THREE SIGMA

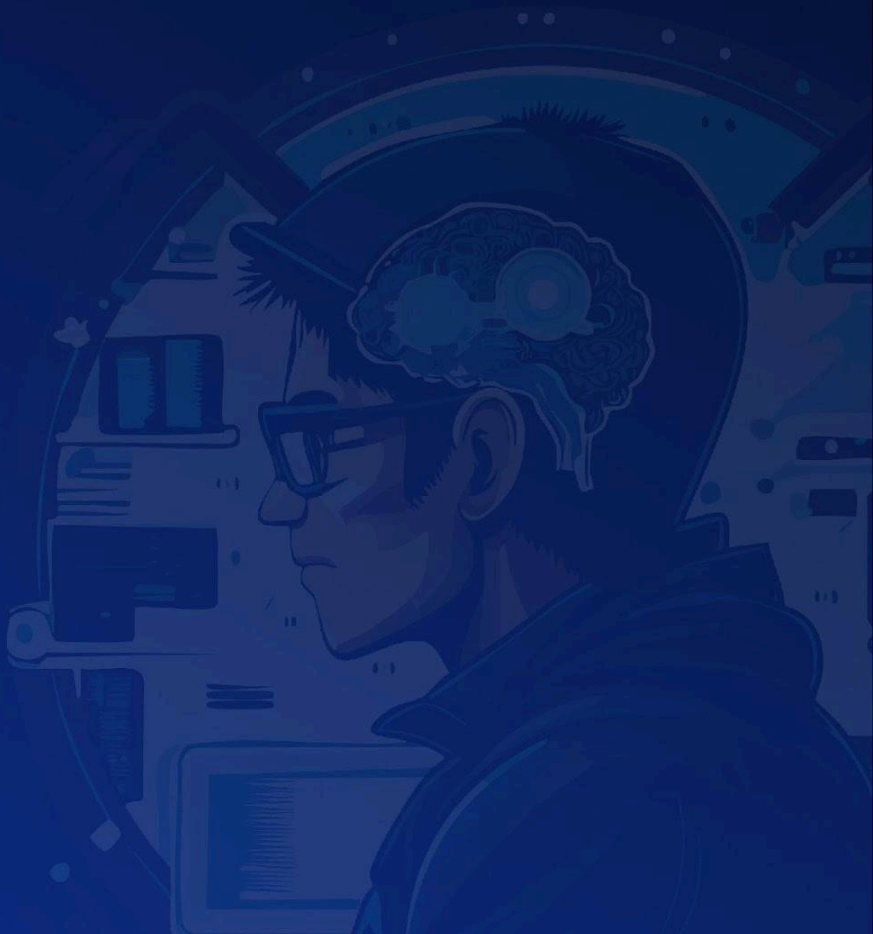
Abyss Vaults

Security Review

Disclaimer

Security Review

Abyss Vaults



Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Security Review

Abyss Vaults

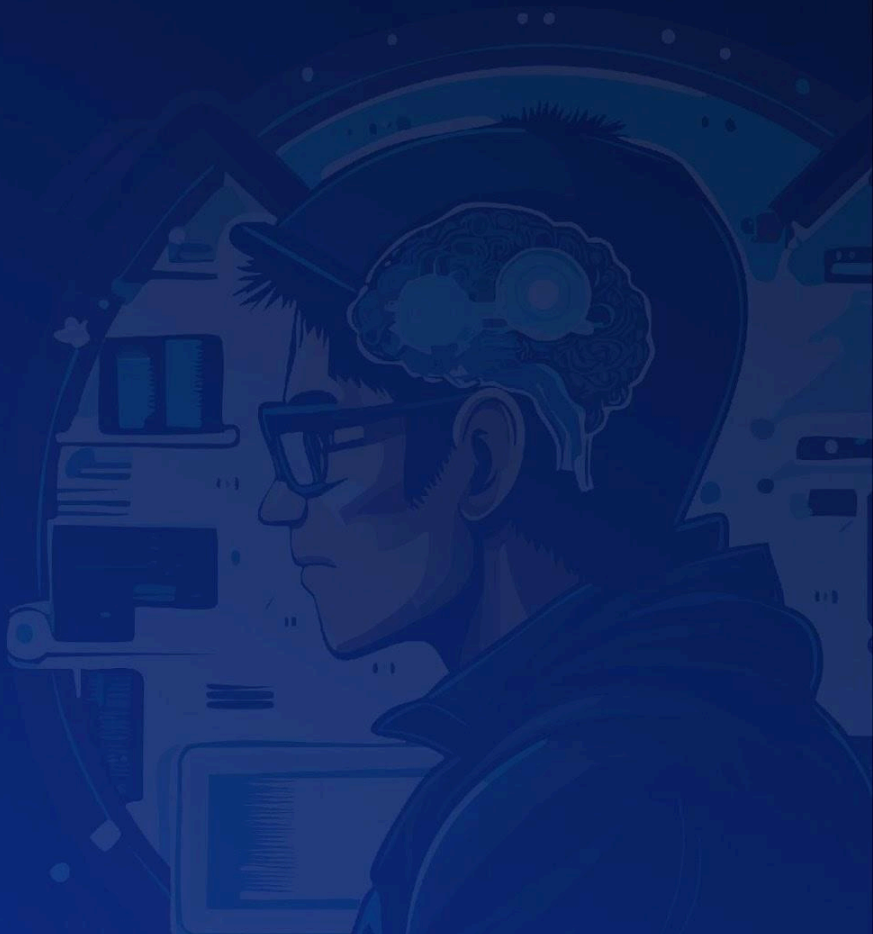
