SHERLOCK

Security Review For
Vesu Vaults

Collaborative Audit Prepared For: **Vesu Vaults**
Lead Security Expert(s): **CODESPECT**
Date Audited: **September 24 - October 1, 2025**

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: vesuxyz/vesu-vaults

Audited Commit: e2674032c6dbd62a837dd9a3ca0926f7f2e17dc2

Final Commit: 58b0d136e1ae94752d7282ebd251c81354c078ec

Files:

- src/aum_provider/aum_provider.cairo
- src/aum_provider/errors.cairo
- src/aum_provider/interface.cairo
- src/lib.cairo
- src/merkle_tree/base.cairo
- src/merkle_tree/integrations/avnu.cairo
- src/merkle_tree/integrations/erc4626.cairo
- src/merkle_tree/integrations/starknet_vault_kit_strategies.cairo
- src/merkle_tree/integrations/vesu_v1.cairo
- src/merkle_tree/integrations/vesu_v2.cairo
- src/merkle_tree/registry.cairo
- src/periphery_interfaces/avnu_router_middleware.cairo
- src/periphery_interfaces/manager.cairo
- src/periphery_interfaces/pragma.cairo
- src/periphery_interfaces/price_router.cairo
- src/periphery_interfaces/starknet_vault_kit.cairo
- src/periphery_interfaces/vault_allocator.cairo
- src/periphery_interfaces/vesu_v1.cairo
- src/periphery_interfaces/vesu_v2.cairo

- src/price_router/errors.cairo
- src/price_router/interface.cairo
- src/price_router/price_router.cairo
- src/vault_factory/interface.cairo
- src/vault_factory/vault_factory.cairo
- src/vault_governor/errors.cairo
- src/vault_governor/interface.cairo
- src/vault_governor/vault_governor.cairo

Final Commit Hash

58b0d136e1ae94752d7282ebd251c81354c078ec

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- 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.
- 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
0	7	7

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: Vault governor cannot upgrade `target` contracts [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/28

Summary

The `vault_governor` contract implements the function `upgrade_contract_setup(...)`, intended to upgrade specified target contracts. However, due to an incorrect implementation, the function instead upgrades the `vault_governor` contract itself.

Vulnerability Detail

The issue lies in the following implementation:

```
/// Upgrades a target contract to a new class hash.
/// Only callable by the owner.
/// # Arguments
/// * `target` - Target contract address.
/// * `new_class_hash` - New class hash.
fn upgrade_contract_setup( // @audit-issue: function does not upgrade the TARGET
    ref self: ContractState, target: ContractAddress, new_class_hash: ClassHash,
) {
    self.ownable.assert_only_owner();
    self.upgradeable.upgrade(new_class_hash);
}
```

According to the function documentation, the upgrade should be applied to the provided `target` contract. Instead, the current implementation invokes `upgrade(...)` on the `vault_governor` contract itself.

Impact

This results in broken functionality: the contract cannot upgrade other target contracts as intended.

Code Snippet

vault_governor.cairo#L293

Tool Used

Manual Review

Recommendation

Update the implementation so that the function upgrades the specified `target` contract rather than the `vault_governor` itself.

Discussion

0xSacha

Arf this was not included in the provided commit hash https://github.com/vesuxyz/vesu-vaults/commit/b3fb5a1cca9cd5956654e8f2f24ca8d8b47b7a00

talfao

i do not have access to this codebase

talfao

Fixed before just wrong init commit pushed for audit

Lucas | Sherlock

Issue had already been fixed by the team - initial commit was not properly updated.

Issue M-2: Vault fee distribution incorrectly incurs redemption fees [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/29

Summary

The original design of `vault.cairo` allows requesting redemption of shares without paying redemption fees. However, since the `fee_recipient` is set to the `vault_governor` contract, this mechanism no longer works as intended. As a result, actors redeeming shares are still subject to redemption fees, even when they should be exempt.

