FYEO

Secure Code Review Concordia

Concordia Systems Inc.

July 2023
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

List of Figures

List of Tables

EXECUTIVE SUMMARY

Overview

Concordia Systems Inc. engaged FYEO Inc. to perform a Secure Code Review Concordia.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on April 24 - May 19, 2023, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-CONCORDIA-ID-01 – The borrow & repay functionality works like lend & redeem
- FYEO-CONCORDIA-ID-02 – User can lend 300USDC, then redeem 300ETH
- FYEO-CONCORDIA-ID-03 – Any TCoin can be lend to TBroker but accounting is global across all TCoin
- FYEO-CONCORDIA-ID-04 – No way to recover in case the oracle key is compromised
- FYEO-CONCORDIA-ID-05 – Asset id is being used as collateral id
- FYEO-CONCORDIA-ID-06 – Deposit and loan note supplies are truncated to u64
- FYEO-CONCORDIA-ID-07 – Oracle price feed checks are insufficient
- FYEO-CONCORDIA-ID-08 – insert_collateral should check bounds, allows updating asset
- FYEO-CONCORDIA-ID-09 – Check non whitelisted TCoin using is_coin_initialized
- FYEO-CONCORDIA-ID-10 – Functions accepting amounts should fail early for 0 amount
- FYEO-CONCORDIA-ID-11 – No sanity check on admin input data for broker
- FYEO-CONCORDIA-ID-12 – The redeem function does not update the interest accrued
- FYEO-CONCORDIA-ID-13 – Unimplemented functionality
- FYEO-CONCORDIA-ID-14 – Code style and maintainability
- FYEO-CONCORDIA-ID-15 – Double check of condition, ignores 0 amount returned by first check
- FYEO-CONCORDIA-ID-16 – Lending parameters for broker can only be set during initialization
- FYEO-CONCORDIA-ID-17 – Near identical init functions
- FYEO-CONCORDIA-ID-18 – Some functions do not call price_updater
- FYEO-CONCORDIA-ID-19 – Superfluous function check_register

- FYEO-CONCORDIA-ID-20 – Superposition does not check if TBroker is part of its frontends
- FYEO-CONCORDIA-ID-21 – The function decrease_available_coins assumes there are always enough TCoin available
- FYEO-CONCORDIA-ID-22 – The liquidation_parameters should check against bounds
- FYEO-CONCORDIA-ID-23 – The transfer_revenue function should return early for 0 note amount
- FYEO-CONCORDIA-ID-24 – Using a raw value to identify a chain instead of a const

Based on our review process, we conclude that the reviewed code implements the documented functionality.

The re-review has verified that all the findings have been remediated.

SCOPE AND RULES OF ENGAGEMENT

The FYEO Review Team performed a Secure Code Review Concordia. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/concordia-fi/concordia with the commit hash dd9653686b5981cdf1d2f9fb8dc4eec89e42d574.

The re-review was done on the commit hash 58c31baae32e6b19dbe6e5c8222ef0845c74f00b.

Files included in the code review

```
concordia/
└── pkg/
    ├── concordia/
    │   └── sources/
    │       ├── asset_registry/
    │       │   ├── asset_registry.move
    │       │   └── asset_registry.spec.move
    │       ├── bribes/
    │       │   └── bribes.move
    │       ├── collateral/
    │       │   ├── collateral.move
    │       │   ├── collateral_api.move
    │       │   ├── collateral_baskets.move
    │       │   ├── collateral_baskets.spec.move
    │       │   └── collateral_manifest.move
    │       ├── cryptography/
    │       │   ├── elliptic.move
    │       │   └── elliptic.spec.move
    │       ├── entry/
    │       │   ├── accessor/
    │       │   │   └── portfolio_accesor.move
    │       │   ├── config.move
    │       │   ├── entry.move
```

Files included in the code review

```
│   │   ├── entry_acl.move
│   │   └── entry_admin.move
│   ├── health/
│   │   └── health.move
│   ├── lending/
│   │   ├── broker/
│   │   │   ├── specs/
│   │   │   │   ├── lending_broker.spec.move
│   │   │   │   ├── lending_broker_notes.spec.move
│   │   │   │   ├── lending_broker_params.spec.move
│   │   │   │   └── lending_interest_v1.spec.move
│   │   │   ├── lending_broker.move
│   │   │   ├── lending_broker_math.move
│   │   │   ├── lending_broker_notes.move
│   │   │   ├── lending_broker_params.move
│   │   │   └── lending_interest_v1.move
│   │   ├── frontend/
│   │   │   ├── specs/
│   │   │   │   └── lending_revenue_collector.spec.move
│   │   │   ├── lending_frontend.move
│   │   │   ├── lending_frontend_api.move
│   │   │   └── lending_revenue_collector.move
│   │   ├── lending_api.move
│   │   └── lending_broker_types.move
│   ├── liquidations/
│   │   ├── liquidation_parameters.move
│   │   ├── liquidation_params.spec.move
│   │   ├── liquidations.move
│   │   └── liquidations.spec.move
│   ├── portfolio/
│   │   ├── specs/
│   │   │   ├── portfolio.spec.move
│   │   │   ├── portfolio_positions.spec.move
│   │   │   └── portfolio_registry.spec.move
│   │   ├── portfolio.move
│   │   ├── portfolio_api.move
│   │   ├── portfolio_math.move
│   │   ├── portfolio_positions.move
│   │   └── portfolio_registry.move
│   ├── price/
│   │   ├── price_decoder.move
│   │   ├── price_decoder.spec.move
│   │   ├── price_store.move
```

Files included in the code review

```
│       │   └── price_updater.move
│       ├── profile/
│       │   ├── profile.move
│       │   ├── profile_api.move
│       │   └── profile_registry.move
│       ├── toolbox/
│       │   ├── interop/
│       │   │   ├── chains.move
│       │   │   ├── cursor.move
│       │   │   ├── deserialize.move
│       │   │   ├── deserialize.spec.move
│       │   │   └── serialize.move
│       │   ├── int_map.move
│       │   ├── int_map.spec.move
│       │   ├── interest_rate.move
│       │   ├── lite_iterable_table.move
│       │   ├── lite_iterable_table.spec.move
│       │   ├── ra.move
│       │   ├── safe_coin_supply.move
│       │   ├── seeder.move
│       │   ├── set.move
│       │   ├── set.spec.move
│       │   ├── vault.move
│       │   └── vault.spec.move
│       ├── core.move
│       ├── core.spec.move
│       └── test_coins.move
├── math/
│   └── sources/
│       ├── precise_number.move
│       └── precise_number.spec.move
└── superv2/
    └── sources/
        ├── super_api.move
        └── super_core.move
```

