SHERLOCK

# Security Review For
# Vesu - Starknet Vault Kit

| | |
|---|---|
| Collaborative Audit Prepared For: | **Vesu - Starknet Vault Kit** |
| Lead Security Expert(s): | **CODESPECT** |
| Date Audited: | **September 10 - September 17, 2025** |
| Final Commit: | **0ccbf4c** |

## Introduction

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.

## Scope

Repository: ForgeYields/starknet_vault_kit

Audited Commit: 2b3ddc8602f4fe51baef767e3cc2d05f6a898dd2

Final Commit: 0ccbf4ce0f3131fdced947b7c7b4406fea110481

Files:

- packages/vault_allocator/src/decoders_and_sanitizers/avnu_exchange_-decoder_and_sanitizer/avnu_exchange_decoder_and_sanitizer.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/avnu_exchange_-decoder_and_sanitizer/interface.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/base_decoder_and__-sanitizer.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/decoder_custom_-types.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/erc4626_decoder_and__-sanitizer/erc4626_decoder_and_sanitizer.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/erc4626_decoder_and__-sanitizer/interface.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/interface.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/multiply_decoder_and__-sanitizer/interface.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/multiply_decoder_and__-sanitizer/multiply_decoder_and_sanitizer.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/simple_decoder_and__-sanitizer.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/uncap_decoder_and__-sanitizer/interface.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/uncap_decoder_and__-sanitizer/uncap_decoder_and_sanitizer.cairo

- packages/vault_allocator/src/decoders_and_sanitizers/vesu_decoder_and_-sanitizer/interface.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/vesu_decoder_and_-sanitizer/vesu_decoder_and_sanitizer.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/vesu_v2_decoder_and_-sanitizer/interface.cairo
- packages/vault_allocator/src/decoders_and_sanitizers/vesu_v2_decoder_and_-sanitizer/vesu_v2_decoder_and_sanitizer.cairo
- packages/vault_allocator/src/integration_interfaces/avnu.cairo
- packages/vault_allocator/src/integration_interfaces/pragma.cairo
- packages/vault_allocator/src/integration_interfaces/vesu.cairo
- packages/vault_allocator/src/lib.cairo
- packages/vault_allocator/src/manager/errors.cairo
- packages/vault_allocator/src/manager/interface.cairo
- packages/vault_allocator/src/manager/manager.cairo
- packages/vault_allocator/src/middlewares/avnu_middleware/avnu_-middleware.cairo
- packages/vault_allocator/src/middlewares/avnu_middleware/errors.cairo
- packages/vault_allocator/src/middlewares/avnu_middleware/interface.cairo
- packages/vault_allocator/src/periphery/price_router/errors.cairo
- packages/vault_allocator/src/periphery/price_router/interface.cairo
- packages/vault_allocator/src/periphery/price_router/price_router.cairo
- packages/vault_allocator/src/vault_allocator/errors.cairo
- packages/vault_allocator/src/vault_allocator/interface.cairo
- packages/vault_allocator/src/vault_allocator/vault_allocator.cairo
- packages/vault/src/lib.cairo
- packages/vault/src/redeem_request/errors.cairo
- packages/vault/src/redeem_request/interface.cairo
- packages/vault/src/redeem_request/redeem_request.cairo
- packages/vault/src/vault/errors.cairo
- packages/vault/src/vault/interface.cairo
- packages/vault/src/vault/vault.cairo

## Final Commit Hash

0ccbf4ce0f3131fdced947b7c7b4406fea110481

## Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

## Issues Found

| High | Medium | Low/Info |
|---|---|---|
| 1 | 4 | 2 |

## Issues Not Fixed and Not Acknowledged

| High | Medium | Low/Info |
|---|---|---|
| 0 | 0 | 0 |

# Issue H-1: Incorrect amount of fee shares minted for management and perfomance fees

Source: https://github.com/sherlock-audit/2025-09-vesu/issues/40

## Summary

Within the `report(...)` call in `vault.cairo`, two types of fees are calculated:

- **Management fees** – based on all assets managed by the vault.
- **Performance fees** – based on the profit generated by the vault.