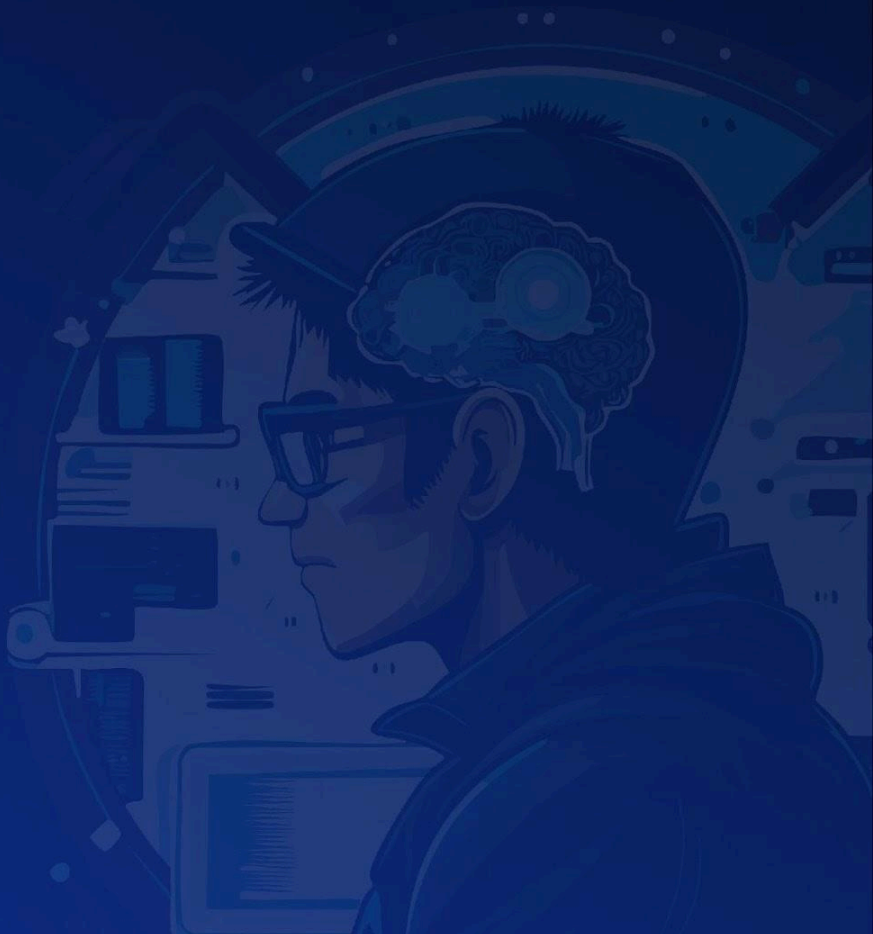


Table of Contents

Summary

Security Review

Abyss Vaults



Summary

Three Sigma audited Abyss in a 1.2 person week engagement. The audit was conducted from 10/11/2025 to 12/11/2025.