Vulnerability Detail

The intended logic can be seen in vault.cairo:request_redeem(...):

```
let redeem_fees = if (owner == fees_recipient) {
    0
}
```

In practice, the `fees_recipient` is always the `vault_governor` contract, as configured by the following function:

```
fn set_fees_config(
    ref self: ContractState,
    redeem_fees: u256,
    management_fees: u256,
    performance_fees: u256,
) {
    self._assert_vault_curator();
    IVaultDispatcher { contract_address: self.vault.read() }
        .set_fees_config(
            get_contract_address(), redeem_fees, management_fees, performance_fees,
        );
    self.emit(FeesConfigSet { redeem_fees, management_fees, performance_fees });
}
```

All collected fees are routed to the `vault_governor`, which redistributes them via `claim_fees(...)`:

```
fn claim_fees(ref self: ContractState) {
    self._assert_owner_or_vault_curator();
    let vault_erc20_dispatcher = ERC20ABIDispatcher { contract_address:
    ↪  self.vault.read() };
```

```
    let this_vault_balance =
    ↪  vault_erc20_dispatcher.balance_of(get_contract_address());
    let protocol_fees = this_vault_balance * self.protcol_fees_bps.read() / MAX_BPS;
    let remaining_balance = this_vault_balance - protocol_fees;
    vault_erc20_dispatcher.transfer(self.protocol_fees_recipient.read(),
    ↪  protocol_fees);
    vault_erc20_dispatcher.transfer(self.fees_recipient.read(), remaining_balance);
}
```

As a result, both the protocol fee recipient and the vault governor's fee recipient end up holding shares that are still subject to redemption fees. This undermines the original intent of fee-exempt redemptions.

Impact

All fee shares distributed by the vault remain subject to redemption fees, preventing the designed exemption from functioning.

Code Snippet

vault.cairo#L578-L580]

Tool Used

Manual Review

Recommendation

Introduce a configurable list of addresses that are exempt from redemption fees within `vault.cairo`, rather than relying solely on the `fee_recipient`.

Discussion

nbundi

we could also fix this by fixing the fee_recipient in the vault to the vault_governor contract and then introduce a fee_recipient and claim_fees fn in the vault_governor contract wdyt

talfao

I little bit don't understand your comment,

Current fee_recipient is `vault_governor` that is the reason why the governor's recipient and protocol recipient are subject to the fees.

Like one of the other fixes would allow requesting redemption of fees through the governor's contract.

nbundi

”Like one of the other fixes would allow requesting redemption of fees through the governor's contract.” this is what i meant yes

talfao

oh okay :D yes!

0xSacha

I think it's a good way to deal with this issue

0xSacha

Finally i just removed this logic to have special address without redeem fees, just make things easier

https://github.com/ForgeYields/starknet_vault_kit/commit/18a9b54dc1f9c31787f703d9c962210f87096028

talfao

This fix kind of changes the original design.

We believe that different issue arise now that the fee receiver needs to pay redemption fees, which seems like an improper implementation because it will lead to the situation where the fee receiver is never able to fully redeem his fees, as he will always pay a fee.

0xSacha

It's fine the fee receiver is curator, and curator can change redeem fees at anypoint. It adds useless complexity to have condition if an address is subject or not to it.

shaflow01

We think that the current change will cause the fee recipient to incur fees again when collecting fees, resulting in the fees never being fully redeemed. This is a kind of destruction to the original design of `starknet_ault_kit`. We suggest retaining the original design Perhaps other fix methods can be chosen for this issue

0xSacha

The original design created unnecessary complexity, here is how vesu vault will collect fees without being affected by redemptions fees

https://github.com/vesuxyz/vesu-vaults/commit/58b0d136e1ae94752d7282ebd251c81354c078ec

talfao

Looks good now

Issue M-3: AUM inflated in vault kit strategies due to missing redemption fee deductions [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/30

Summary

Since `vault.cairo` is not fully ERC-4626 compatible, it reflects the charge of redemption fees only when the redemption request is initiated. The current logic in the AUM provider slightly inflates the value of shares.