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Secure Code Review Concordia, we discovered:

- 2 findings with CRITICAL severity rating.
- 2 findings with HIGH severity rating.
- 4 findings with MEDIUM severity rating.
- 5 findings with LOW severity rating.
- 11 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

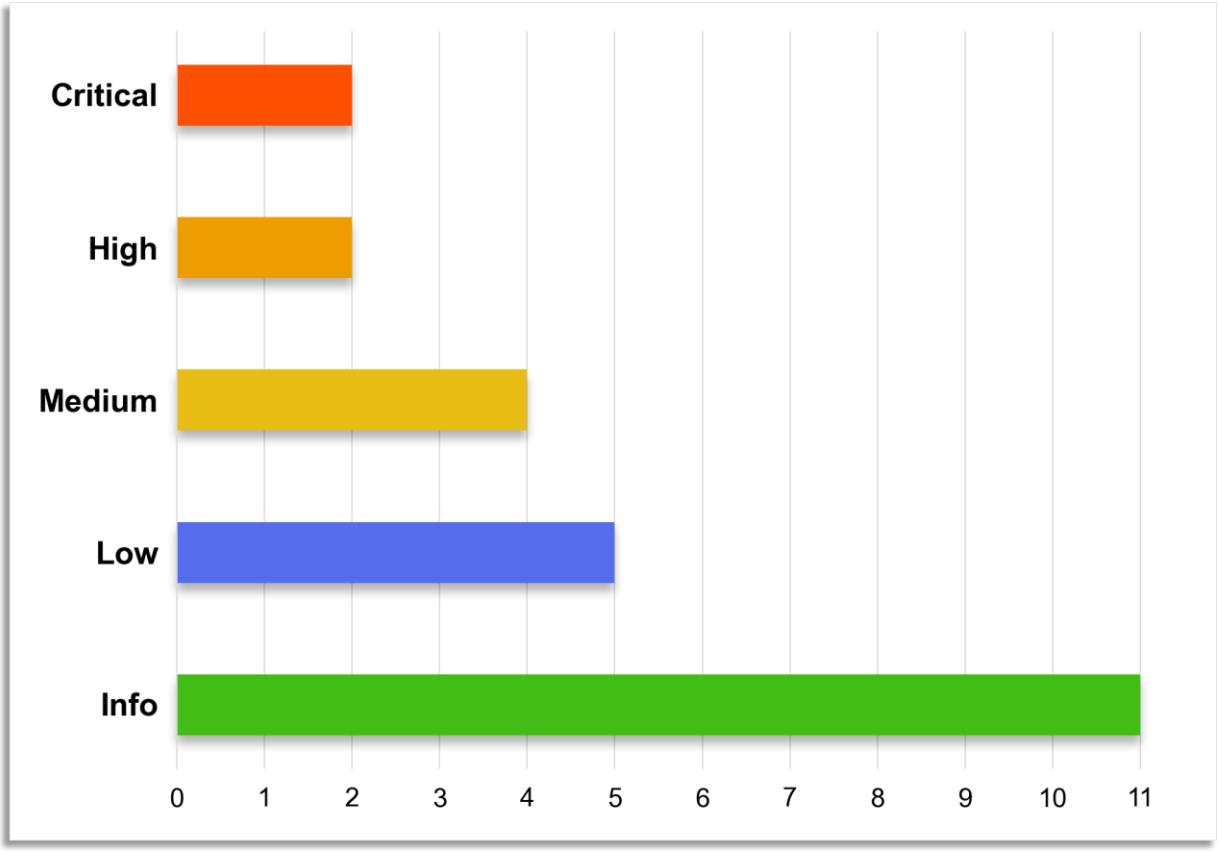


Figure 1: Findings by Severity

FINDINGS

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-CONCORDIA-ID-01	**Critical**	The borrow & repay functionality works like lend & redeem
FYEO-CONCORDIA-ID-02	**Critical**	User can lend 300USDC, then redeem 300ETH
FYEO-CONCORDIA-ID-03	**High**	Any TCoin can be lend to TBroker but accounting is global across all TCoin
FYEO-CONCORDIA-ID-04	**High**	No way to recover in case the oracle key is compromised
FYEO-CONCORDIA-ID-05	**Medium**	Asset id is being used as collateral id
FYEO-CONCORDIA-ID-06	**Medium**	Deposit and loan note supplies are truncated to u64
FYEO-CONCORDIA-ID-07	**Medium**	Oracle price feed checks are insufficient
FYEO-CONCORDIA-ID-08	**Medium**	insert_collateral should check bounds, allows updating asset
FYEO-CONCORDIA-ID-09	**Low**	Check non whitelisted TCoin using is_coin_initialized
FYEO-CONCORDIA-ID-10	**Low**	Functions accepting amounts should fail early for 0 amount
FYEO-CONCORDIA-ID-11	**Low**	No sanity check on admin input data for broker
FYEO-CONCORDIA-ID-12	**Low**	The redeem function does not update the interest accrued
FYEO-CONCORDIA-ID-13	**Low**	Unimplemented functionality
FYEO-CONCORDIA-ID-14	**Informational**	Code style and maintainability
FYEO-CONCORDIA-ID-15	**Informational**	Double check of condition, ignores 0 amount returned by first check
FYEO-CONCORDIA-ID-16	**Informational**	Lending parameters for broker can only be set during initialization
FYEO-CONCORDIA-ID-17	**Informational**	Near identical init functions

FYEO-CONCORDIA-ID-18	**Informational**	Some functions do not call price_updater
FYEO-CONCORDIA-ID-19	**Informational**	Superfluous function check_register
FYEO-CONCORDIA-ID-20	**Informational**	Superposition does not check if TBroker is part of its frontends
FYEO-CONCORDIA-ID-21	**Informational**	The function decrease_available_coins assumes there are always enough TCoin available
FYEO-CONCORDIA-ID-22	**Informational**	The liquidation_parameters should check against bounds
FYEO-CONCORDIA-ID-23	**Informational**	The transfer_revenue function should return early for 0 note amount
FYEO-CONCORDIA-ID-24	**Informational**	Using a raw value to identify a chain instead of a const

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

The re-review has verified that the findings have all been remediated.

TECHNICAL FINDINGS

GENERAL OBSERVATIONS

Based on the code review, it is evident that the team has put in significant effort to create a well-organized code base that is easy to follow. The team was responsive, communicative and helped with any questions raised.