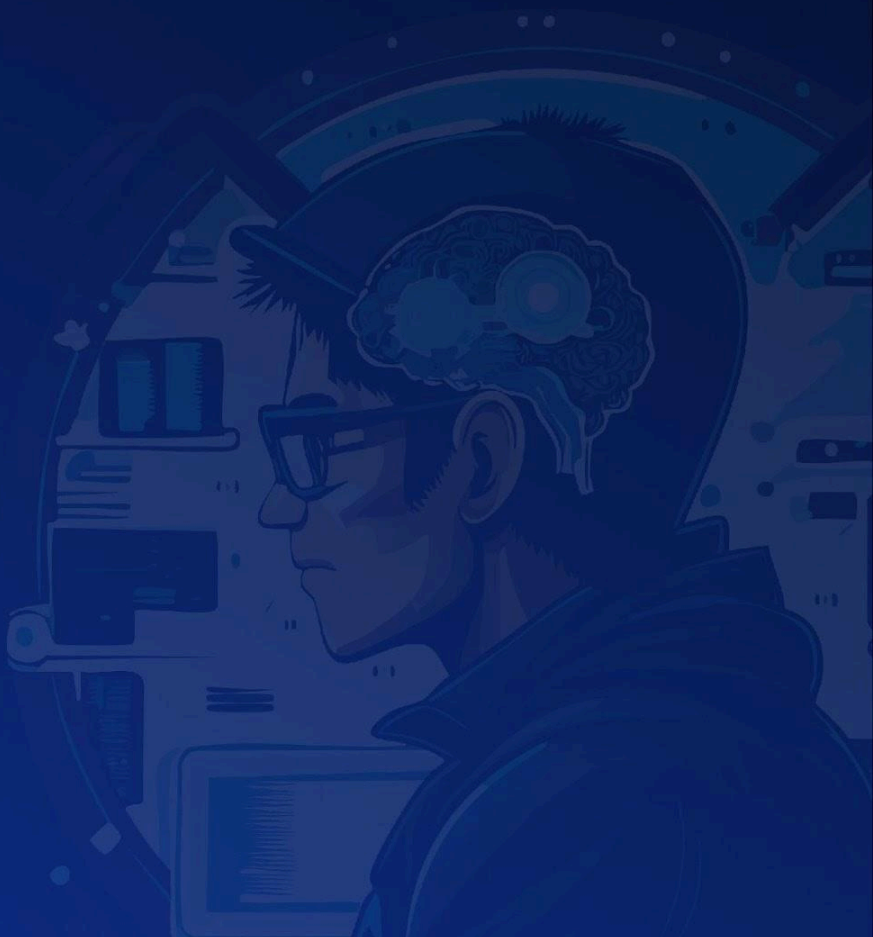
Protocol Description

Abyss is a liquidity venue built on Sui, providing spot and margin trading alongside liquidity-provision vaults. Its design focuses on efficient execution and accessible liquidity, creating a streamlined environment for both traders and LPs.