Vulnerability Detail

The issue arises in the following code block:

```
let svk_len = self.starknet_vault_kit_strategies_len.read();
while i < svk_len {
    let strat = self.starknet_vault_kit_strategies.read(i); // @audit what is this?
    let disp = ERC4626ABIDispatcher { contract_address: strat };
    let asset = disp.asset();
    let bal = ERC20ABIDispatcher { contract_address: strat }
        .balance_of(vault_allocator);
    let conv = disp.convert_to_assets(bal); // @audit-issue: inflated AUM,
    ↪  redemption fees are not deducted
    let due = IVaultDispatcher { contract_address: strat }
        .due_assets_from_owner(vault_allocator);
    let assets_amount = conv + due;
    positions
        .append((strat, asset, Zero::zero(), assets_amount.try_into().unwrap(), 0));
    i += 1;
}
```

The `_get_aum(...)` function constructs positions with all share values and pending redemption requests that the vault allocator holds in other vault kit strategies. The problem is that this overestimates the AUM:

- `vault.cairo` is not fully ERC-4626 compatible and does not account for fees in `preview_redeem(...)`. Therefore, this approach cannot be used.
- Even if it did, the current `convert_to_assets(...)` implementation does not incorporate redemption fee deductions.

Impact

The AUM is inflated because `convert_to_assets(...)` returns values that do not account for redemption fees.

Code Snippet

aum_provider.cairo#L484-L487

Tool Used

Manual Review

Recommendation

Subtract redemption fees from the share value.

Discussion

0xSacha

Rejected, the aum provider price vault shares from contracts with based-epoch reports at their last reported value, yes it does not account potential fees on next report but neither the potential upside generated by underlying startegies. The longer a report is, the les efficient is this pricing method, but those vaults are selected from the desired vault curator config and those risks should be understood

talfao

For sure, agree on the side of the management fees about these risks, but the redemption fees are always there, wdyt?

talfao

Issue updated to reflect only redemption fees, which are always in charge (if set) and therefore the pure convert_to_assets approach is incorrect)

You are right about management fees, as we do not know potential profits/losses, which influence their amount

0xSacha

Valid and fixed

change in vesu vault https://github.com/vesuxyz/vesu-vaults/commit/b6a110187df0d630a158a81759d79ad35be3474f

include fee in preview redeem https://github.com/ForgeYields/starknet_vault_kit/commit/18a9b54dc1f9c31787f703d9c962210f87096028

allowbreak
#diff-92f78d7127597ace0179a784b462b9b6ee8826962210a24034d232272d9266ca

talfao

Fixed

Issue M-4: Missing debt asset APPROVAL leafs in `_add_vesu_v1_leafs(...)` and `_add_vesu_v2_leafs(...)` [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/31

Summary

When generating the Merkle root, `_add_vesu_v1_leafs(...)` and `_add_vesu_v2_leafs(...)` are missing the `approve(...)` for the debt asset, which prevents the strategist from repaying the debt.

Vulnerability Detail

When calling `_set_vault_config(...)`, the `VesuV1Config` and `VesuV2Config` authorize the `vault_strategist` for related operations and generate a `merkle_root`. The corresponding leafs are generated through `_add_vesu_v1_leafs(...)` and `_add_vesu_v2_leafs(...)`. In these two functions, authorization is granted for `collateral_asset` approval and the related `modify_position(...)`. However, authorization for the `debt_asset` is missing, which results in the strategist being able to borrow debt tokens but unable to repay them.

Impact

The strategist can borrow debt tokens but cannot repay the debt.

Code Snippet

vesu_v1.cairo#L13 vesu_v2.cairo#L14

Tool Used

Manual Review

Recommendation

It is recommended to add APPROVAL leafs for each debt asset.