The Move prover specs have been consistently added throughout the code base. But they would benefit from expansion and more explicit instructions. For instance, if the function calculates something the spec could be explicit about the expected result.

It is great to see that integration tests have been considered a priority. They would still benefit from more complex test cases and especially tests against things that are expected to fail. Adding some random elements in the test could also help improve them, especially when checking calculations.

It was noted that the code style is a bit inconsistent and would benefit from more precise naming of variables as well as generics.

THE BORROW & REPAY FUNCTIONALITY WORKS LIKE LEND & REDEEM

Finding ID: FYEO-CONCORDIA-ID-01
Severity: **Critical**
Status: **Remediated**

Description

Any asset TCoin can be borrowed from TBroker in return for loan notes - these notes can be used to repay any TCoin at a global exchange rate.

Proof of Issue

File name: pkg/concordia/sources/lending/broker/lending_broker.move
Line number: 148

```
public(friend) fun borrow<TBroker, TCoin>(
    amount: u64
): (Coin<TCoin>, Coin<LoanNote<TBroker>>) acquires LendingBroker {
    let loan_notes_amount = loan_notes_from_raw_coins<TBroker>(
        amount,
        true, // round up
    );

    let loan_notes = mint_loan_notes<TBroker>(loan_notes_amount);
    let coins = extract_coins<TBroker, TCoin>(amount);
    increase_borrowed_coins<TBroker>(amount);
    decrease_available_coins<TBroker>(amount);
    ...
}
```

The amount of notes issued does not depend on the TCoin but just on TBroker and the amount of coins (u64).

Severity and Impact Summary

This allows theft by repaying with any asset worth less.

Recommendation

Implement accounting based on the asset type or based on USD values.

USER CAN LEND 300USDC, THEN REDEEM 300ETH

Finding ID: FYEO-CONCORDIA-ID-02
Severity: **Critical**
Status: **Remediated**

Description

Any asset TCoin can be lend to TBroker in return for deposit notes - these notes can be used to redeem any TCoin at a global exchange rate. If a user Bob has lend 300 WEth to broker Charlie, Alice can lend 300 USDC to Charlie and redeem 300WEth.

Proof of Issue

File name: pkg/concordia/sources/lending/broker/lending_broker.move
Line number: 206

```
public(friend) fun lend<TBroker, TCoin>(
    coins: Coin<TCoin>
): Coin<DepositNote<TBroker>> acquires LendingBroker {
    // Get dnote amount, rounding down
    let notes_amount = deposit_notes_from_raw_coins<TBroker>(
        coin::value(&coins),
        false
    );

    ...

    mint_deposit_notes<TBroker>(notes_amount)
}
```

The amount of notes issued does not depend on the TCoin but just on TBroker and the amount of coins (u64).

Severity and Impact Summary

This allows theft of any asset lend to a broker.

Recommendation

Implement accounting based on the asset type or based on USD values.

Test case

This test case was placed just before `it('removes deposit notes as collateral', async () => {`

```
it('inserts a WETH as a 2nd asset', async () => {
  const b1 = await concordiaClient.fetcher.broker(tBroker)
  expect(b1.availableCoins).to.eql(430)

  const depositNoteAfter = await concordiaClient.fetcher.note(
    rootAccount.address().toString(),
    tBroker,
```

```
    depNoteType
  )
  expect(depositNoteAfter.value).to.eql(390)

  // Setup wETH asset
  const wEthAssetId = 111
  await registerAsset(wEthAssetId, WETHType)
  await insertCollateral(wEthAssetId)

  const registeredAsset = await asset(wEthAssetId)
  expect(registeredAsset.assetId).to.eql(wEthAssetId)

  // Set price for wETH price
  await setTestPrice(wEthAssetId, 20000n)
  const wETHPrice = await price(wEthAssetId)
  expect(wETHPrice.price).to.eql(20000n)

  // Check collateral manifest
  const manifest = await concordiaClient.fetcher.collateralManifest()
  const cPreview: ICollateral = {
    assetId: wEthAssetId,
    haircutBps: 100,
    maxGlobalDeposit: 1000,
    maxGlobalDepositUsd: 10000,
    maxPortfolioDeposit: 999,
    maxPortfolioDepositUsd: 9999
  }
  const wEthCollateralId = manifest.highestID
  const collateral = await concordiaClient.fetcher.collateral(manifest.handle,
wEthCollateralId)
  expect(collateral).to.deep.equal(cPreview)
  console.log('Collateral wETH added with id ', wEthCollateralId, ' data: ',
collateral)

  // Add wETH to basket
  const cb = await basket()
  const basketIx = concordiaClient.updateBasketIX(cb.basketId, ['' +
wEthCollateralId], [])
  await signSubmitWait(aptosClient, rootAccount, basketIx)

  // Mint some wETH coins, add them as collateral
  const portAddress = await portfolioAddress()
  await mintCoins(aptosClient, rootAccount, rootAccount.address().hex(), WETHType,
2000)
  const ix1 = concordiaClient.addCollateralIX(portAddress, 1000, WETHType)
  await signSubmitWait(aptosClient, rootAccount, ix1)
  const { values } = await activeCollaterals()

  // Check collateral amounts in portfolio
  expect(values[0]).to.eql(990)
  expect(values[1]).to.eql(100)
  expect(values[2]).to.eql(1000)


  // Lend some wETH
  const amount = 500
  const ix = concordiaClient.lendIX(tBroker, WETHType, amount)
```

```
  await signSubmitWait(aptosClient, rootAccount, ix)
  const balanceAfter = await checkBalance(WETHType)
  expect(balanceAfter).to.eql(500)

  const b = await concordiaClient.fetcher.broker(tBroker)
  expect(b.availableCoins).to.eql(930) // 430 + 500 = 930
  const depositNote = await concordiaClient.fetcher.note(
    rootAccount.address().toString(),
    tBroker,
    depNoteType
  )
  expect(depositNote.value).to.eql(880) // @audit was 390 above line 359



  // Add a 2nd user to act as a lender of USDType
  const user2Account = await createAndFund(faucetClient)

  const ix20 = concordiaClient.initProfileIX()
  await signSubmitWait(aptosClient, user2Account, ix20)

  const ix21 = concordiaClient.initPortfolioIX(cb.basketId)
  await signSubmitWait(aptosClient, user2Account, ix21)

  const user2Address = user2Account.address().toString()
  const profile = await concordiaClient.fetcher.profile(user2Address)
  const user2PortAddress = profile.portfolios[0]

  await registerCoin(user2Account, aptosClient, USDCType)
  await registerCoin(user2Account, aptosClient, WETHType)
  await mintCoins(aptosClient, rootAccount, user2Account.address().hex(), USDCType,
2000)

  // const ix22 = concordiaClient.addCollateralIX(user2PortAddress, 777, USDCType)
  // await signSubmitWait(aptosClient, user2Account, ix22)

  const ix23 = concordiaClient.lendIX(tBroker, USDCType, 333)
  await signSubmitWait(aptosClient, user2Account, ix23)

  // Check balances
  const b2 = await concordiaClient.fetcher.broker(tBroker)
  expect(b2.availableCoins).to.eql(1263) // @audit 430 + 500 = 930 + 333 = 1263

  const depositNote2 = await concordiaClient.fetcher.note(
    user2Account.address().toString(),
    tBroker,
    depNoteType
  )
  expect(depositNote2.value).to.eql(326) //@audit check math on 326


  const coinClient = new CoinClient(aptosClient)
  const user2BalanceA = await coinClient.checkBalance(user2Account, { coinType:
WETHType })
  expect(parseInt(user2BalanceA.toString())).to.eq(0)

  // The user2Account instead redeems WETH for their deposit notes
```

```
  const ix24 = concordiaClient.redeemIX(tBroker, WETHType, 326)
  await signSubmitWait(aptosClient, user2Account, ix24)


  // Check user balance
  const user2BalanceB = await coinClient.checkBalance(user2Account, { coinType:
WETHType })
  expect(parseInt(user2BalanceB.toString())).to.eq(332)
})
```