Scope

Security Review

Abyss Vaults



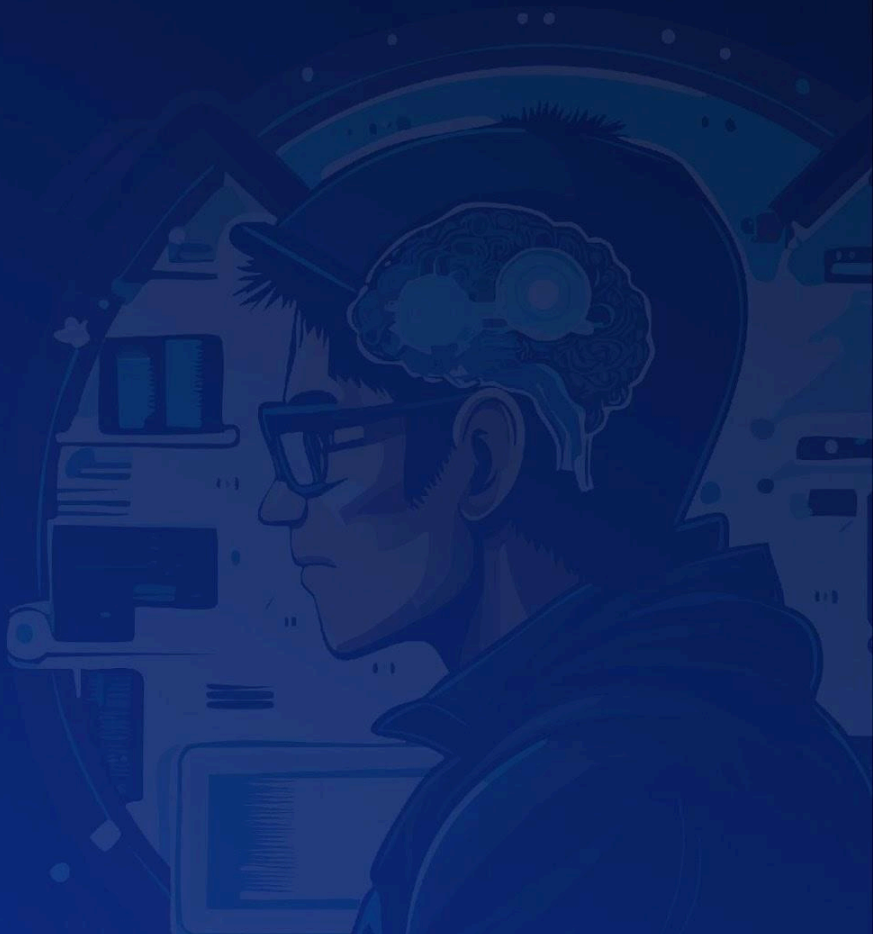
Scope

Filepath	nSLOC
sources/vault.move	542
sources/vault_registry.move	176
sources/fee_manager.move	40
sources/constants.move	25
sources/protocol_config.move	22
Total	**805**

Methodology

Security Review

Abyss Vaults



Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

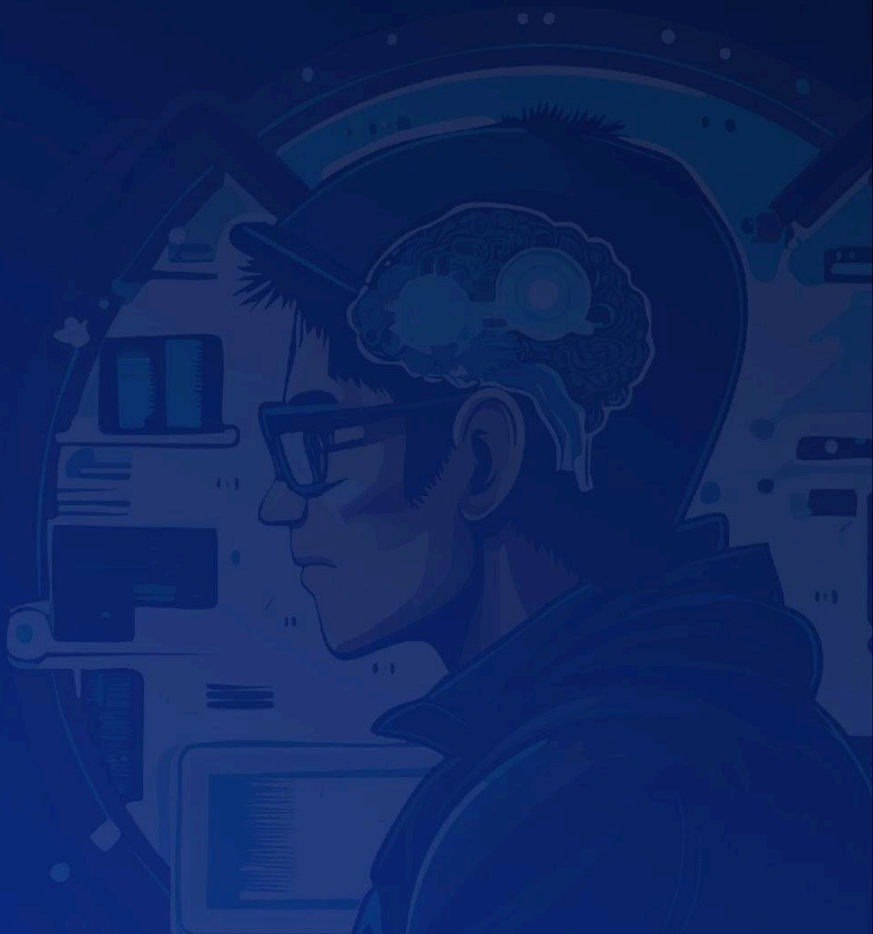
In this audit, we classify findings based on Immunefi's Vulnerability Severity Classification System (v2.3) as a guideline. The final classification considers both the potential impact of an issue, as defined in the referenced system, and its likelihood of being exploited. The following table summarizes the general expected classification according to impact and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Impact / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Security Review