Discussion

talfao

Changed formatting little bit

0xSacha

Valid and fixed

https://github.com/vesuxyz/vesu-vaults/commit/76c4ada6727c6c288e7253a634e3add336493371

talfao

Fixed

Issue M-5: Use `preview_redeem(...)` instead of `convert_to_assets(...)` [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/32

Summary

The `aum_provider` retrieves the number of assets corresponding to owned shares in ERC4626 strategies (vaults). However, the current implementation is flawed because it directly uses the current share price via `convert_to_assets(...)`. This method ignores potential redemption fees that may apply during `withdraw(...)` or `redeem(...)`. To align with the ERC4626 standard and obtain an accurate estimate, `preview_redeem(...)` should be used instead.

Vulnerability Detail

The current approach to interacting with ERC4626 strategies:

```
let erc4626_len = self.erc4626_strategies_len.read();
while i < erc4626_len {
    let strat = self.erc4626_strategies.read(i);
    let disp = ERC4626ABIDispatcher { contract_address: strat };
    let asset = disp.asset();
    let shares = ERC20ABIDispatcher { contract_address: strat }
        .balance_of(vault_allocator);
    let assets_amount = disp.convert_to_assets(shares); // @audit
    ↪  preview_redeem(shares) should be used
    positions
        .append((strat, asset, Zero::zero(), assets_amount.try_into().unwrap(), 0));
    i += 1;
}
```

As shown above, the contract relies on `convert_to_assets(...)`, which does not factor in redemption fees. This results in an inflated view of the redeemable asset amount and fails to reflect the actual number of assets available to the protocol.

Impact

Overestimation of collateral assets, leading to incorrect pricing of `vault.cairo` shares and potentially misleading AUM calculations.

Code Snippet

aum_provider.cairo#L469

Tool Used

Manual Review

Recommendation

Replace the usage of `convert_to_assets(...)` with `preview_redeem(...)` to ensure redemption fees or other deductions are properly accounted for and the asset amount reflects actual redeemable value.

Discussion

0xSacha

Valid and fixed

https://github.com/vesuxyz/vesu-vaults/commit/b6a110187df0d630a158a81759d79ad35be3474f

talfao

Fixed

Issue M-6: `report(...)` may be vulnerable to DoS [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/33

Summary

For `starknet_vault_kit`, the `report(...)` function is used to advance the cycle and handle the redemption queue. However, under certain circumstances, this function can be DoSed, causing the vault funds to be locked.

Vulnerability Detail

In the `report(...)` function, if `prev_aum` equals 0, then `new_aum` cannot be 0.

```
// 1) Validate AUM change is within acceptable bounds
if (prev_aum.is_non_zero()) {
    //...
} else if (new_aum.is_non_zero()) {
    Errors::invalid_new_aum(new_aum);
}
```

However, in `aum_provider`, reporting AUM will fetch the balance of the underlying tokens from `vault_allocator`.

```
// --- Add idle balances of all accounted assets to positions
i = 0;
while i < assets_accounted.len() {
    let asset: ContractAddress = *assets_accounted.at(i);
    let bal = ERC20ABIDispatcher { contract_address: asset }
        .balance_of(vault_allocator);
    positions
        .append((vault_allocator, asset, Zero::zero(), bal.try_into().unwrap(), 0));
    i += 1;
}
```

Impact

In a newly created vault, a malicious actor can transfer any amount of underlying tokens to the `vault_allocator` before the first `report(...)` is executable. This causes the new AUM to be non-zero during `report(...)`, preventing the `report(...)` from executing. As a result, the vault's profit and loss reporting and the redemption queue are blocked, leading to funds being locked.

Code Snippet

aum_provider.cairo#L527

Tool Used

Manual Review

Recommendation

It is recommended to add relevant logic to handle this situation.

Discussion

0xSacha

Valid

0xSacha

https://github.com/vesuxyz/vesu-vaults/commit/2c97bb3448e1bce18a180cf482d895d6874ba389

here is the fix

talfao

The fix looks good

Issue M-7: Storage var `vault_strategist` is never set [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/37

Summary

The storage variable `vault_strategist` is never updated. As a result, the previously assigned strategist continues to retain privileges over the vault allocator even after a new strategist is configured.