ANY TCOIN CAN BE LEND TO TBROKER BUT ACCOUNTING IS GLOBAL ACROSS ALL TCOIN

Finding ID: FYEO-CONCORDIA-ID-03
Severity: **High**
Status: **Remediated**

Description

The `lend` instruction lets the TX signer deposit any TCoin to any front-end TBroker whereas the interest rates and accounting are global for that TBroker.

Proof of Issue

File name: pkg/concordia/sources/entry/entry.move
Line number: 46

```
public entry fun lend<TBroker, TCoin>(
    user: &signer,
    amount: u64
) {
    let coins = coin::withdraw<TCoin>(user, amount);
    let dnotes = core::lend<TBroker, TCoin>(coins);
    coin::register<DepositNote<TBroker>>(user);
    coin::deposit(signer::address_of(user), dnotes);
}
```

Severity and Impact Summary

Any asset can be deposited, there is no accounting of who deposited what and interest rates are managed in a global count.

Recommendation

Implement checks on TCoin and adjust interest calculation to be asset specific?

NO WAY TO RECOVER IN CASE THE ORACLE KEY IS COMPROMISED

Finding ID: FYEO-CONCORDIA-ID-04
Severity: **High**
Status: **Remediated**

Description

There is an init function `init_price_updater` for the price oracle key - there is however no way to change / update the key - a single key is a single point of failure. A way to change the key may be important.

Proof of Issue

There is no way to modify the oracle key stored.

Severity and Impact Summary

If the key is compromised and can not be changed, the attacker can pretty much steal assets by setting any price they want.

Recommendation

Ensure key safety and add appropriate fail safes.

ASSET ID IS BEING USED AS COLLATERAL ID

Finding ID: FYEO-CONCORDIA-ID-05
Severity: **Medium**
Status: **Remediated**

Description

The `process_liquidation` function assigns `collat_id` from `asset_registry::asset_id_from_type<Collat>()`.
It then calls `check_payment_ratio` which calls `collat::get_haircut_by_collat_id(collat_id)` - this functions appears to expect a collateral_id because it borrows `im::borrow(&cs.manifest, collat_id)`.

Later on in `handle_liquidation_transfers` the same id is used to `pm::usd_into_amount(collat_id, collat_transfer_value)`, where it is used as an asset_id.

Proof of Issue

File name: pkg/concordia/sources/liquidations/liquidations.move
Line number: 68

```
let collat_id = asset_registry::asset_id_from_type<Collat>(); // Note the collat_id is the asset_id of the collateral. Not to be confused with the collateral_id from the collateral_manifest.

...

fun check_payment_ratio(collat_transfer_usd: math::PreciseNumber, liab_transfer_usd: math::PreciseNumber, collat_id: u64) {
    ...
    let haircut = collat::get_haircut_by_collat_id(collat_id);
    ...
}

...

fun handle_liquidation_transfers<Collat, Debt, Broker>(
    liqee_port_addr: address,
    liqor_port_addr: address,
    _basket_id: u64,
    collat_id: u64,
    debt_id: u64,
    collat_transfer_value: math::PreciseNumber,
    liab_transfer_value: math::PreciseNumber
) {
    ...
    let collat_transfer_amount =  pm::usd_into_amount(collat_id, collat_transfer_value);
    ...
}
```

Severity and Impact Summary

The lookup would likely fail as this is an asset_id - not a collateral_id.

Recommendation

The variable should indicate that it is an asset_id - maybe call it collateral_asset_id.
Make sure this is correctly implemented. And consider wrapping asset_id and collateral_id in specific types so there can not be any confusion as the compiler would prevent it.

DEPOSIT AND LOAN NOTE SUPPLIES ARE TRUNCATED TO U64

Finding ID: FYEO-CONCORDIA-ID-06
Severity: **Medium**
Status: **Remediated**

Description

Both loan and deposit note supplies are u128 in Aptos, yet are truncated in the supply function wrapper. While Move will cause an error should that cast overflow, it would still render parts of the contract unusable and lock user funds.

Proof of Issue

File name: pkg/concordia/sources/toolbox/safe_coin_supply.move
Line number: 6

```
public fun supply<T>(): u64 {
    let s = coin::supply<T>();

    if (!option::is_some(&s)) {
        0
    } else {
        // TODO - check u128 overflow
        (option::destroy_some(s) as u64)
    }
}
```

Severity and Impact Summary

If a brokers supply of loan or deposit notes were to exceed any instruction calling the supply function will run into an error making it impossible to run certain transactions.

Recommendation

Leave the supply as a u128.

ORACLE PRICE FEED CHECKS ARE INSUFFICIENT

Finding ID: FYEO-CONCORDIA-ID-07
Severity: **Medium**
Status: **Remediated**

Description

The way the oracle price feed, used i.e. in remove_collateral, works with `price_updater::update_prices(oracle)` is not ideal. There is no check of the timestamp in the header - the user supplied oracle message can therefore be any old message send by any previous request.