Abyss Vaults



Project Dashboard

Application Summary

Name	Abyss
Repository	https://github.com/abyss-protocol/abyss
Commit	39a575b
Language	Move
Platform	Sui

Engagement Summary

Timeline	10/11/2025 to 12/11/2025
№ of Auditors	2
Review Time	1.2 person weeks

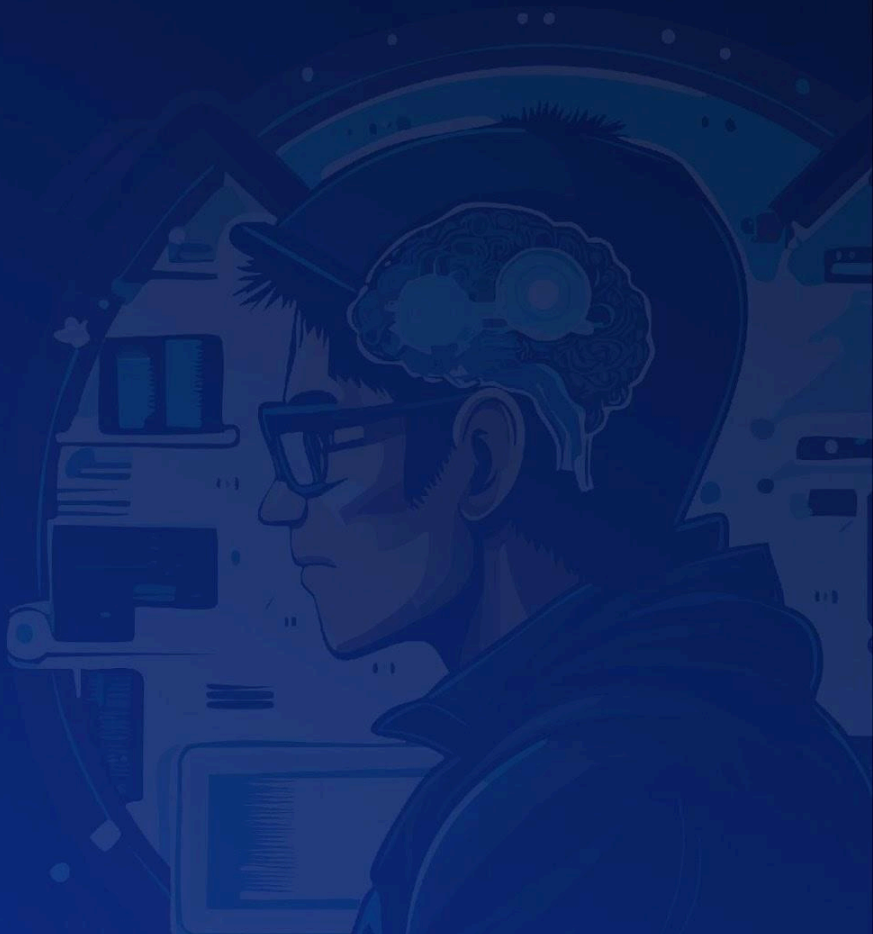
Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	2	2	0
None	2	2	0