Vulnerability Detail

The `vault_strategist` root for the management merkle tree is intended to be updated through `_set_vault_config(...)`:

```
fn _set_vault_config(
    ref self: ContractState,
    vault_strategist: ContractAddress,
    // ...,
) {
    let current_vault_strategist = self.vault_strategist.read(); // @audit-issue
    ↪  always zero as never updated
    let vault_allocator_manager_disp = IManagerDispatcher {
        contract_address: vault_allocator_manager,
    };
    if current_vault_strategist.is_non_zero()
        && current_vault_strategist != vault_strategist {
        vault_allocator_manager_disp
            .set_manage_root(current_vault_strategist, Zero::zero());
    }
    let (_, root) = self
        ._generate_merkle_tree(
            vault,
            vault_allocator,
            vault_decoder_and_sanitizer,
            vault_vesu_v2_specific_decoder_and_sanitizer,
            vesu_v1_configs,
            vesu_v2_configs,
            erc4626_strategies,
            starknet_vault_kit_strategies,
            avnu_configs,
            avnu_router_middleware,
        );
    vault_allocator_manager_disp.set_manage_root(vault_strategist, root);
```

```
    // ...
}
```

The function checks whether a current strategist exists and, if so, resets its merkle root to zero. However, because `vault_strategist` is **never updated in storage**, `current_vault_strategist` always contains zero address. This makes the reset logic ineffective. Consequently, previously assigned strategists retain their original merkle roots and can continue controlling the vault allocator even after a new strategist is set.

Impact

The vault's trust model is compromised: multiple strategists may hold active control simultaneously, instead of only the most recently assigned one. This creates a risk of unauthorised fund management.

Code Snippet

vault_governor.cairo#L501

Tool Used

Manual Review

Recommendation

Ensure that `vault_strategist` is updated in storage after successfully setting the new merkle root in the allocator manager.

Discussion

0xSacha

Validated: fixed https://github.com/vesuxyz/vesu-vaults/commit/a7328c64584262712963fa5da48aa98ff7db6d23

talfao

Fixed now

Issue L-1: Uncheck whether `starknet_vault_kit_strategies` contain the vault that is the source of the funds [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/35

Summary

The funds controlled by `vault_governor` come from the vault, but strategies can also add other `starknet_vault_kit_strategies`. There is no check to verify whether the `starknet_vault_kit_strategies` contain the vault that is the source of the funds, which may lead to malicious depletion of vault funds.

Vulnerability Detail

The `vault_governor` allows the strategist to manage funds from the vault, which is a `starknet_vault_kit`. However, the strategy target can also be other `starknet_vault_kit`, and there is no mechanism to prevent the `starknet_vault_kit` in the strategy from being the same as the vault that is the source of the funds.

```
for starknet_vault_kit_strategy_elem in starknet_vault_kit_strategies {
    _add_starknet_vault_kit_strategies(
        ref leafs,
        ref leaf_index,
        vault_allocator,
        vault_decoder_and_sanitizer,
        *starknet_vault_kit_strategy_elem,
    )
}
```

Impact

A malicious vault creator can add the vault itself as a strategy target, allowing unlimited use of the vault's funds to mint shares and dilute the value of existing shares.

Code Snippet

vault_governor.cairo#L584

Tool Used

Manual Review

Recommendation

It is recommended to check that the `starknet_vault_kit_strategies` do not include the vault itself.

Discussion

0xSacha

Valid, here is the fix: https://github.com/vesuxyz/vesu-vaults/commit/ad71c0af6d064a0ddaeba98262afd857b0bb50ea

talfao

Fixed

Issue L-2: `get_value(...)` always rounds up [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/36

Summary

When fetching AUM, `get_value(...)` is used for conversion. However, `get_value(...)` always rounds up, which may result in an overestimation of AUM.

Vulnerability Detail

`get_value(...)` always rounds up. For conservative calculations, rounding up is not always appropriate.

```
fn get_value(...) -> u256 {
    //...
    let num: u256 = amount * base_price * scale_quote;
    let den: u256 = quote_price * scale_base;
    math::u256_mul_div(num, 1, den, math::Rounding::Ceil)
}
```

Impact

For example, when calculating `total_collateral_value`, rounding up may lead to an overestimation of AUM.

