was of low impact.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Mitosis in the Discussion section ([4. 7](#)).

Breakdown of Finding Impacts

Impact Level	Count
 Critical	0
 High	0
 Medium	1
 Low	1
 Informational	0



2. Introduction

We were asked to review a minor patch to Extensible Vaults, which refactored some contract structures and introduced a few new contracts, based on the previous version.

2.1. Scope

The engagement involved a review of the following targets:

Extensible Vaults Contracts

Type	Solidity
Platform	EVM-compatible
Target	Only changes between 17b1d6b...cb43175b
Repository	https://github.com/mitosis-org/extensible-vaults ↗
Version	cb43175b8bc274c8fcb4b65d996f057f2ffffb910
Programs	src/ERC4626NativeDepositProxy.sol src/ExtensibleVault.sol src/ExtensibleVaultWithExtraData.sol src/ReclaimQueueWithExtraData.sol src/managers/ExternalManager.sol src/managers/ListaCDPSStrategyManager.sol src/managers/ListaEarnStrategyManager.sol src/managers/ManagedAssetsBase.sol src/managers/yoVaultManager.sol

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
✈ Engagement Manager
jacob@zellic.io ↗

Chad McDonald
✈ Engagement Manager
chad@zellic.io ↗

Pedro Moura
✈ Engagement Manager
pedro@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Kritsada Dechawattana
✈ Engineer
kritsada@zellic.io ↗

Qingying Jie
✈ Engineer
qingying@zellic.io ↗

2.2. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.

3. Detailed Findings

3.1. Inaccessible contract balance due to flawed withdrawal mechanism

Target	ListaEarnStrategyManager		
Category	Business Logic	Severity	Medium
Likelihood	Medium	Impact	Medium

Description

The ListaEarnStrategyManager contract has a design flaw where its `totalBalance` function correctly tracks both vault shares and direct asset balance held by the contract, but the `withdraw` function can only withdraw assets from the vault. This creates a scenario where assets that accumulate directly in the contract balance become permanently inaccessible.

The `totalBalance` function calculates the following:

```
function totalBalance() external view returns (uint256) {
    return vault.convertToAssets(vault.balanceOf(address(this)))
        + IERC20(vault.asset()).balanceOf(address(this));
}
```

However, the `withdraw` function only handles vault withdrawals:

```
function withdraw(uint256 amount, address receiver)
    external
    onlyRole(WITHDRAWAL_ROLE)
    nonReentrant
{
    require(amount > 0, StdError.ZeroAmount());

    uint256 shares = vault.withdraw(amount, receiver, address(this));
    require(shares > 0, ListaEarnStrategyManager__ZeroSharesReturned());

    emit Withdraw(asset, receiver, amount, shares);
}
```

Impact

The `ExtensibleVault.totalAssets` function that aggregates `ListaEarnStrategyManager`'s `totalBalance` values, which includes inaccessible contract balances, will report these assets as available, but they cannot be withdrawn. In extreme cases where significant assets become

trapped in contract balances, the vault reports high `totalAssets` but will be unable to honor withdrawal requests.

```
function totalAssets()  
    public  
    view  
    virtual  
    override(IExtensibleVault, ERC4626Upgradeable)  
    returns (uint256)  
{  
    address[] memory managers = getRoleMembers(MANAGER_ROLE);  
    uint256 managerAssets = 0;  
    for (uint256 i = 0; i < managers.length; i++) {  
        managerAssets += IManager(managers[i]).totalBalance();  
    }  
    return managerAssets + IERC20(asset()).balanceOf(address(this));  
}
```

Recommendations

Implement a comprehensive withdrawal strategy that handles both vault and direct contract balance to ensure all tracked assets from `totalBalance` remain accessible.

Remediation

This issue has been acknowledged by Mitosis, and a fix was implemented in commit [b30eddef](#).