While there are extra timestamps per asset - those just skip the price update for that asset. There could be some coin that is not often updated and if the user finds an old signed oracle message, the call will go through without updating the potentially old and outdated asset price.

Since this data is user input it would also be possible to add a specific user portfolio id in the signed message to make this more specific - that way a user can not use some signed price feed from someone else.

There is also no check what assets (or if any) are updated in the feed.
What would be the point of updating the BTC price if the whole portfolio is based on ETH? There should be a check that this feed is relevant to the assets actually inside the portfolio / relevant to the transaction.

Proof of Issue

File name: pkg/concordia/sources/price/price_decoder.move
Line number: 79

```
public(friend) fun decode(
    price_signer: elliptic::ValidatedPublicKey,
    update_payload: vector<u8>): vector<PriceUpdate> {
    // Parse header
    let header_cursor = cursor::init(update_payload);

    let body_cursor = validate_payload(price_signer,
        header_cursor);

    //Payload type and creation time not used yet
    let payload_type = deserialize::deserialize_u64(&mut body_cursor);
    assert!(payload_type == 0, ERR_PAYLOAD_NOT_PRICE_DATA);
    let _creation_time = deserialize::deserialize_u64(&mut body_cursor);

    let length = deserialize::deserialize_u64(&mut body_cursor);
    assert_payload_size(&update_payload, length);

    decode_price_vector(body_cursor, length)
}
```

Severity and Impact Summary

This oracle price feed, as currently implemented, can be thought of as optional. There is no guarantee that anything of importance is actually updated.

Recommendation

Make sure that assets relevant to the transaction have an updated price.

INSERT_COLLATERAL SHOULD CHECK BOUNDS, ALLOWS UPDATING ASSET

Finding ID: FYEO-CONCORDIA-ID-08
Severity: **Medium**
Status: **Remediated**

Description

The `insert_collateral` function does not check values against bounds. It also appears to be meant to insert but as it eventually calls an upsert function it could be used to update. When using it for updating it would call `let collat_id = im::insert(&mut cs.manifest, c);` assigning a new id.

Proof of Issue

File name: pkg/concordia/sources/collateral/collateral.move
Line number: 20

```
public(friend) fun make_collateral(
    asset_id: u64,
    haircut_bps: u64,
    max_global_deposit: u64,
    max_global_deposit_usd: u64,
    max_portfolio_deposit: u64,
    max_portfolio_deposit_usd: u64,
): Collateral {
    return Collateral {
        asset_id,
        haircut_bps,
        max_global_deposit,
        max_global_deposit_usd,
        max_portfolio_deposit,
        max_portfolio_deposit_usd
    }
}
```

No sanity check on values.

File name: pkg/concordia/sources/collateral/collateral_manifest.move
Line number: 37

```
public(friend) fun insert(c: collateral::Collateral) acquires Collaterals {
    assert!(exists<Collaterals>(@concordia), ERR_COLLATERAL_MANIFEST_DOES_NOT_EXIST);
    let cs = borrow_global_mut<Collaterals>(@concordia);
    let asset_id = collateral::get_asset_id(&c);// c = Collateral -> was created above
from user input
    let collat_id = im::insert(&mut cs.manifest, c);// insert generates a new ID,
adding data to the end of the table - there could be an identical asset in the table
    table::upsert(&mut cs.asset_id_to_collat_id, asset_id, collat_id);// overwrites
asset_id in the given map with the new collateral_id
}
```

As added in the comments for the code, this will overwrite `asset_id_to_collat_id` with a new `collat_id` should the `asset_id` already exist.

Severity and Impact Summary

This could cause several `collat_id` to exist for one `asset_id`.

Recommendation

Have a separate update function and check during insert that there is no entry in `cs.asset_id_to_collat_id`.

CHECK NON WHITELISTED TCOIN USING IS_COIN_INITIALIZED

Finding ID: FYEO-CONCORDIA-ID-09
Severity: **Low**
Status: **Remediated**

Description

Functions taking TCoin(s) should check that coin::is_coin_initialized(). This is less necessary for the functions checking against a whitelist. But may also be beneficial when adding assets to the whitelist to avoid potential mistakes.

Proof of Issue

File name: pkg/concordia/sources/entry/entry.move
Line number: 46

```
public entry fun lend<TBroker, TCoin>(
    user: &signer,
    amount: u64
) {
    let coins = coin::withdraw<TCoin>(user, amount);
    let dnotes = core::lend<TBroker, TCoin>(coins);
    coin::register<DepositNote<TBroker>>(user);
    coin::deposit(signer::address_of(user), dnotes);
}
```

Severity and Impact Summary

This verifies that TCoin is an actual registered coin.

```
public fun is_coin_initialized<CoinType>(): bool {
    exists<CoinInfo<CoinType>>(coin_address<CoinType>())
}
```

Recommendation

Add this check where required.

FUNCTIONS ACCEPTING AMOUNTS SHOULD FAIL EARLY FOR 0 AMOUNT

Finding ID: FYEO-CONCORDIA-ID-10
Severity: **Low**
Status: **Remediated**

Description

Most functions accepting amounts accept 0 amounts.
Borrowing TCoin with 0 amount causes a ERR_DIV_BY_ZERO for error clarity it would be better to fail early.
Redeem causes a EZERO_COIN_AMOUNT when attempting to burn 0 coins. Lend works with 0 amount, doing nothing of use.

Proof of Issue

File name: pkg/concordia/sources/entry/entry.move
Line number: 46

```
public entry fun lend<TBroker, TCoin>(
    user: &signer,
    amount: u64
) {
    let coins = coin::withdraw<TCoin>(user, amount);
    let dnotes = core::lend<TBroker, TCoin>(coins);
    coin::register<DepositNote<TBroker>>(user);
    coin::deposit(signer::address_of(user), dnotes);
}
```

As well as many other `entry` functions accepting amounts.

Severity and Impact Summary

Errors might be unintelligible, gas may be wasted and errors may be introduced when issuing something of value for 0 amounts.

Recommendation

For the sake of error clarity and avoiding any bad behavior, amounts should be checked to be greater than 0.

NO SANITY CHECK ON ADMIN INPUT DATA FOR BROKER

Finding ID: FYEO-CONCORDIA-ID-11
Severity: **Low**
Status: **Remediated**

Description

The function `lending_interest_v1::upsert` stores some critical values but there is no check if these parameters make any sense. The range on some of these parameters is rather specific and, even though this is an admin function, should be checked.