Code Snippet

price_router.cairo#L134

Tool Used

Manual Review

Recommendation

It is recommended to add a parameter to `get_value(...)` to control the rounding direction. For example, round up when calculating slippage and `total_debt_value`, and round down when calculating `total_collateral_value`.

Discussion

0xSacha

Valid

0xSacha

and fixed https://github.com/vesuxyz/vesu-vaults/commit/45f4d4fca7f5c34482f270233d7cc6eb985a0073

talfao

Fixed

Issue L-3: Missing Multiply Decoder leaves in the merkle tree [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/38

Summary

The current implementation of `simple_decoder_and_sanitizer` includes the Multiply Decoder, which enables the Vesu extension for adjusting leverage on positions. However, the Merkle tree that governs vault management, constructed during configuration setup, does not include leaves for this decoder.

Vulnerability Detail

When the merkle tree is generated in `_set_vault_config(...)` of the `vault_governor` contract:

```
let (_, root) = self
        ._generate_merkle_tree(
            vault,
            vault_allocator,
            vault_decoder_and_sanitizer,
            vault_vesu_v2_specific_decoder_and_sanitizer,
            vesu_v1_configs,
            vesu_v2_configs,
            erc4626_strategies,
            starknet_vault_kit_strategies,
            avnu_configs,
            avnu_router_middleware,
        );
```

The resulting tree only incorporates configuration for Vesu v1 and v2, but omits the Vesu multiplier. As a result, the Multiply Decoder present in `simple_decoder_and_sanitizer` is never usable through the configured merkle root.

Impact

The Multiply functionality cannot be accessed, preventing strategists from modifying the leverage of vault positions through this extension.

Code Snippet

vault_governor.cairo#L511-L521 simple_decoder_and_sanitizer.cairo#L69

Tool Used

Manual Review

Recommendation

Extend the Merkle tree generation in `_set_vault_config(...)` to include leaves for the Multiply Decoder, ensuring the functionality is properly integrated into vault management.

Discussion

0xSacha

removed it

https://github.com/ForgeYields/starknet_vault_kit/commit/16723695e39438ac8fcd1a102b0631b6f98b8d2a

talfao

Fixed by removing such implementation from base decoder

Issue L-4: Potential DoS of the `_get_positions(...)` due to the unbounded loop of redemption requests [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/39

Summary

The function `_get_positions(...)` retrieves all of the vault's positions. Some positions may be other Starknet vault kits, where the function queries the value of owned shares and pending redemption requests via `due_assets_from_owner(...)`. The issue is that `due_assets_from_owner(...)` iterates over **all NFTs (redemption requests)** owned by the `vault_allocator`. This creates an unbounded loop, which an attacker can exploit to cause a denial-of-service (DoS).

Vulnerability Detail

When the AUM provider fetches pending redemption requests:

```
let due = IVaultDispatcher { contract_address: strat
↪  }.due_assets_from_owner(vault_allocator);
```

The `due_assets_from_owner(...)` implementation is:

```
fn due_assets_from_owner(self: @ContractState, owner: ContractAddress) -> u256 {
    let balance = ERC721ABIDispatcher {
        contract_address: self.redeem_request.read().contract_address,
    }
        .balance_of(owner);
    let mut total_due_assets = 0;
    for i in 0..balance {
        total_due_assets += self
            .due_assets_from_id(
                IERC721EnumerableDispatcher {
                    contract_address: self.redeem_request.read().contract_address,
                }
                    .token_of_owner_by_index(owner, i),
            );
    }
    total_due_assets
}
```

Here, the `for` loop iterates over **every NFT** redemption request held by the `vault_allocator`. An attacker can mint and transfer a large number of minimal-value redemption

requests (e.g., worth only a few wei) to the `vault_allocator`. This artificially inflates the iteration count, which can lead to a `Too many Cairo steps` error and block execution/compilation of the transaction.