3.2. The `claimLendingEmissions` function lacks proper batch-claim support

Target	ListaEarnStrategyManager		
Category	Coding Mistakes	Severity	Low
Likelihood	Low	Impact	Low

Description

The `claimLendingEmissions` function in `ListaEarnStrategyManager` contains a design flaw where it creates a single-element accounts array but accepts multiple tokens, amounts, and proofs arrays. The function hardcodes `accounts[0] = address(this)`, creating an array with only one account, while the `batchClaim` function expects the arrays to be properly aligned for batch operations across multiple accounts.

```
function claimLendingEmissions(
    address receiver,
    address[] memory tokens,
    uint256[] memory amounts,
    bytes32[][] memory proofs
) external onlyRole(EMISSION_MANAGER_ROLE) nonReentrant {
    address[] memory accounts = new address[](1);
    accounts[0] = address(this); // Only one account

    rewardDistributor.batchClaim(accounts, tokens, amounts, proofs); // Zellic
    Note: Expects the arrays to be properly aligned across multiple accounts

    [...]
}
```

Impact

The function will revert when called with multiple tokens. Users with `EMISSION_MANAGER_ROLE` who attempt to claim multiple token types in a single call will be unable to do so.

Recommendations

Modify the `claimLendingEmissions` function to support multiple accounts, aligning with the `batchClaim` function's behavior.

Remediation

This issue has been acknowledged by Mitosis, and a fix was implemented in commit [8a182d69](#) ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Privileged role operation

All manager contracts (ExternalManager, ListaCDPStrategyManager, ListaEarnStrategyManager, and yoVaultManager) implement a `withdraw` function protected only by the `WITHDRAWAL_ROLE`, allowing any address granted this role to extract funds to arbitrary receivers instead of being exclusive to the contract ExtensibleVault. This design creates an access-control vulnerability where administrators can grant `WITHDRAWAL_ROLE` to malicious actors or where compromised accounts with this role can drain manager funds.

```
function withdraw(uint256 amount, address receiver)
external
onlyRole(WITHDRAWAL_ROLE)
nonReentrant
{
    [...]
}
```

Consider implementing a multi-signature or DAO mechanism alongside timelock controls for critical operations such as role assignments. This would help prevent single-point-of-failure vulnerabilities in admin accounts.

Remediation

This issue has been acknowledged by Mitosis.

Mitosis provided the following response:

```
WITHDRAWAL_ROLE will actually be integrated with properly implemented on-chain contracts such as ExtensibleVault in the actual contract setup, and in such cases, there are practically no issues.
```

4.2. Updated business logic in the isValidSignature function now restricts calls to off-chain only

During our assessment, we identified that a new function, `isValidSignature`, was added to the `ListaEarnStrategyManager` contract, providing signature validation capability.

```
function isValidSignature(bytes32 hash_, bytes memory signature)
    external
    view
    returns (bytes4 magicValue)
{
    if (getRoleMemberCount(DEFAULT_ADMIN_ROLE) == 0) return 0xffffffff;

    // NOTE: we will only verify the signature of the admin at zero index
    address admin = getRoleMember(DEFAULT_ADMIN_ROLE, 0);
    if (admin == address(0)) return 0xffffffff;

    if (ECDSA.recover(hash_, signature) == admin) return ERC1271_MAGICVALUE;

    // invalid
    return 0xffffffff;
}
```

The Mitosis team updated their business requirements for the `isValidSignature` function from commit [26457c2](#) to [cb43175](#) and modified their codebase in commit [d86fa16](#) by restricting the function to be callable only off-chain.

```
function isValidSignature(bytes32 hash_, bytes memory signature)
    external
    view
    returns (bytes4 magicValue)
{
    // NOTE: restrict access to off-chain calls only by checking both sender and
    gas price
    require(_msgSender() == address(0), StdError.Unauthorized());
    require(tx.gasprice == 0, StdError.Unauthorized());

    return hasRole(SIGNATURE_VERIFIER_ROLE, ECDSA.recover(hash_, signature))
        ? ERC1271_MAGICVALUE
        : bytes4(0xffffffff);
}
```

5. Patch Review

The purpose of this section is to document the exact diffs of the codebase that were considered in scope for this audit. Here we note the changes in contract structure as well as the introduction of new contracts.

As requested by Mitosis, we focused on the changes made between commit [17b1d6b8](#) and commit [cb43175b](#).

5.1. Changes in contract structure

Changes were made to the `ManagedAssetVault`, `ManagerVault`, and `yoVaultManager` contracts.

These are the changes related to **ManagedAssetVault**:

- The contract `ManagedAssetVault` has been renamed to `ExtensibleVault`.
- The logic for modifying `managedAssets` in the contract has been removed. The return value of the `totalAssets` function has been changed from the sum of `managedAssets` and the contract's asset balance to the sum of all managers' `totalBalance` and the contract's asset balance.
- It inherited the contract `Pausable`. A sender with the `DEFAULT_ADMIN_ROLE` can pause or unpause certain functions — for example, functions `deposit` and `withdraw`.

These are the changes related to **ManagerVault**:

- The contract `ManagerVault` has been renamed to `ExternalManager` and has inherited the contract `ManagedAssetsBase`.
- The inherited contract `ManagedAssetsBase` contains the logic for modifying the state variable `managedAssets`.
- Two view functions were added, `asset` and `totalBalance`, to comply with the new `IManager` interface.
 - The `asset` function returns the address of the asset that the contract can operate on.
 - The `totalBalance` function returns the sum of `managedAssets` and the contract's asset balance.
- The functions `depositToAssetManager` and `withdrawFromAssetManager` were added, allowing a sender with the `ASSET_MANAGER_ROLE` to withdraw assets from the contract or return assets to the contract. When using the `depositToAssetManager` function to withdraw assets from the contract, `managedAssets` increases by the same amount; using the `withdrawFromAssetManager` function does the opposite.

These are the changes related to **yoVaultManager**:

- Two view functions were added, `asset` and `totalBalance`, to comply with the new `IManager` interface.
 - The `asset` function returns the underlying asset of the `yoVault`.
 - The `totalBalance` function returns the sum of the amount of assets corresponding to the shares held by the contract, assets pending redemption, and the contract's asset balance.

- The function `deposit` has added a check to ensure that the number of shares obtained from depositing into the `yoVault` is greater than zero.

Additionally, the contract `ManagedAssetVaultWithExtraData` has been renamed to `ExtensibleVaultWithExtraData`.

5.2. Introduction of new contracts

The update introduced contracts `ERC4626NativeDepositProxy`, `ListaCDPStrategyManager`, and `ListaEarnStrategyManager`.

The contract `ERC4626NativeDepositProxy` supports depositing directly with the native token. The functions `deposit` and `mint` first convert the native token into the wrapped native token and then deposit the wrapped native token into the specified vault.

The contract `ListaEarnStrategyManager` supports claiming Lista emissions in addition to basic deposit/withdraw operations to/from the specified vault.

In the contract `ListaCDPStrategyManager`, the `DEFAULT_ADMIN_ROLE` can configure the delegator for `slisBNBx`. When users call the function `deposit` to transfer `WBNB` to the contract, the contract converts it into `BNB` and stakes it in the `staking` contract to obtain `slisBNB`, then converts it into `slisBNBx` through the `pool` contract. The withdrawal process consists of three steps:

1. Call the function `requestWithdrawToLista` to convert `slisBNBx` into `slisBNB`, and create a withdraw request in the `staking` contract.
2. Call the function `claimWithdrawFromLista` to claim the withdrawal.
3. Call the function `withdraw` to retrieve `WBNB` to the specified receiver.

Additionally, the contract `ListaCDPStrategyManager` also supports claiming Lista emissions.