Category Breakdown

Suggestion	2
Documentation	0
Bug	2
Optimization	0
Good Code Practices	0

Risk Section
Security Review
Abyss Vaults



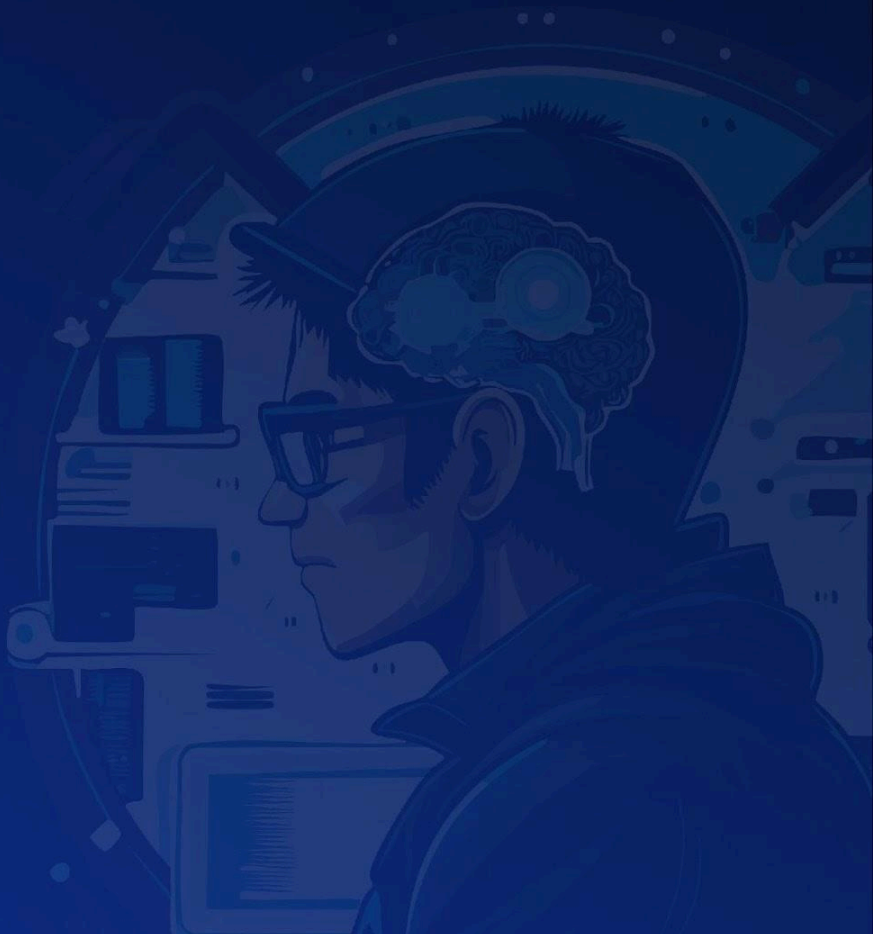
Risk Section

No risks were identified.

Findings

Security Review

Abyss Vaults



Findings

3S-Abyss-L01

Inconsistent share calculation order may lead to slight accounting discrepancies

Id	3S-Abyss-L01
Classification	Low
Impact	Low
Likelihood	Low
Category	Bug
Status	Addressed in #efac5a5.

Description

In Abyss, the calculation order of **mp_shares** is inconsistent with the calculation order of **shares** in **margin_pool**.

For example, in the `vault.deposit()` function, **mp_shares** to be added is calculated in the following order in the `vault.asset_to_mp_shares()` function:

```
math::mul(
    asset_amount,
    math::div(total_shares, total_supply),
)
```

However, **MarginPool** uses a different order here:

```
math::div(
    asset_amount,
    math::div(total_supply, total_shares),
)
```

At the same time, the calculation method in `vault.withdraw()` differs even more from how **shares** are reduced in **MarginPool** withdrawals.

Although they are mathematically equivalent, in actual computation, due to rounding issues, the **shares** calculated using different methods may vary slightly. This can lead to minor discrepancies between **margin_pool_shares** in the Abyss vault and the actual shares in the MarginPool.

Recommendation

It is recommended to calculate the exact change in shares by taking the difference in **supplier_cap**'s shares before and after performing **margin_pool.supply()** and **margin_pool.withdraw()** operations.