Impact

- Denial of Service: Calls to `_get_positions(...)` may fail due to excessive loop iterations.
- Griefing vector: While the attacker must spend assets to mint requests, even tiny requests suffice to disrupt the vault.
- Temporary: The effect persists until the epoch passes and redemption requests can be claimed, after which execution resumes normally.

Code Snippet

vault.cairo#L968-L984

Tool Used

Manual Review

Recommendation

Prevent the `vault_allocator` from receiving redemption requests.

Discussion

0xSacha

Hi, not really its not iterating over nfts but epochs delta between last handled and current, making the DoS almost impossible

talfao

But it is iterating via all unhandled redemption requsts no?

Like the likehood is low therefore the severity was adjusted of it.

But it can be caused by attacker if he transfers unhandled requests to allocator

Or am i missing anything

0xSacha

yes but unhandled redemption are pooled my epoch with redem nominal and redeem assets so attacker can not mint full of redeem requets

talfao

Let us please take a look, maybe we missed something or maybe we have not expressed ourselves well.

shaflow01

@0xSacha The DoS risk we identified exists here:

1. An attacker performs a small redemption, which mints an NFT.
2. The attacker sends this NFT to the `vault_allocator`.
3. Repeating the above steps results in many NFTs being owned by `vault_allocator`.
4. When executing `due_assets_from_owner`, it will iterate over all NFTs in `vault_allocator`, causing a DoS.

```
let balance = ERC721ABIDispatcher {
    contract_address: self.redeem_request.read().contract_address,
}
    .balance_of(owner);
let mut total_due_assets = 0;
for i in 0..balance {
```

0xSacha

Indeed this is an issue, sorry for not understanding it initially.

So there are 2 fixes I see:

- transform erc721 to erc1155, and id is the epoch and nominal being the amount within the epoch, so DOS becom impossible -> good fix but implies a lot of changes
- change request redeem and remove the receiver arg so only owner can receive its requested redeems and block transfers -> not the best but it works

talfao

And as each vault kit has only one allocator and exactly one redeem request contract...

Cannot you just block transfers to allocator?

0xSacha

Found an other way to resolve it.

The vault allocator was not able to receive any NFTs at all because no src5 component exposed. So I added it and make the vault allocator allowed nft receiver only when he is doing calls, so he can receive redeem request when requesting but not outside of doing calls.

There is still this issue when a malicious contract approved by the curator being called by the strategist, mint the vault allocator tons of nft request. But the vault allocator should again, allow only trusted sources for allowed strategies to interact with.

here is the fix https://github.com/ForgeYields/starknet_vault_kit/commit/efc010d2deb2c25d1ea7ea21b64bb326d79b23dd

talfao

Hello, the issue is not fixed as the SRC5 component is necessary only when the `safe_transfer_from(...)` is used, but not when the normal `transfer_from` is used.

So the attacker can mint to himself and then just transfer via `transfer_from`.

talfao

Additionally, if interface::IERC721_RECEIVER_ID is registered, the on_erc721_received function must be implemented, but the fix only registers IERC721_RECEIVER_ID.

0xSacha

Indeed, now obky safe transfer and safe mint is allowed for the redeem request, also the vault allocator implements erc721Receiver, here is the commit https://github.com/ForgeYields/starknet_vault_kit/commit/de13f33e60d8751562e7e0a915ef9c070654e9bf

talfao

Looks good now

Issue L-5: Starknet vault kit decoder implementation may break merkle tree proofs [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/40

Summary

Although potentially out of scope, this issue affects how Starknet Vault Kit leaves are constructed. The `claim_redeem(...)` functionality expects empty arguments during leaf creation, but the current `starknet_vault_kit_decoder_and_sanitizer.cairo` implementation requires the request ID as an argument. This discrepancy originates in the decoder contract, not the merkle tree generation.

