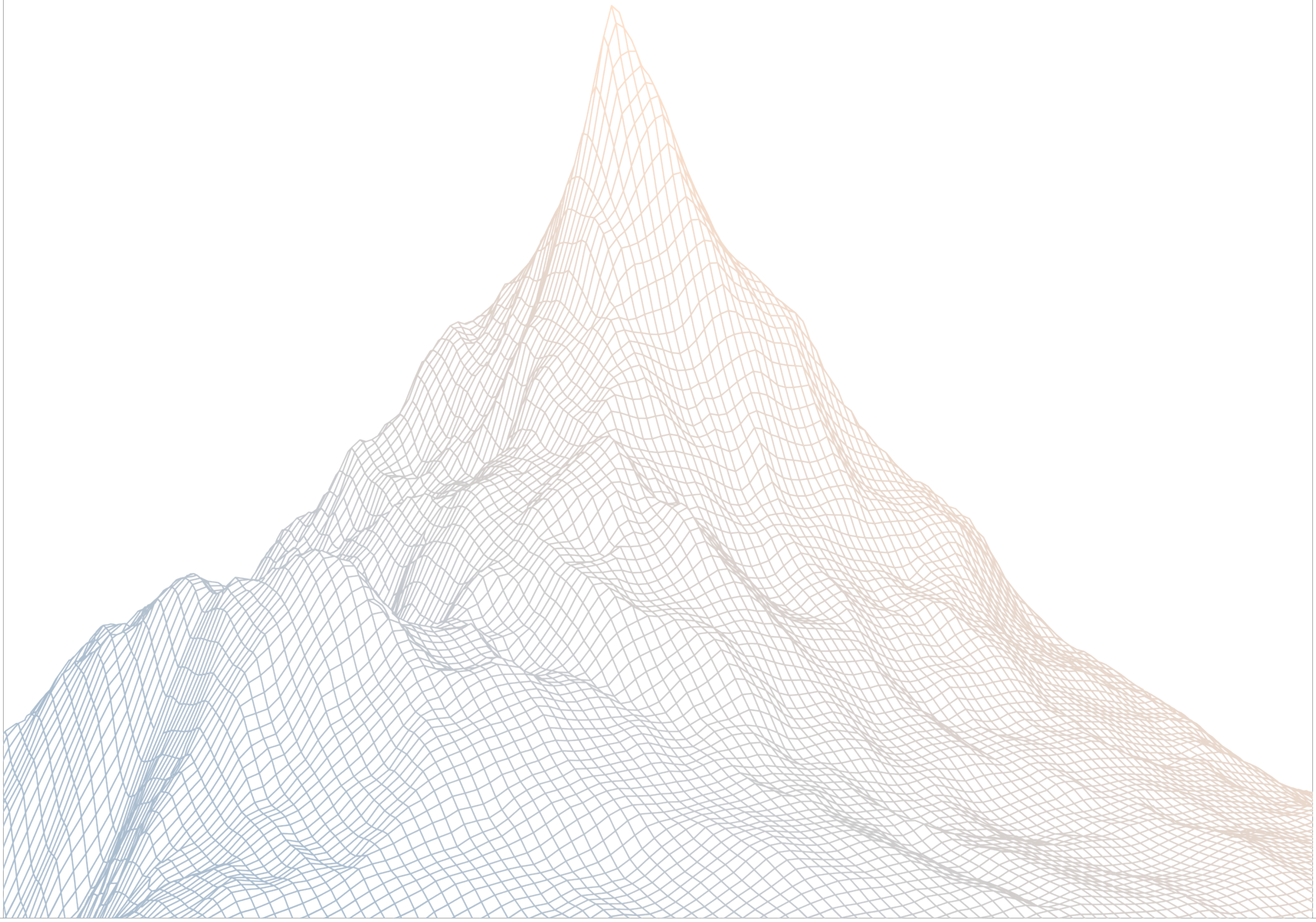
Zenith

Vesu V2

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: August 28th to September 4th, 2025

AUDITED BY: J4X
Mario Poneder

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 Vesu V2

Vesu is a fully open and permissionless lending protocol built on Starknet. Users can supply crypto assets (earn), borrow crypto assets and build new lending experiences on Vesu without relying on intermediaries. The Vesu lending protocol is not controlled by a governance body and there exists no governance token. Instead, Vesu is built as a public infrastructure giving everyone equal access to all functions and is free for everyone to use.

2.2 Scope

The engagement involved a review of the following targets:

Target	vesu-v2
Repository	https://github.com/vesuxyz/vesu-v2
Commit Hash	`053905a06e807197c0d750bbf66d1df133e268d0`
Files	`Diff up to 053905a06e807197c0d750bbf66d1df133e268d0`

Target	Vesu v2 Mitigation Review
Repository	https://github.com/vesuxyz/vesu-v2
Commit Hash	`7a848ce3196d62cae96cbf84fd7f80ee433fe203`
Files	`Diff up to 7a848ce3196d62cae96cbf84fd7f80ee433fe203`

2.3 Audit Timeline

August 28, 2025	Audit start
September 4, 2025	Audit end
September 18, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	3
Informational	2
Total Issues	**5**

3

Findings Summary

ID	Description	Status
L-1	Missing vToken creation and factory mapping updates in pool's add_asset function	Resolved
L-2	Missing oracle validation in permissionless pool creation	Acknowledged
L-3	Rounding in calculate_withdrawable_assets is done incorrectly	Acknowledged
I-1	Oracle can be set to zero address	Acknowledged
I-2	Delegations can be done multiple times	Acknowledged

4

Findings

4.1 Low Risk

A total of 3 low risk findings were identified.

[L-1] Missing vToken creation and factory mapping updates in pool's `add_asset` function

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

Target

- src/pool.cairo#L1084-L1129

Description:

The pool's `add_asset` function contains an inconsistency compared to the factory's `create_pool` function (and compared to Vesu v1-upgrade). When adding a new asset to an existing pool, the `add_asset` function does not create a corresponding vToken contract for the asset and does not update the factory's `v_token_for_asset/asset_for_v_token` mappings.

This is inconsistent with the factory's `create_pool` function, which properly calls `create_v_token` for each asset during pool creation. The `create_v_token` function correctly deploys a new vToken contract and updates both factory mappings `v_token_for_asset/asset_for_v_token`.

This means that assets added after pool creation will not have accessible vTokens, breaking the expected functionality where every pool asset should have a corresponding vToken for collateral representation.

The issue causes the following impacts:

- Users cannot interact with vTokens for assets added post-creation.
- The factory's mapping function `v_token_for_asset` will return zero addresses for these assets
- Asymmetry between assets added during pool creation vs. assets added later.

Recommendations:

It is recommended to extend the `add_asset` function signature to include vToken parameters, add factory integration to update the mappings and include vToken creation.

Vesu: Resolved with PR-62 and @813fc315cd...

Zenith: Verified. Resolved by adding an `add_asset` function to the factory, which updates the mappings and creates the corresponding vToken.

[L-2] Missing oracle validation in permissionless pool creation

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

Target

- src/pool_factory.cairo#L193-L285
- src/pool.cairo#L1182-L1189

Description:

The `create_pool` function in the `PoolFactory` contract and the `set_oracle` function in the `Pool` contract accept an arbitrary `oracle` parameter without any validation or whitelisting mechanism. This allows anyone to deploy a pool with a custom, potentially malicious oracle contract, which poses significant security risks to users by providing manipulated price data.

Trust assumption violation: Users naturally assume that pools created through the official factory contract use trusted infrastructure.

Contrast with Vesu v1-upgrade: This represents a security regression from Vesu v1-upgrade, where oracles were components of trusted extension contracts, providing inherent validation and trust guarantees.

Recommendations:

It is recommended to implement a whitelist of trusted oracle contract instances that need to be used for validation on pool creation (`create_pool`) and when changing a pool's oracle later on (`set_oracle`).

Vesu: Acknowledged. The trust model does indeed change in Vesu V2 in this regard. We do not enforce a specific oracle in the factory, but allow curators to use custom oracles. Instead, we will show the oracle info on the Vesu UI, including warnings if the oracle config is not an "official" one.

Zenith: Acknowledged.

[L-3] Rounding in `calculate_withdrawable_assets` is done incorrectly

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

Target

- src/pool.cairo

Description:

In `calculate_withdrawable_assets` the utilization will always be upped by one, even if no rounding occured on the call to `utilization()`.

```
// Add 1 to the utilization returned by the pool to round it up.
if self.pool().utilization(asset) + 1 >= asset_config.max_utilization {
    return 0;
}
```

This leads to a incorrect calculation and the user not being able to withdraw assets even if the asset is not fully utilized in the edge case that no rounding occured on the calculation.

Recommendations:

We recommend allowing the rounding direction to be passed to the `utilization()` function. That way, no rounding will occur in case it is not needed.

Vesu: Acknowledged. The benefits of conservatively computing withdrawable assets outweigh the possible impact. Furthermore, this edge case would fix itself if the function is called again in the next block with interest accruing and changing the market's utilization.

4.2 Informational

A total of 2 informational findings were identified.

[I-1] Oracle can be set to zero address

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

Target

- src/pool.cairo

Description:

The `set_oracle()` function employs no check that ensures that the oracle is not set to a zero address.

Recommendations:

We recommend adding a check.

Vesu: Acknowledged.

[I-2] Delegations can be done multiple times

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

Target

- src/pool.cairo

Description:

The `modify_delegation()` function allows for setting the delegation to the same value that it is already and emitting an additional event.

```
/// Modifies the delegation status of a delegator to a delegatee
/// # Arguments
/// * `delegatee` - address of the delegatee
/// * `delegation` - delegation status (true = delegate, false = undelegate)
fn modify_delegation(ref self: ContractState, delegatee: ContractAddress,
    delegation: bool) {
    self.assert_not_paused();

    self.delegations.write((get_caller_address(), delegatee), delegation);

    self.emit(ModifyDelegation { delegator: get_caller_address(), delegatee,
    delegation });
}
```

Recommendations:

We recommend checking if `self.delegations.read((delegator, delegatee)) == delegation` and reverting in that case.

Vesu: Acknowledged.