Proof of Issue

File name: pkg/concordia/sources/lending/broker/lending_interest_v1.move
Line number: 50

```
move_to(root, InterestRateParams<TBroker> {
    u1, u2, r0, r1, r2, r3
});
```

Severity and Impact Summary

Setting completely out of bound values, by mistake or with malicious intend, could cause severe problems.

Recommendation

Values that have to be within a range should be checked to be so.

THE REDEEM FUNCTION DOES NOT UPDATE THE INTEREST ACCRUED

Finding ID: FYEO-CONCORDIA-ID-12
Severity: **Low**
Status: **Remediated**

Description

The `fun redeem<TBroker, TCoin>(...)` in `lending_api.move` does not update the interest accrued. Yet the lend function does.

Proof of Issue

File name: pkg/concordia/sources/lending/lending_api.move
Line number: 36

```
public(friend) fun redeem<TBroker, TCoin>(
    deposit_notes: Coin<DepositNote<TBroker>>
): Coin<TCoin> {
    broker::redeem<TBroker, TCoin>(deposit_notes)
}

public(friend) fun lend<TBroker, TCoin>(
    coins: Coin<TCoin>
): Coin<DepositNote<TBroker>> {
    broker::accrue_interest<TBroker>();
    broker::lend<TBroker, TCoin>(coins)
}
```

Severity and Impact Summary

It seems that the redeem function should be calling `broker::accrue_interest<TBroker>()` but does not.

Recommendation

Verify this is as intended.

UNIMPLEMENTED FUNCTIONALITY

Finding ID: FYEO-CONCORDIA-ID-13
Severity: **Low**
Status: **Remediated**

Description

This functionality has not yet been implemented.

Proof of Issue

File name: pkg/concordia/sources/core.move
Line number: 225

```
fun assert_subaccount_broker_match_basket<TBroker>(_portfolio: address) {
    // TODO
    assert!(true, 1);
}
```

File name: pkg/concordia/sources/lending/broker/lending_broker.move
Line number: 162

```
public(friend) fun borrow<TBroker, TCoin>(
    amount: u64
): (Coin<TCoin>, Coin<LoanNote<TBroker>>) acquires LendingBroker {
    let loan_notes_amount = loan_notes_from_raw_coins<TBroker>(
        amount,
        true, // round up
    );

    let loan_notes = mint_loan_notes<TBroker>(loan_notes_amount);
    let coins = extract_coins<TBroker, TCoin>(amount);
    increase_borrowed_coins<TBroker>(amount);
    decrease_available_coins<TBroker>(amount);

    // assert_limits

    (coins, loan_notes)
}
```

There is no implementation to `assert_limits`.

Severity and Impact Summary

This functionality is important to the functionality of the contract.

Recommendation

Implement this function.

CODE STYLE AND MAINTAINABILITY

Finding ID: FYEO-CONCORDIA-ID-14
Severity: **Informational**
Status: **Remediated**

Description

Some suggestions to optimize the code style.

Proof of Issue

File name: pkg/concordia/sources/portfolio/portfolio.move
Line number: 396

```
let cur_amount = *lit::borrow_mut_with_default(a_table, frontend_id, 0);

lit::set(
    a_table,
    frontend_id,
    cur_amount + amount
);
```

mutable borrow, deref and set

File name: pkg/concordia/sources/collateral/collateral_baskets.move
Line number: 137

```
public(friend) fun assert_asset_is_eligible_collateral(basket_id: u64, asset_id: u64)
acquires Baskets {
    ...
    let basket = table::borrow_mut(&mut baskets.baskets, basket_id);

    let collat_id = collateral_manifest::get_collateral_id(asset_id);
    assert!(set::contains(&basket.collaterals, collat_id),
ERR_ASSET_IS_NOT_ELIGIBLE_COLLATERAL_IN_BASKET);
}
```

mutable borrow for reading

File name: pkg/concordia/sources/portfolio/portfolio_positions.move
Line number: 322

```
let p = *lit::borrow_mut_with_default(
    t,
    asset_id,
    0
);

lit::set(
    t,
    asset_id,
```

```
    p + amount
);
```

mutable borrow, deref and set

File name: pkg/concordia/sources/portfolio/portfolio_positions.move
Line number: 292

```
let p = *lit::borrow_mut(
    t,
    asset_id
);

assert!(
    p >= amount,
    ERR_DECREASING_POSITION_BY_TOO_MUCH
);

lit::set(
    t,
    asset_id,
    p - amount
);
```

mutable borrow, deref and set

File name: pkg/concordia/sources/portfolio/portfolio.move
Line number: 412

```
let cur_amount = *lit::borrow(a_table, frontend_id);

assert!(amount <= cur_amount, ERR_DECREASE_LIAB_BY_TOO_MUCH);

lit::set(
    a_table,
    frontend_id,
    cur_amount - amount
);
```

mutable borrow, de-ref and set

File name: pkg/concordia/sources/price/price_decoder.move
Line number: 131, 161

```
PriceUpdate {
    asset_id: asset_id,
    publish_time_ms: publish_time_ms,
    price: price,
    confidence: confidence
}
```

There is no need to be explicit if names are identical, so `asset_id: asset_id` can be written as just `asset_id` and so on.

Most admin init functions, (signer == @concordia), "hard code" the address `@concordia` however, some do not:

```
public(friend) fun init(account: &signer)
    assert!(!exists<Collaterals>(signer::address_of(account)),
ERR_COLLATERAL_MANIFEST_ALREADY_EXISTS);
```

Same in `price_store::init`, `lending_broker::init_broker_registry`. It would be good to follow one style for clarity.

Init functions called from `public entry fun setup(concordia: &signer)` are inconsistently checking for existing data - including `init_frontend_container`. Structs that do not have the `Drop` ability could not be overwritten without a de-structure.

In `public(friend) fun register_asset<Asset>` naming of vars is inconsistent: `let g = borrow_global_mut<AssetRegistry>(@concordia);` usually this is called `ar`. Other such instances exist. Same for `let ps = borrow_global<ProfileRegistry>(@concordia);` which is otherwise `pr`.

Generics are inconsistently named. For example the Aptos `CoinType` is sometimes a `TCoin` and otheres a `C`. In general it would be good to use a clear naming convention.

File name: pkg/concordia/sources/core.move
Line number: 40

```
let portfolio_id = portfolio_registry::new_portfolio(user, basket_id);
```

The variable portfolio_id actually holds an address. For the sake of maintainability it would be good to appropriately name variables.