Vulnerability Detail

The current implementation of `starknet_vault_kit_decoder_and_sanitizer::claim_redeem(...)`:

```
fn claim_redeem(self: @ComponentState<TContractState>, id: u256) -> Span<felt252> {
    let mut serialized_struct: Array<felt252> = ArrayTrait::new();
    id.serialize(ref serialized_struct);  // @audit ID is part of this
    serialized_struct.span()
}
```

Meanwhile, the vault leafs are constructed as:

```
leafs
  .append(
      ManageLeaf {
          decoder_and_sanitizer,
          target: starknet_vault_kit_strategy,
          selector: selector!("claim_redeem"),
          argument_addresses: array![].span(), // @audit no ID
          description: "Claim Redeem" + " " +
          ↪  get_symbol(starknet_vault_kit_strategy), // @
      },
  );
```

Because the leafs omit the ID, the proof for `claim_redeem` may fail. However, the ID should **not** be part of the leaf, indicating that the issue lies in the decoder implementation rather than in the leaf generation.

Impact

The `claim_redeem` functionality may be uncallable due to the incorrect decoder implementation.

Code Snippet

starknet_vault_kit_decoder_and_sanitizer.cairo#L38

Tool Used

Manual Review

Recommendation

Correct the `claim_redeem(...)` implementation in the decoder to align with leaf construction, ensuring that the ID is **not** serialised within the leaf and can be properly proofed.

Discussion

0xSacha

Valid, just sent am update to the svk https://github.com/ForgeYields/starknet_vault_kit/commit/16723695e39438ac8fcd1a102b0631b6f98b8d2a

talfao

We decreased severity as we realised that anyone can call claim

talfao

Fixed correctly as ID was deleted

Issue L-6: Misspelled storage variable name [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/41

Summary

The storage variable `protcol_fees_bps` is misspelt.

Vulnerability Detail

One of the storage variables `protcol_fees_bps` of `vault_governor` contract is misspelt, it should be `protocol_fees_bps`.

```
#[storage]
struct Storage {
     //...
     protcol_fees_bps: u256,
}
```

Impact

Reduces code readability, but this finding serves as only a general code practice improvement.

Code Snippet

vault_governor.cairo#89

Tool Used

Manual Review

Recommendation

Correct the spelling of the storage variable from `protcol_fees_bps` to `protocol_fees_bps`.

Discussion

0xSacha

Valid

https://github.com/vesuxyz/vesu-vaults/commit/7ba3192a76989019a9b477ccba5cfb591fe4e58f

talfao

Fixed properly

Issue L-7: Unsafe one-step ownership transfer for Vault curator [FIXED]

Source: https://github.com/sherlock-audit/2025-09-vesu-vaults-periphery-contracts-sept-24th/issues/42

Summary

The `VaultGovernor` contract implements a risky one-step process to set the vault curator. This can lead to the permanent loss of curator privileges if the new curator's address is set incorrectly.

Vulnerability Detail

The function `set_vault_curator(...)` allows the current curator to immediately and irreversibly transfer their role to a new address in a single transaction.

```
fn set_vault_curator(ref self: ContractState, new_vault_curator: ContractAddress) {
    self._assert_vault_curator();
    let old_vault_curator = self.vault_curator.read();
    self.vault_curator.write(new_vault_curator);
    self.emit(VaultCuratorSet { old_vault_curator, new_vault_curator });
}
```

If the `new_vault_curator` address is mistyped or set to an invalid address, the curator role will be lost forever.

Impact

- Permanent loss of vault curator privileges if the address is set incorrectly.
- No recovery mechanism if transferred to an invalid address.

Code Snippet

vault_governor.cairo#L463

Tool Used

Manual Review

Recommendation

Implements a two step vault curator transfer mechanism similar to `VesuV2` curator transfer pattern. Which involves :

1. The current vault curator nominates a new address for the curator role.
2. The new vault curator must explicitly accept the role in a separate transaction.

Discussion

0xSacha

Agree! here is the commit fix

https://github.com/vesuxyz/vesu-vaults/commit/e5ef5ea7b897cfe8503803cc3b405e474891d6ca

talfao

Looks good

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.