The issue arises when both fees are applied simultaneously. This leads to an incorrect calculation of the number of shares to be minted, resulting in a loss for the protocol.

## Vulnerability Details

The issue can be best illustrated with the following example:

1. Initial state:
    - `prev_aum = 1000 USD`
    - `total_shares = 1000`
    - Initial share price = **1 USD**
2. After executing `report(2000)` (ignoring virtual shares):
    - `after_aum = 2000 USD`
    - Fees assumed:
        - `management_fees = 100 USD`
        - `performance_fees = 100 USD`
        - **Total fees = 200 USD**
3. Calculations:

| Step | Formula | Result |
|---|---|---|
| Management shares | (100 × 1000) / (2000 – 100) | ≈ 52.63 |
| Performance shares | (100 × (1000 + 52.63)) / (2000 – 100) | ≈ 55.4 |
| Total shares minted | 52.63 + 55.4 | ≈ 108 |
| New share price | 2000 / (1000 + 108) | ≈ 1.81 USD |

| Step | Formula | Result |
| --- | --- | --- |
| Fee share value | 108 × 1.81 | ≈ 195.48 USD |
| Actual fees owed | – | 200 USD |

This mismatch occurs because the calculation of management shares does **not** subtract `performance_fee_assets` from `total_assets`.

## Impact

The protocol consistently loses a portion of the fees whenever both management and performance fees are applied.

## Code Snippet

vault.cairo#L750

## Tool Used

Manual Review

## Recommendation

Adjust the management fee share calculation to also subtract performance fee assets:

```
let management_fee_shares = math::u256_mul_div(
   management_fee_assets,
   total_supply + 1,
-     (total_assets - management_fee_assets) + 1,
+     (total_assets - management_fee_assets - performance_fee_assets) + 1,
   Rounding::Floor,
);
```

## Discussion

**0xSacha**

Validated.

Commit here https://github.com/ForgeYields/starknet_vault_kit/commit/642714d4bad547985567da57070ddf5aa9005cc7

**talfao**

The issue was fixed by subtracting performance fees from the total assets during the calculation of management shares

# Issue M-1: Incorrect slot calculation logic

Source: https://github.com/sherlock-audit/2025-09-vesu/issues/41

## Summary

Within `avnu_middleware.cairo`, setting `config` enables the rate limit. The slot is calculated based on the current time and `period`. Each caller is restricted from making more than `allowed_calls_per_period` calls within a slot. However, the slot calculation logic is incorrect, causing the slot to roll back, which may hinder normal interactions.

## Vulnerability Detail

In the `enforce_rate_limit` function, using the `%` operator to calculate the slot causes it to be limited to `[0, period - 1]`. When the `block_timestamp` advances by a full period, the slot will roll back. Moreover, under this calculation logic, each second within a period corresponds to a different slot, so it cannot effectively enforce call limits.

## Impact

Within a period, call limits cannot be effectively enforced. And the slot rolling back may prevent valid calls from being executed.

## Code Snippet

avnu_middleware.cairo#L201

## Tool Used

Manual Review

## Recommendation

Using `/` to calculate the slot ensures that the slot continuously moves forward and enforces call limits within each slot period.

```
if (self.config.read().is_some()) {
    let cfg = self.config.read().unwrap();
    let ts: u64 = get_dd();
-   let slot = ts % cfg.period;
+   let slot = ts / cfg.period * cfg.period;
    //...
}
```

## Discussion

**0xSacha**

Rejected, the slot is supposed to rollback at the end of each period

**talfao**

I just want to ask again, as you said, it should roll back.

In the current design, if `\%` is used, the `cfg.period` defines the number of slots.

If `allowed_calls_per_period` is, for example, 10, then we can use just 10 times some X slot; therefore, at some time in the future, we will not be able to call this router again.

**0xSacha**

yes exactly! Idea is tio no being able to call more than X time in the same period

**talfao**

For sure, that makes sense,

But if the period is **60 seconds**, the ts % 60, creating only 60 slots.

So if we are in second `1`, we are getting slot `1`. But if we are in second `61`, we are getting slot `1` again.

**0xSacha**

Hum inded this is an issue. I think we can keep it in cyclic but we need to store the last slot. if it's a new slot we reset the amount of calls to 0. Because we can also modify the period param.

This is valid then, let me add a fix

**0xSacha**

https://github.com/ForgeYields/starknet_vault_kit/commit/0ccbf4ce0f3131fdced947b7c7b4406fea110481

**talfao**

The issue was fixed with a design similar to the auditor's recommendation

# Issue M-2: Insufficient `modify_lever` parameter check allows strategists to steal tokens

Source: https://github.com/sherlock-audit/2025-09-vesu/issues/42

This issue has been acknowledged by the team but won't be fixed at this time.

## Summary

In the vault design, strategists are considered partially trusted actors. The Merkle tree restricts which contracts and functions a strategist can call, and controls which tokens they can operate on and who the beneficiaries of the operation are.
However, in the `MultiplyDecoderAndSanitizerComponent` component, the parameter check for the `modify_lever` operation is insufficient, which may allow strategists to use unauthorized tokens and steal tokens.

## Vulnerability Detail

In the `modify_lever` operation, pre-swap operations are allowed. Users can provide a custom swap path and slippage control to exchange input tokens for `collateral_asset` and `debt_asset`. When the swap path is not empty, the multiply actually charges the caller the first token in the swap path instead of the `collateral_asset`. However, `MultiplyDecoderAndSanitizerComponent` does not check these parameters. This could allow strategists to use other tokens from the `vault_allocator`.

Furthermore, because slippage is set by the user and there is no global minimum slippage control like in `avnu_middleware`, strategists could insert malicious tokens they deployed into the swap path to steal funds from the vault through the swap.

## Impact

Strategists could operate using unauthorized types of tokens, and swaps could be used to steal tokens from the vault.

## Code Snippet

multiply_decoder_and_sanitizer.cairo#L27

## Tool Used

Manual Review

## Recommendation

It is recommended to disable pre-swaps for the `modify_lever` operation. Swaps should only be performed under `avnu_middleware` with a global slippage limit in place.

## Discussion

**0xSacha**

Acknowledged.

We need to add a middleware contract to handle multiply operations. Regarding having a different path for the swap than expected, it would also be handled by the middleware contract.

**talfao**

Acknowledged by the client

# Issue M-3: The price oracle lacks important checks

Source: https://github.com/sherlock-audit/2025-09-vesu/issues/46

This issue has been acknowledged by the team but won't be fixed at this time.

## Summary

The `price_router` contract is used to pull token prices to be used in the `anvu_middleware` as slippage. However, the `get_value(...)` function lacks important checks that verify the integrity of those prices.

## Vulnerability Detail

The `pragma` oracle returns a `PragmaPricesResponse` struct:

```
pub struct PragmaPricesResponse {
    pub price: u128,
    pub decimals: u32,
    pub last_updated_timestamp: u64,
    pub num_sources_aggregated: u32,
    pub expiration_timestamp: Option<u64>,
}
```

There are 2 checks that are missing:

1. Staleness checks: The `last_updated_timestamp` variable is meant to be used to verify that the price returned is not too old and outdated. Additionally, the more volatile assets or the ones with fewer sources aggregated benefit from having more frequent staleness checks than the others. The `num_sources_aggregated` variable can be used for such a mechanism.
2. Price checks: The `price` should be checked to be greater than `0`. The pragma oracle doesn't have such a check in its implementation, and it's possible to return such a `price` if the sources misbehave or go down.

## Impact

The oracle can return outdated prices and set the slippage at an older, inaccurate price.

## Code Snippet

price_router.cairo#L54

## Tool Used

Manual Review

## Recommendation

Implement the recommended staleness and price checks.

## Discussion

**lpetroulakis**

This is acknowledged by the team but there is no need for a fix. The team will eventually use the Vesu price router here where there's already a sanity check in regard to the price validity. No need to be fixed.

# Issue M-4: The `vault` contract is not ERC-4626 compliant

Source: https://github.com/sherlock-audit/2025-09-vesu/issues/49

This issue has been acknowledged by the team but won't be fixed at this time.

## Summary

The `vault` contract is supposed to be ERC-4626 compliant. However, according to the EIP-4626, it doesn't fulfill all the `MUST` statements.

## Vulnerability Detail

More specifically:

1. The `max_deposit(...)/max_mint(...)` functions must return `0` when the contract is `paused`.
2. The `max_withdraw(...)` function must always return `0`, since withdrawals are disabled.
3. The `preview_redeem(...)` function must include the fees that get charged in the `request_redeem(...)` function.

## Impact

The `vault` contract is not ERC-4626 compliant.

## Code Snippet

vault.cairo

## Tool Used

Manual Review

## Recommendation

Implement the listed features in order to be ERC-4626 compliant.

## Discussion

**0xSacha**

Acknowledged. Valid issue. We accept the finding and will align the vault with ERC-4626 “MUST” rules:

max_deposit / max_mint: return 0 when paused == true. max_withdraw: return 0 at all times since direct withdraw is not supported; redemptions happen via the delayed redeem flow. preview_redeem: return net assets after all deterministic fees applied in request_redeem.

Will add a commit

**0xSacha**

This is kept as acknowledge but won't change the code as it would require a huge change. The logic for preview is defined in OZ component.

**talfao**

Acknowledged by the client

# Issue L-1: The `bring_liquidity(...)` function should have access control

Source: https://github.com/sherlock-audit/2025-09-vesu/issues/43

## Summary

The `bring_liquidity(...)` function allows the allocators to transfer assets into the vault. The function modifies the storage `buffer` and `aum` variables and doesn't have access control, allowing anyone to manipulate those variables.

## Vulnerability Detail

This is the `bring_liquidity(...)` function:

```
fn bring_liquidity(
    ref self: ContractState, amount: u256,
) { // Amount of assets to bring back
    let caller = get_caller_address();
    ERC20ABIDispatcher { contract_address: self.erc4626.asset() }
        .transfer_from(caller, starknet::get_contract_address(), amount);
    let new_buffer = self.buffer.read() + amount; // Calculate new buffer
    let new_aum = self.aum.read() - amount; // Calculate new AUM

    self.buffer.write(new_buffer); // Increase buffer
    self.aum.write(new_aum); // Decrease deployed AUM

    self
        .emit(
            BringLiquidity {
                caller, amount, new_buffer, new_aum, epoch: self.epoch.read(),
            },
        );
}
```

Users can essentially donate some tokens to manipulate the `aum` variable. For example, if `aum` is close to `0`, someone could make it `0`. This forces the protocol to call the `report(...)` function with `new_aum = 0`, else the call will revert because of this `if` statement:

```
fn report(ref self: ContractState, new_aum: u256) {

    // ...

    // @audit prev_aum will be 0 here
    let prev_aum = self.aum.read();
```

```
    // 1) Validate AUM change is within acceptable bounds
    if (prev_aum.is_non_zero()) {
        let abs_diff = if (new_aum >= prev_aum) {
            new_aum - prev_aum
        } else {
            prev_aum - new_aum
        };
        // Calculate percentage change: (abs_diff * 1e18) / prev_aum
        let mut delta_ratio_wad = (abs_diff * WAD) / prev_aum;
        if ((abs_diff * WAD) % prev_aum).is_non_zero() {
            delta_ratio_wad += 1; // Round up for safety
        }
        if (delta_ratio_wad > self.max_delta.read()) {
            Errors::aum_delta_too_high(delta_ratio_wad, self.max_delta.read());
        }
    } else if (new_aum.is_non_zero()) {
        Errors::invalid_new_aum(new_aum);
    }

// ...

}
```

## Impact

The protocol is forced to call `report(...)` with the value of 0.

## Code Snippet

vault.cairo#L786

## Tool Used

Manual Review

## Recommendation

Allow only vault allocator address to call the function.

## Discussion

**0xSacha**

Don't really get how a user could make it 0 if aum is close to 0 can you give more details?

If a random user want to make donation to the vault, he can do it and will increase share price. If the share price deviation sanity check fail, the curator of the vault will have to increase the max delta

**Kalogerone**

> Don't really get how a user could make it 0 if aum is close to 0 can you give more details?
> If a random user want to make donation to the vault, he can do it and will increase share price. If the share price deviation sanity check fail, the curator of the vault will have to increase the max delta

A user can just call the `bring_liquidity(...)` function with the current `aum` as the amount. For example, if `self.aum` is `10e6` in the form of USDC tokens, a user can call this function with an amount of `10e6`. This will trigger these calculations in the function:

```
let new_aum = self.aum.read() - amount; // Calculate new AUM which will be 0

// this will make self.aum to be 0
self.aum.write(new_aum); // Decrease deployed AUM
```

After that, protocol will have to call the `report()` function of the next epoch with `0` amount, as it will revert otherwise.

**0xSacha**

You are right, this specific case the manager migh come with a non zero value for aum and it will revert.

This is valid will provide a fix to make bring liqudidity only accessible via the vault allocator

**0xSacha**

https://github.com/ForgeYields/starknet_vault_kit/commit/f180fb0f5d4a498eedbca7223bc06f735b4e389d

**talfao**

The issue was fixed by introducing access control

# Issue L-2: Incorrect naming of `MultiplyDecoderAndSanitizerComponent`'s implementation

Source: https://github.com/sherlock-audit/2025-09-vesu/issues/44

## Summary

`MultiplyDecoderAndSanitizerComponent` is incorrectly named in its embeddable implementation, as it uses the same name as `VesuDecoderAndSanitizerComponent`.

## Vulnerability Detail

The implementation of the Multiply decoder is defined as:

```
#[embeddable_as(VesuDecoderAndSanitizerImpl)]
impl VesuDecoderAndSanitizer<
    TContractState, +HasComponent<TContractState>,
> of IMultiplyDecoderAndSanitizer<ComponentState<TContractState>>
```

This is identical to the Vesu decoder implementation:

```
#[embeddable_as(VesuDecoderAndSanitizerImpl)]
impl VesuDecoderAndSanitizer<
    TContractState,
    +HasComponent<TContractState>,
    +Erc4626DecoderAndSanitizerComponent::HasComponent<TContractState>,
> of IVesuDecoderAndSanitizer<ComponentState<TContractState>>
```

The Multiply implementation should follow the established naming convention and be named `MultiplyDecoderAndSanitizer`.

## Impact

This is a naming best-practice issue.

## Code Snippet

multiply_decoder_and_sanitizer.cairo#L19

## Tool Used

Manual Review

## Recommendation

Rename `VesuDecoderAndSanitizerImpl` to `MultiplyDecoderAndSanitizer`.

## Discussion

**0xSacha**

Absolutely, will provide a commit to fix it, this is validated

**0xSacha**

https://github.com/ForgeYields/starknet_vault_kit/commit/c620410eff84414e8cce4afd1333713cde0af6a5

**talfao**

The issue was fixed by renaming the mentioned component implementation.

# Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.