File name: pkg/concordia/sources/core.move
Line number: 256, 269

```
public(friend) fun accrue_interest<TBroker>() acquires LendingBroker {
    let amt = interest_accrued_amount<TBroker>(
    );

    let b = borrow_global_mut<LendingBroker<TBroker>>(@concordia);

    if (b.ts_interest_accrued >= now_seconds()) {
        return
    };

    // TODO - split out fee

    // increase borrowed coins
    b.borrowed_coins = math::add(
        b.borrowed_coins,
        amt
```

```
    );

    // sync interest accrued ts
    b.ts_interest_accrued = now_seconds();
}
```

The function `now_seconds()` which acquires state is called twice. This could be optimized by using a variable.

File name: pkg/concordia/sources/toolbox/lite_iterable_table.move
Line number: 35

```
public fun get_mut<K: drop + copy, V>(lit: &mut LIT<K, V>, k: K): &mut V {
    let idx = get_index<K, V>(k, lit);
    vector::borrow_mut<V>(&mut lit.items, idx)
}

public fun borrow_mut<K: drop + copy, V>(lit: &mut LIT<K, V>, k: K): &mut V {
    get_mut<K, V>(lit, k)
}
```

This function wraps another and they are therefore identical.

File name: pkg/concordia/sources/entry/accessor/portfolio_accesor.move

This file contains various functions that are unused. Some of them call themselves, as they are not prefixed with `p::` i.e.:

```
public fun has_liability(
    p_addr: address,
    liab_id: u64
): bool {
    has_liability(p_addr, liab_id)
}
```

There are various unused functions such as check_loan_notes_equal_deposit_notes, assert_loan_notes_equal_deposit_notes, assert_loan_notes_equal_deposit_notes, lend_collateral, change_authority, broker_matches_asset, vault_address, update_collateral.

File name: pkg/concordia/sources/portfolio/portfolio_registry.move
Line number: 64

```
fun make_portfolio(
    account: &signer,
    basket_id: u64
): address acquires PortfolioRegistry {
    let p = portfolio::new(
        account,
        basket_id,
        peek_sequence(),
    );
    ...
}
```

This function uses `peek_sequence` to determine the next portfolio id. The code flow later arrives at `insert_portfolio` where `new_key` is called which again gets the vector length and adds 1 to arrive at the id determined before. This is a maintainability concern. The id should be passed along and not generated twice.

```
const FACTOR: u128 = 1000000000000000000; // 1e18
```
Large numbers can be written with underscores to improve readability i.e.: 1_000_000_000_000_000_000

Severity and Impact Summary

No security impact.

Recommendation

Cleaning up the code could improve readability and maintainability.

DOUBLE CHECK OF CONDITION, IGNORES 0 AMOUNT RETURNED BY FIRST CHECK

Finding ID: FYEO-CONCORDIA-ID-15
Severity: **Informational**
Status: **Remediated**

Description

This function double checks that `b.ts_interest_accrued >= now_seconds()`. Since `amt` is set to 0 maybe test against that the 2nd time.

Proof of Issue

File name: pkg/concordia/sources/lending/broker/lending_broker.move
Line number: 252

```
let amt = interest_accrued_amount<TBroker>(
);

...

if (b.ts_interest_accrued >= now_seconds()) {
    return
};
```

The `amt` is calculated here:

```
public(friend) fun time_since_interest_accrual<TBroker>(): u64 acquires LendingBroker
{
    let now = now_seconds();
    let last_accrued = borrow_global<LendingBroker<TBroker>>(
        @concordia
    ).ts_interest_accrued;

    if (now <= last_accrued) {
        return 0
    };

    return now - last_accrued
}
```

If brokers ts_interest_accrued were to be greater than now in seconds this would return 0.

Severity and Impact Summary

One condition is checked twice yet if the variable `amt` is 0 it might be appropriate return early by checking for that.

Recommendation

The code can be cleaned up. Also consider returning an error on 0.

LENDING PARAMETERS FOR BROKER CAN ONLY BE SET DURING INITIALIZATION

Finding ID: FYEO-CONCORDIA-ID-16
Severity: **Informational**
Status: **Remediated**

Description

The only function setting lending parameters on broker is the initialization, therefore they can not be updated.

Proof of Issue

File name: pkg/concordia/sources/entry/entry_admin.move
Line number: 177

```
lending_interest_v1::upsert<TBroker>(
    concordia,
    from_bps(u1),
    from_bps(u2),
    from_bps(r0),
    from_bps(r1),
    from_bps(r2),
    from_bps(r3)
);
```

Severity and Impact Summary

Consider if these parameters ever need adjustment. Add an update function if appropriate.

Recommendation

Add an update function if appropriate.

NEAR IDENTICAL INIT FUNCTIONS

Finding ID: FYEO-CONCORDIA-ID-17
Severity: **Informational**
Status: **Remediated**

Description

There are two near identical init front-end functions. One returns the front-end id.

Proof of Issue

File name: pkg/concordia/sources/entry/entry.move
Line number: 38

```
public entry fun init_frontend<TBroker>(user: &signer) {
    fe::init_frontend<TBroker>(user);
}

public fun init_frontend_via_contract<TBroker>(user: &signer): u64 {
    fe::init_frontend<TBroker>(user)
}
```

Severity and Impact Summary

Why not always return the front-end id.

Recommendation

Consider if one function is sufficient.

SOME FUNCTIONS DO NOT CALL PRICE_UPDATER

Finding ID: FYEO-CONCORDIA-ID-18
Severity: **Informational**
Status: **Remediated**

Description

The functions `repay_with_coins` and `repay_with_deposit_notes` do not call `price_updater::update_prices(oracle)`. The function `repay_with_collateralized_deposit_notes` and `repay_with_collateralized_coins` however do.

Proof of Issue

File name: pkg/concordia/sources/entry/entry.move
Line number: 168

```
public entry fun repay_with_coins<TBroker, TCoin>(
    user: &signer,
    portfolio: address,
    frontend_id: u64,
    amount: u64,
) {
    // assert oracle allows
    entry_acl::assert_user_owns_portfolio(
        signer::address_of(user),
        portfolio
    );

    let coins = coin::withdraw<TCoin>(user, amount);

    core::repay_with_coins<TBroker, TCoin>(portfolio, frontend_id, coins);
}
```

Severity and Impact Summary

It seems that all these functions were probably meant to udpate prices.

Recommendation

Consider adding the price update feed to these functions.