3S-Abyss-L02

Stale exchange rate in withdrawal causes vault share accounting drift

Id	3S-Abyss-L02
Classification	Low
Impact	Low
Likelihood	High
Category	Bug
Status	Addressed in #e3006d6.

Description

The `vault.withdraw()` function calculates margin pool shares to subtract using a **pre-withdrawal exchange rate**, but DeepBook's `margin_pool.withdraw()` accrues interest before burning shares, changing the rate. The vault then subtracts the stale pre-calculated amount, creating accounting drift.

if you look into **abyss/sources/vault.move**, function withdraw it calculate shares using old ratio (pre-accrual) but deepbook accrues interest inside withdraw(), and then we subtract old shares which is not correct after interest accrual. Sso the vault state is updated using stale pre-calculated amount.

```
// Calculate shares using old ratio (pre-accrual)
let assets_withdraw_amount = vault.convert_to_assets(margin_pool, withdraw_amount);
let margin_pool_shares_withdraw_amount =
vault.atoken_amount_to_mp_share(withdraw_amount);
// DeepBook accrues interest INSIDE withdraw()
let asset = margin_pool.withdraw(margin_registry, supplier_cap,
option::some(assets_withdraw_amount), clock, ctx);
// Subtract old shares (now incorrect after interest accrued)
vault.abyss_vault_state.margin_pool_shares =
vault.abyss_vault_state.margin_pool_shares - margin_pool_shares_withdraw_amount;
```

Inside DeepBook's `decrease_supply_shares()`, the `update()` function accrues interest, modifying **total_supply** and changing the exchange rate **after** vault's calculation.

One concrete impact is that vault progressively under-counts its real position in deepbook, causing stranded value, mispriced conversions, diluted depositors, and unreliable asset reporting that accumulates with every withdrawal.

Additionally, users receive slightly less assets (~0.01%) than their aTokens should represent because **convert_to_assets()** uses the pre-interest exchange rate.

Recommendation

The fix requires two steps to address both the user underpayment & vault accounting issues. First, pre-calculate any pending interest and apply it to convert_to_assets before the withdrawal, ensuring users receive a fair payout based on the current exchange rate. Second, after the withdrawal executes, recalculate the actual shares burned by querying the supplier position before and after the operation, then update the vault's accounting with this accurate value to prevent drift.

3S-Abyss-N01

The **mint_supplier_cap** function lacks access control

Id	3S-Abyss-N01
Classification	None
Category	Suggestion
Status	Addressed in #dd89607.

Description

`vault_registry.mint_supplier_cap()` is used to mint **supplier_cap** and wrap it in **AbyssSupplierCap**. It is used in **vault.deposit()** and **vault.withdraw()** to operate on the funds under the position represented by **supplier_cap**.

The comment above this function indicates "Public Admin Functions," but this function lacks access control.

```
// === Public Admin Functions ===
public fun mint_supplier_cap(margin_registry: &MarginRegistry, clock: &Clock, ctx: &mut TxContext) {
```

The absence of access control here allows anyone to create a new **AbyssSupplierCap** for deposits, which could result in vault funds not being concentrated under the intended cap.

Recommendation

It is recommended to add access control to this function, allowing only holders of **AbyssAdminCap** to call it.

3S-Abyss-N02

Incorrect parameter order in the **new_currency** function call

Id	3S-Abyss-N02
Classification	None
Category	Suggestion
Status	Addressed in #855ae21.

Description

The `vault.create_vault()` function calls `coin_registry.new_currency()` to create the AToken, but the order of arguments passed is incorrect.

```
let (currency_builder, treasury_cap) = coin_registry.new_currency<AToken<Asset>>(
    underlying_decimals,
    name,
    symbol,
    description,
    icon_url,
    ctx,
);
```

The call passes arguments as **underlying_decimals, name, symbol**, while the `coin_registry.new_currency()` function's actual parameter order is **underlying_decimals, symbol, name**.

Recommendation

Use the correct parameter order.

```
let (currency_builder, treasury_cap) = coin_registry.new_currency<AToken<Asset>>(
    underlying_decimals,
-   name,
+   symbol,
-   symbol,
+   name,
    description,
    icon_url,
```

```
    ctx,
);
```