SUPERFLUOUS FUNCTION CHECK_REGISTER

Finding ID: FYEO-CONCORDIA-ID-19
Severity: **Informational**
Status: **Remediated**

Description

The function `check_register` is superfluous. The `register` function in Aptos is short-circuiting and returns early if nothing needs to be done. Calls to this function can simply be replaced with a call to `coin::register`.

Proof of Issue

File name: pkg/concordia/sources/toolbox/vault.move
Line number: 63

```
public fun check_register<T>(v: &Vault) {
    let vault_addr = get_address(v);
    if (!coin::is_account_registered<T>(vault_addr)) {
        spec {
            assume coin::balance<T>(vault_addr) == 0;
        };
        let vault_signer = ra::get_signer(&v.ra);
        coin::register<T>(&vault_signer);
    };
}
```

As can be seen, the `register` function in Aptos coin has a short circuit:

```
public fun register<CoinType>(account: &signer) {
    let account_addr = signer::address_of(account);
    // Short-circuit and do nothing if account is already registered for CoinType.
    if (is_account_registered<CoinType>(account_addr)) {
        return
    };
    ...
```

Severity and Impact Summary

This function increases the complexity of the code base.

Recommendation

Replace this function with calls to `coin::register`.

SUPERPOSITION DOES NOT CHECK IF TBROKER IS PART OF ITS FRONTENDS

Finding ID: FYEO-CONCORDIA-ID-20
Severity: **Informational**
Status: **Remediated**

Description

Both `public(friend) fun lend_int<TBroker, TCoin>` and `public(friend) fun redeem_int<TBroker, TCoin>` take a TBroker but do not check if said TBroker is part of the contracts front-ends.

Proof of Issue

File name: pkg/superv2/sources/super_core.move
Line number: 39

```
public(friend) fun lend_int<TBroker, TCoin>(
    user: &signer,
    amount: u64
) {
    entry::lend<TBroker, TCoin>(user, amount);
}
```

Severity and Impact Summary

While these wrap public functions on the Concordia contract, other parts of the superposition contract check that TBroker is registered in this contract.

Recommendation

Consider if that is a desirable check to add.

THE FUNCTION DECREASE_AVAILABLE_COINS ASSUMES THERE ARE ALWAYS ENOUGH TCOIN AVAILABLE

Finding ID: FYEO-CONCORDIA-ID-21
Severity: **Informational**
Status: **Remediated**

Description

The function `fun borrow<TBroker, TCoin>` calls `decrease_available_coins<TBroker>(amount)` which does not check if `available_coins >= amount`. There should probably be a custom error if a person attempts to borrow more than available.

Proof of Issue

File name: pkg/concordia/sources/lending/broker/lending_broker.move
Line number: 512

```
fun decrease_available_coins<TBroker>(amount: u64) acquires LendingBroker {
    let b = borrow_global_mut<LendingBroker<TBroker>>(@concordia);
    b.available_coins = b.available_coins - amount;
}
```

Severity and Impact Summary

The error caused will be unintelligible to users.

Recommendation

Add a custom error if the requested amount exceeds the available.

THE LIQUIDATION_PARAMETERS SHOULD CHECK AGAINST BOUNDS

Finding ID: FYEO-CONCORDIA-ID-22
Severity: **Informational**
Status: **Remediated**

Description

The `init_liquidation_parameters` and `update_liquidation_parameters` functions should have some sort of maximum check to avoid mistakes.

Proof of Issue

File name: pkg/concordia/sources/liquidations/liquidation_parameters.move
Line number: 16

```
public(friend) fun init_liquidation_parameters(account: &signer, liq_fee_rate_bps:
u64) {
    let fee_vault = vault::new(account, b"concordia-liquidation-vault");
    move_to(account, LiquidationParameters {
        liq_fee_rate: from_bps(liq_fee_rate_bps),
        fee_vault
    })
}

public(friend) fun update_liquidation_parameters(liq_fee_rate_bps: u64) acquires
LiquidationParameters {
    let parameters = borrow_global_mut<LiquidationParameters>(@concordia);
    parameters.liq_fee_rate = from_bps(liq_fee_rate_bps);
}
```

Severity and Impact Summary

Checking for a maximum allowed value would prevent potential mistakes when updating this data.

Recommendation

Add a check to ensure the value is within bounds.

THE TRANSFER_REVENUE FUNCTION SHOULD RETURN EARLY FOR 0 NOTE AMOUNT

Finding ID: FYEO-CONCORDIA-ID-23
Severity: **Informational**
Status: **Remediated**

Description

In the event that the calculated notes amount is 0, the function can return early.

Proof of Issue

File name: pkg/concordia/sources/lending/frontend/lending_revenue_collector.move
Line number: 38

```
let notes = calc_share_revenue(
    global_index,
    current_index,
    share
);


transfer_deposit_notes<TBroker>(notes, target);
```

Severity and Impact Summary

There is no need to transfer 0 of anything.

Recommendation

Check for possible 0 amount.

USING A RAW VALUE TO IDENTIFY A CHAIN INSTEAD OF A CONST

Finding ID: FYEO-CONCORDIA-ID-24
Severity: **Informational**
Status: **Remediated**

Description

Chain id is assigned a raw value of 1.

Proof of Issue

File name: pkg/concordia/sources/portfolio/portfolio_registry.move
Line number: 101

```
PortfolioKey {
    chain_id: 1,
    local_id: len + 1
}
```

Use already declared constants:

```
module concordia::chains {
    const APTOS: u64 = 1;
    const SOLANA: u64 = 2;
    const ARBITRUM: u64 = 3;
    const ALGORAND: u64 = 4;
    const ETHEREUM: u64 = 5;
    const CELO: u64 = 6;

    public fun aptos(): u64 {
        APTOS
    }
...
```

Severity and Impact Summary

Using a raw value here might cause issues with maintainability.

Recommendation

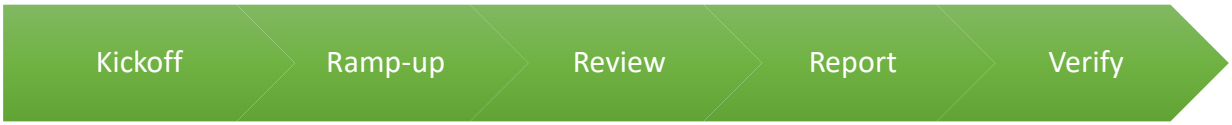
Use const values already declared.

OUR PROCESS

METHODOLOGY

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow



KICKOFF

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

RAMP-UP

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

REVIEW

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

VERIFY

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

ADDITIONAL NOTE

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials

- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations