Security Audit Report

Fee Abstraction Module by Notional

Authors: Aleksandar Ljahovic, Ivan Golubovic

Last revised 30 October, 2023

Table of Contents

Audit overview

The Project

In collaboration with Osmosis team, Notional developed Fee Abstraction Module as a tool available to other chains to facilitate transaction fee payments in IBC tokens. This module can be integrated to a chain, and this chain then communicated with Osmosis chain to be able to obtain all the required data to make IBC tokens eligible for fee payments.

The core mechanism of the module derives the exchange rate (Time-Weighted Average Price or TWAP) from Osmosis, facilitating the conversion of IBC tokens to native tokens for fee calculation. To kickstart this operation, the module is endowed with a foundational token fund. Moreover, it leverages Osmosis' smart contracts to exchange its accumulated IBC tokens for native tokens, replenishing the token pool needed for fee transactions.

A crucial aspect of this project was the customization of `AnteHandle` functions, which play an important role in the transaction validation process. Additionally, IBC transfers are identified as valuable components for the seamless operation of this module. Through this audit, we inspected the Fee Abstraction Module, aiming to ensure its robustness, accuracy, and security in handling transaction fees, while enabling interaction with the Osmosis chain.

Scope of this audit

The audit was scheduled from October 2, 2023, to October 27, 2023. The audit team consisted of the following personnel:

- Aleksandar Ljahovic
- Ivan Golubovic

Conducted work

The audit project encompassed the following activities:

- Manual code inspection of the Fee Abstraction Module. We conducted a thorough examination of the codebase, documenting our insights in the "System Overview" section. The manual code inspection revealed the majority of the findings presented in this report.
- Thread modeling and inspection is documented in “Threat Inspection” chapter, giving the detailed insight in conducted work, identified threats and code snippets of interest etc.
- Where possible, we attempted to reproduce the findings using end-to-end test suite, as well as try to use the environment to test edge cases.

Conclusions

Overall, we found the codebase to exhibit quality, characterized by well-structured and comprehensible code. The comprehensive test suite includes unit, and valuable end-to-end testing environment. During the audit, we identified 1 critical, 4 high and 2 medium severity findings, while others were of low and informational severity. Our collaboration with the Osmosis/Notional team was exemplary, significantly enhancing the quality of work executed within the audit's timeframe.

Audit dashboard

Target Summary

- **Type**: Specification and Implementation
- **Platform**: Go
- **Artifacts**:
    - Commit hash: 3ef7b78d034d93632fa363519da9b77ff59c9805

Engagement Summary

- **Dates**: 02.10.2023. to 27.10.2023
- **Method**: Manual code review, protocol analysis, design analysis, testing
- **Employees Engaged**: 2

Severity Summary

Finding Severity	**#**
Critical	1
High	4
Medium	2
Low	2
Informational	4
Total	**13**

System overview

Introduction

In this audit, we are evaluating the implementation and security of the Fee Abstraction Module developed by the Notional team for the Cosmos ecosystem, specifically in conjunction with the Osmosis decentralized exchange. This module is designed to offer unique functionalities that enhance the flexibility of transaction fee payments through the innovative use of Inter-Blockchain Communication (IBC) tokens.

Fee Abstraction Module empowers customer chains to execute fee payments using any IBC token available on Osmosis. The ibc tokens are exchanged for the native tokens held in module’s account. This is achieved by integrating oracle Time-Weighted Average Price (TWAP) data from Osmosis, ensuring real-time and accurate conversion rates. Users on customer chains that integrate this module can pay transaction fees in IBC tokens, with the conversion rates determined by Osmosis’ TWAP oracles.

The second feature of the module, on the other hand, utilizes packet forward middleware and Crosschain Swaps smart contract to enable an automated process where accumulated fees in IBC tokens are swapped for the native tokens using Osmosis. This not only simplifies the fee collection process for the customer chains but also potentially boosts the trading volume on Osmosis, given the automated token swap mechanism.

This audit will provide a comprehensive analysis of the Fee Abstraction Module’s security, efficiency, and operational integrity. We will evaluate the technical implementation, the reliability of the TWAP data integration, the effectiveness of the cross-chain token swapping, and the overall impact on the transaction fee payment ecosystem within the Cosmos network. Each aspect will be scrutinized to ensure that the module not only enhances transaction fee payment flexibility but also adheres to the highest standards of security and efficiency.

Key design features

Decorators

One of the key design approaches here is the customization of `MempoolFeeDecorator`. More precisely, decorators are used to modify the AnteHandler, a crucial component in the Cosmos SDK that handles the processing of transaction fees. The decorators, specifically `FeeAbstractionDeductFeeDecorate` and `FeeAbstrationMempoolFeeDecorator`, are customized to facilitate fee payment using IBC tokens.

FeeAbstractionDeductFeeDecorate

This decorator is responsible for handling the deduction of transaction fees. It is initialized with keepers for account, bank, fee abstraction, and fee grant. The `AnteHandle` function is the core, checking if the transaction is a `FeeTx` and then proceeding to handle the fee deduction based on the transaction's characteristics.

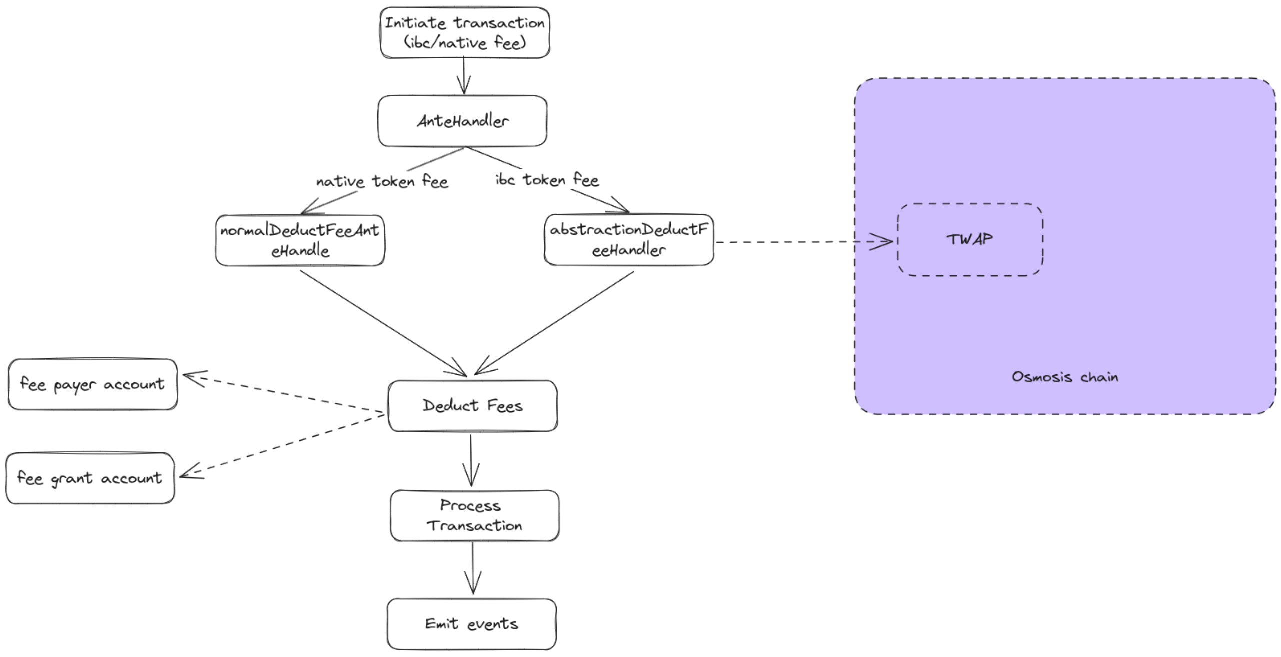
- If the transaction does not involve IBC tokens or the host chain configuration is not set for the fee's denomination, it calls the `normalDeductFeeAnteHandle` method. This method deducts the fee from the fee payer or fee granter's account and ensures that the fee collector module account is set.
- If the transaction involves IBC tokens and the host chain configuration is set, it calls the `abstractionDeductFeeHandler` method. This method calculates the equivalent native tokens from IBC tokens and deducts the fees accordingly.

FeeAbstrationMempoolFeeDecorator

This decorator checks if the transaction's fee meets the local validator's minimum gas fee during the transaction's inclusion in the mempool. It ensures that the fee is sufficient, and if not, the transaction is rejected from the mempool.

- It calculates the required fee based on the transaction's gas limit and the minimum gas prices. If the fee is insufficient, an error is returned, and the transaction is not processed.
- Special conditions like bypassing the minimum fee requirement or handling global fees are also considered. Transactions that meet these conditions are processed without the minimum fee check.

These two customizations are illutrated in the following diagram:



Epochs and IBC

In the context of the fee abstraction module, Inter-Blockchain Communication (IBC) and epochs are combined to facilitate automated interactions with the Osmosis chain for querying TWAP and executing cross-chain swaps. This is achieved using epoch mechanism, which triggers specific IBC operations at the end of each epoch.

The `AfterEpochEnd` function serves as the routing point where epochs and IBC intersect. At the end of each epoch, identified by the epoch identifier, this function is invoked to initiate either a TWAP query or a cross-chain swap on the Osmosis chain.

TWAP Query

The `executeAllHostChainTWAPQuery` function is called, which internally invokes the `handleOsmosisIbcQuery` function to manage the IBC query to Osmosis. This ensures that the TWAP is queried from the Osmosis chain at the end of each specific epoch, facilitating real-time and automated retrieval of TWAP data.

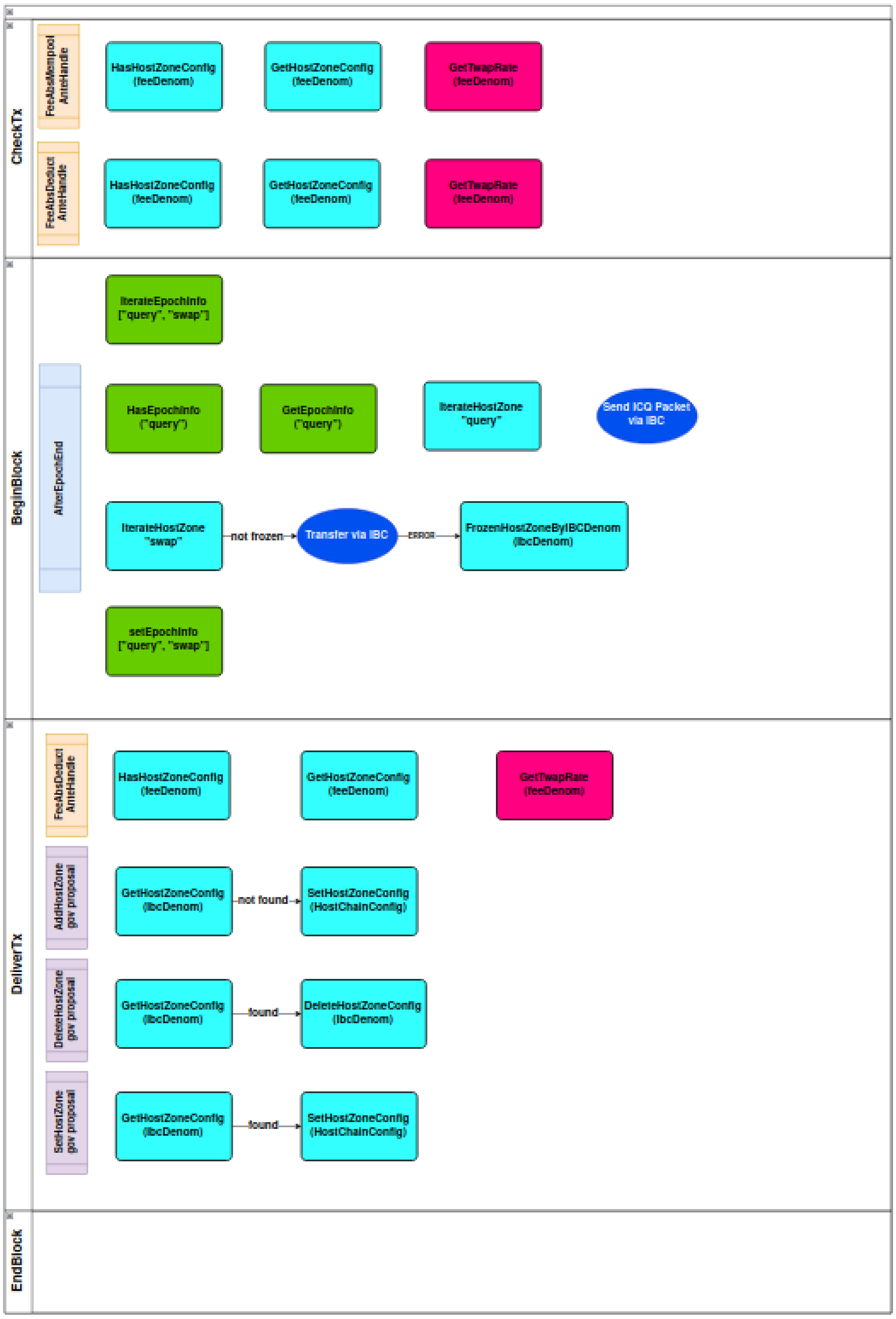
Cross-Chain Swap

To be able to support these ibc token paid transactions, the fee abstraction module has a designated account to store native tokens. Initial fund is needed after the launch, but the swapping using Osmosis liquidity pools is the mechanism to keep this account in operation.

For the swap, the `executeAllHostChainSwap` function is invoked, iterating through all host zones and executing the swap over the chain. This automated process ensures that cross-chain swaps are performed at the end of each designated epoch, enabling automated liquidity management and token swaps between chains. Osmosis' specific Crosschain Swaps and Swaprouter smart contract are utilized to perform this operation.

Transactions through ABCI functions.

The diagram below shows the processes that are called in the ABCI functions (CheckTx, BeginBlock, DeliverTx) in the fee abstraction module. It shows the calls to KVStores and the initiation of sending packages via IBC:


CheckTx
FeeAbsMempool AnteHandle
HasHostZoneConfig (feeDenom)
GetHostZoneConfig (feeDenom)
GetTwapRate (feeDenom)
FeeAbsDeduct AnteHandle
HasHostZoneConfig (feeDenom)
GetHostZoneConfig (feeDenom)
GetTwapRate (feeDenom)
BeginBlock
AfterEpochEnd
IterateEpochInfo ["query", "swap"]
HasEpochInfo ("query")
GetEpochInfo ("query")
IterateHostZone "query"
Send ICQ Packet via IBC
IterateHostZone "swap"
not frozen
Transfer via IBC
ERROR
FrozenHostZoneByIBCDenom (ibcDenom)
setEpochInfo ["query", "swap"]
DeliverTx
FeeAbsDeduct AnteHandle
HasHostZoneConfig (feeDenom)
GetHostZoneConfig (feeDenom)
GetTwapRate (feeDenom)
AddHostZone gov proposal
GetHostZoneConfig (ibcDenom)
not found
SetHostZoneConfig (HostChainConfig)
DeleteHostZone gov proposal
GetHostZoneConfig (ibcDenom)
found
DeleteHostZoneConfig (ibcDenom)
SetHostZone gov proposal
GetHostZoneConfig (ibcDenom)
found
SetHostZoneConfig (HostChainConfig)
EndBlock

Conclusion

This system ensures that users on the Cosmos network can seamlessly pay transaction fees using IBC tokens. By leveraging the Osmosis chain for real-time exchange rates and token swaps, the system is both user-friendly and adaptable to future changes and improvements. The use of decorators and the IBCModule ensures that the process is secure, efficient, and seamless for all users.

Threat inspection

Threat Model Entry: Network Congestion and Transaction Failures in Fee Abstraction Module

System Component: IBC Communication and Transaction Processing

Description:

The Fee Abstraction Module in the Cosmos ecosystem is heavily reliant on Inter-Blockchain Communication (IBC) for various operations, including querying TWAP prices from Osmosis, performing token swaps, and unwinding IBC denoms. Network congestion or relayer failures could lead to delayed or failed transactions, impacting the module's ability to calculate fees or perform swaps efficiently.

Potential Threat:

Network congestion or a halt in the relayer's operation could lead to a backlog of transactions, resulting in delayed or failed transactions. This could impact users attempting to pay fees with IBC tokens, leading to transaction failures or significant delays, affecting the overall user experience and system performance.

Attack Vector:

- **Network Congestion:** An unusually high volume of transactions could overload the network, leading to delayed processing of transactions related to fee payments in IBC tokens.
- **Relayer Halt:** If the relayer stops functioning, IBC transactions could be stuck, leading to failed or delayed transactions.
- **Failed Transactions:** In scenarios of network congestion or relayer halts, transactions could fail, and it needs to be inspected if recovery systems are in place to handle such scenarios.

Code to Focus on for this Threat:

1. **Sending Interchain Query:**
   `_, err := k.SendInterchainQuery(ctx, icqReqs, sourcePort, sourceChannel)`
   - **Observation:** The module sends interchain queries to Osmosis. The handling of errors and the efficiency of this process during network congestion should be inspected.
2. **Handling Acknowledgement Packet:**
   `func (k Keeper) OnAcknowledgementPacket(ctx sdk.Context, ack channeltypes.Acknowledgement, icqReqs []abci.RequestQuery) error`
   - **Observation:** The handling of acknowledgements and errors is crucial. The system's response to errors and its ability to recover or retry transactions should be evaluated.
3. **Timeout Handling:**
   `func (k Keeper) OnTimeoutPacket(ctx sdk.Context) error`
   - **Observation:** The system's response to timeouts is essential, especially during network congestion. The mechanism to retry or handle timed-out transactions should be robust.
4. **IBC Token Transfer:**
   `func (k Keeper) transferOsmosisCrosschainSwap(ctx sdk.Context, hostChainConfig types.HostChainFeeAbsConfig) error`
   - **Observation:** The transfer of IBC tokens, especially during the swap process, should be efficient and error-resistant to ensure that network congestion doesn't lead to failed swaps.

Important Requirements:

- The system should be equipped to handle a high volume of transactions efficiently to mitigate the impact of network congestion.
- The error handling and recovery mechanisms should be robust to ensure that the system can recover from failed transactions or timeouts effectively.
- The use of Interchain Queries from Strangelove should be evaluated to understand its efficiency and reliability during network congestion.

Example of Impact:

In a scenario where the network is congested, users attempting to pay transaction fees with IBC tokens could experience significant delays or transaction failures. This could lead to a poor user experience and could potentially impact the integrity of the transaction processing system if not handled efficiently. The system's ability to recover from such scenarios, handle transaction timeouts, and process a backlog of transactions efficiently is crucial to mitigate the impact of this threat.

Threat Inspection Results:

The Fee Abstraction Module's dependency on Inter-Blockchain Communication (IBC) for various operations, particularly interchain queries and cross-chain swaps on the Osmosis chain, was closely examined. The inspection primarily focused on the code contained in `ibc.go`, with special attention given to the `OnAcknowledgementPacket` and `OnTimeoutPacket` functions, alongside the helper functions invoked from them, namely `handleOsmosisIbcQuery` and `transferOsmosisCrosschainSwap`.

Cross-Chain Swaps Execution:

The code leverages built-in IBC features and Osmosis' smart contract designs (Crosschain Swaps and Swap Router) for handling errors and timeouts during cross-chain swaps. Specifically, the `executeTransferMsg` function relies on the built-in `transferKeeper` to facilitate the token transfer over IBC.

```
func (k Keeper) executeTransferMsg(ctx sdk.Context, transferMsg
*transfertypes.MsgTransfer) (*transfertypes.MsgTransferResponse, error) {
    if err := transferMsg.ValidateBasic(); err != nil {
        return nil, fmt.Errorf("bad msg %v", err.Error())
    }
    return k.transferKeeper.Transfer(sdk.WrapSDKContext(ctx), transferMsg)
}
```

This mechanism appears sufficiently robust to mitigate the potential threats of network congestion, relayer halts, and transaction failures as outlined in the threat model.

Interchain Queries Execution:

The handling of interchain queries exhibits a higher level of customization, particularly within the `OnAcknowledgementPacket` and `OnTimeoutPacket` functions. The code contained is considered less successful in handling network congestion or delayed transactions.

```
func (k Keeper) OnAcknowledgementPacket(ctx sdk.Context, ack
channeltypes.Acknowledgement, icqReqs []abci.RequestQuery) error {
    // ...
    case *channeltypes.Acknowledgement_Error:
        k.IterateHostZone(ctx, func(hostZoneConfig types.HostChainFeeAbsConfig) (stop
bool) {
            err := k.FrozenHostZoneByIBCDenom(ctx, hostZoneConfig.IbcDenom)
            // ...
        })
    // ...
}
```

In the event of an error acknowledgment, the current implementation opts to freeze all host zones, which could lead to an entire feature shutdown due to an error in a single host zone. This approach may not be the most optimal for ensuring system resilience in the face of network congestion or relayer reliability issues concerning timely transfer deliveries.

Dependency on External Systems:

The dependency of the external system (TWAP from Osmosis) has to be better handled in case of timed out transfers as described in one of the findings. The value obtained from Osmosis (exchange rate) is used in token amount calculations which makes it particularly important to keep it up to date. Outdated values could lead to serious consequences for the user or the system itself.

The findings that came out of this threat inspection are:

1. Outdated Exchange Rate Risk Due to Potential IBC Relayer Delays
2. Unnecessary Host Zone Freezing Due to Lack of Granular Error Handling

Threat Model Entry: Fee Deduction and Transaction Validation in Customized AnteHandler

System Component: Customized AnteHandler for Transaction Pre-processing

Description:

The customized AnteHandler in the Fee Abstraction Module is a critical component within the Cosmos ecosystem. It is responsible for the pre-processing of transactions before they enter the mempool, focusing on transactions paid with IBC tokens. The AnteHandler inspects transactions to ensure they meet the minimum fee requirements and calculates the appropriate fees using the Osmosis price oracle.

Potential Threat:

There could be vulnerabilities in the transaction inspection process, allowing invalid transactions to pass through or enabling front-running attacks. Errors in fee deduction, especially in the conversion and calculation of fees paid with IBC tokens, could lead to incorrect fee deductions, impacting the network's financial integrity.

Attack Vector:

1. **Transaction Inspection Weaknesses:** Flaws in the transaction inspection process that might allow invalid transactions to pass or be susceptible to front-running attacks.
2. **Fee Deduction Errors:** Mistakes in the conversion and calculation of fees, especially when paid with IBC tokens, leading to incorrect fee deductions.
3. **Decorator Customization Flaws:** Vulnerabilities arising from the integration of customized decorators with the base AnteHandler, leading to potential inconsistencies or security lapses.

Code to Focus on for this Threat:

Transaction Inspection and Fee Deduction:

```
func (fadfd FeeAbstractionDeductFeeDecorate) AnteHandle(ctx sdk.Context, tx sdk.Tx,
simulate bool, next sdk.AnteHandler) (newCtx sdk.Context, err error) {
    // ... (rest of the code)
    fee := feeTx.GetFee()
    if len(fee) == 0 {
        return fadfd.normalDeductFeeAnteHandle(ctx, tx, simulate, next, feeTx)
    }
    // ... (rest of the code)
    err = fadfd.bankKeeper.SendCoinsFromAccountToModule(ctx, feeAbstractionPayer,
feeabstypes.ModuleName, ibcFees)
    if err != nil {
        return ctx, err
    }
    err = DeductFees(fadfd.bankKeeper, ctx, deductFeesFrom, nativeFees)
    if err != nil {
        return ctx, err
    }
    // ... (rest of the code)
}
```

The code should be thoroughly checked to ensure that the transaction inspection is robust and that the fee deduction, especially for IBC tokens, is accurate. The integration of the customized decorators with the base AnteHandler should also be examined for potential vulnerabilities.

Important Requirements:

- The system must ensure a robust transaction inspection process to effectively identify and reject invalid transactions and prevent front-running attacks.
- The fee deduction mechanism, especially for IBC tokens, must be accurate and error-resistant to maintain the network's financial integrity.
- The integration of customized decorators with the base AnteHandler must be seamless and secure to prevent potential vulnerabilities and ensure consistent transaction processing.

Example of Impact:

If not mitigated, these vulnerabilities could lead to the processing of invalid transactions, incorrect fee deductions, and potential front-running attacks. These issues could result in financial losses for users and undermine the integrity and trust in the Cosmos ecosystem. The network's reputation could be damaged, and the overall security and stability of the system could be compromised. Ensuring the robustness of the AnteHandler and the accuracy of fee deductions is crucial to mitigating these risks.

Threat Inspection Results:

Customized decorators for transaction processing were one of the key focuses in our code review since they present the core structure enabling the transaction fee payments in IBC tokens. The code we focused on to inspect the possibility of exploitation of this threat is mainly located in x/feeabs/ante/decorate.go while some helper functions are placed throughout the Fee Abstraction module.

The fee deduction decorator customization was tracked via `unc (fadfd FeeAbstractionDeductFeeDecorate) AnteHandle` from where the two main flows end up in `func`

`(fadfd FeeAbstractionDeductFeeDecorate) normalDeductFeeAnteHandle` and `func (fadfd FeeAbstractionDeductFeeDecorate) abstractionDeductFeeHandler` which marks the two paths in fee deduction process:
1. Normal Deduction - handling native token fees
2. Abstract Deduction - managing IBC token fees

Another key track to follow the decorator structure was `func (famfd FeeAbstrationMempoolFeeDecorator) AnteHandle`. This `AnteHandle` follows the transactions entering mempool, only in CheckTx phase. The validation includes checking the bypass messages and minimum gas prices, as well as calculation of native

Transaction validation process inspection did not show obvious flaws in the initial phase, where it was concluded that the type of validation, coin properties (denom, amount), authorization and fee payments were exhaustively checked. However, later in the audit a few weaknesses were identified.

First, module account loading using `fadfd.accountKeeper.GetModuleAddress` is performed in `normalDeductFeeAnteHandle` while it is missing in `abstractionDeductFeeHandler` which poses an authorization security risk as it is further elaborated in the following finding:
Missing Fee Collector Account Check in abstractionDeductFeeHandler

The process of freezing host zones as a security measure is introduced to tackle any errors in the zones processing and it serves as a pointer to the zone which is not safe to use. While the two IBC transfers include checks for frozen zone status, the decorators are missing these. This could potentially lead to operations influenced by the zone with a serious flaw in financial or technical procedure and thus be harmful to the user or the chain as it is detailed in this finding:
Missing Frozen Host Zone Checks in AnteHandles

The fee calculation process contains a few simple calculations, but with some rounding processes which were our main focus in this inspection. Rounding include usage of `RoundInt()` and `Ceil()` functions. We were not able to create or confirm a scenario where these rounding could cause any harm to the user or the chain. The rounding process seems carefully crafted perceiving any possible value combination to be fairly taken into account.

While inspecting the code on a more detailed level, a few possible improvements regarding the code were discovered, as depicted in the following findings:
Minor Code Improvements
Optimize Fee Calculation Process
Optimization Opportunities in KVStore Calls

There were no discovered possibilities of serious security errors regarding integration of the module with the cosmos sdk built-in features.

While this threat was analyzed, one particular flaw was discovered which poses a critical security risk. Namely, the key generation process is found to be not unique. This flaw could potentially cause critical errors, disrupting the intended operation flow and user fee calculations as elaborated in the following finding:
Critical Risk of Non-Uniqueness in KVStore Keys

Threat Model Entry: Oracle (TWAP) Manipulation in Fee Abstraction Module

System Component: Osmosis TWAP Query and Fee Calculation Mechanism

Description:

The Fee Abstraction Module in the Cosmos ecosystem allows users to pay transaction fees with IBC tokens, utilizing Osmosis for real-time TWAP data. The module queries the Osmosis chain at the end of every epoch to obtain the exchange rate, which is then used to calculate the transaction fee in terms of IBC tokens.

The process involves querying the Osmosis chain for the TWAP data, receiving this data, and then applying it to calculate the equivalent transaction fee in IBC tokens. This is facilitated by the `handleOsmosisIbcQuery` function, which sets the start time for the query and iterates through the host zone to send multiple TWAP query requests.

Potential Threat:

An attacker could manipulate the TWAP data on the Osmosis chain or interfere with the query process to influence the exchange rate obtained by the Fee Abstraction Module. This could lead to an inaccurate calculation of transaction fees, causing users to either overpay or underpay.

Example of Attack from Osmosis Side:

An attacker aiming to manipulate the TWAP data could potentially gain control over or influence the Osmosis oracle that provides the TWAP data. By either compromising the oracle's security or exploiting potential vulnerabilities in its design, the attacker could feed manipulated data to the Fee Abstraction Module.

For instance, if the attacker manages to gain control over a significant portion of the oracle nodes or manipulates the data they provide, they could artificially inflate or deflate the TWAP. This manipulated data would then be queried by the Fee Abstraction Module, leading to inaccurately calculated fees.

Attack Vector:

- **Data Manipulation:** The attacker could manipulate the TWAP data on the Osmosis chain, leading to inaccurate exchange rates being returned to the Fee Abstraction Module.
- **Query Interference:** The attacker could interfere with the IBC query process, either by blocking the queries or injecting false data into the responses.
- **Time Manipulation:** Given that the start time for the TWAP query is set based on the block time, an attacker could potentially manipulate this to influence the data obtained.

Code to Focus on for this Threat:

1. **Setting Start Time for TWAP Query:** `startTime := ctx.BlockTime().Add(-queryTwapEpochInfo.Duration)`
    - **Observation:** The start time for the TWAP query is determined by the current block time and the epoch duration. An attacker could potentially exploit this to manipulate the start time and influence the TWAP data obtained.
2. **Sending Osmosis Query Request:** `err := k.SendOsmosisQueryRequest(ctx, reqs, types.IBCPortID, params.IbcQueryIcqChannel)`

    - **Observation:** The query requests are sent to the Osmosis chain through IBC. An attacker could potentially target this communication channel to interfere with the query requests or responses.
3. **Handling IBC Queries:**
   ```
   _, err := k.SendInterchainQuery(ctx, icqReqs, sourcePort, sourceChannel)
   ```
    - **Observation:** The IBC queries are handled and sent through this function. Security measures need to be robust here to prevent any interference or manipulation of the query data.

Important Requirements:

1. he system must implement validation mechanisms to ensure the integrity of the TWAP data obtained from the Osmosis chain. This includes validating the authenticity and accuracy of the data to prevent any manipulations that could lead to inaccurate fee calculations.
2. Security protocols for the IBC query process need to be fortified to prevent any external interference or manipulations. This includes securing the communication channels and implementing checks to detect and mitigate any unauthorized alterations in the query or response data.
3. The mechanism that sets the start time for the TWAP query, based on the block time, should be secured and monitored. Implementing safeguards to prevent any potential manipulations of the block time or epoch duration is essential to ensure that the TWAP data obtained is based on accurate and untampered time parameters.
4. The security of the Osmosis oracle that provides the TWAP data should be enhanced to prevent any potential compromises or exploitations. This includes implementing advanced security protocols and monitoring systems to detect and mitigate any attempts to manipulate the oracle or the data it provides.

Example of impact:

In the event of a successful Oracle (TWAP) manipulation, the integrity of transaction fees within the Cosmos ecosystem could be severely compromised. Users might experience financial discrepancies, either overpaying or underpaying for transactions, leading to a loss of trust in the system's fairness and accuracy. The manipulated TWAP data could also change the real-time exchange rates, affecting transactions involving IBC tokens. Such an occurrence would not only undermine the financial integrity of the network but could also damage its reputation, leading to reduced user confidence and participation.

Threat Inspection Results:

To asses the security risk and identify the possibility to execute any of the attack vectors described in the threat above, the code that performs the obtaining of the TWAP rate from Osmosis and the usage of that data was thoroughly inspected. This was done to identify any possibility to improve the code in the module itself, even though the threat originates from the external source, Osmosis in this case.

The process of fetching TWAP rate from Osmosis is executed after every epoch using interchain queries, as the following code shows:

```
func (k Keeper) AfterEpochEnd(ctx sdk.Context, epochIdentifier string) {
    switch epochIdentifier {
    case types.DefaultQueryEpochIdentifier:
        k.Logger(ctx).Info("Epoch interchain query TWAP")
        k.executeAllHostChainTWAPQuery(ctx)
    //...(rest of code)

func (k Keeper) executeAllHostChainTWAPQuery(ctx sdk.Context) {
    err := k.handleOsmosisIbcQuery(ctx)
    if err != nil {
        k.Logger(ctx).Error(fmt.Sprintf("Error executeAllHostChainTWAPQuery %s",
err.Error()))
    }
```

```
}
```

The value received as a response for interchain query is saved to the KVStore, and then later used to calculate the amount in the critical financial calculation in the module:

```
//...(rest of the code)
twapRate, err := k.GetTwapRate(ctx, chainConfig.IbcDenom)
    if err != nil {
        return sdk.Coins{}, nil
    }

    // mul
    coin := ibcCoins[0]
    nativeFeeAmount := twapRate.MulInt(coin.Amount).RoundInt()
    nativeFee := sdk.NewCoin(k.sk.BondDenom(ctx), nativeFeeAmount)
//...(rest of the code)
```

What we first found possibly incorrect in this process is the timestamp manipulation regarding start of the epochs, as it is described in more detail in the following finding:
Inaccurate Time Calculation for ICQ TWAP Request

The more significant security risk emerges from the scenario where erroneous or manipulated data from Osmosis reaches the module. The code does not implement any validation checks on the TWAP data obtained from Osmosis before utilizing it in critical financial calculations within the module, as detailed in the finiding:
Unvalidated External Exchange Rate Data Usage

These findings underscore the necessity for robust validation mechanisms to ensure the integrity and rationality of the TWAP data fetched from external sources. Additionally, they emphasize the importance of secure timestamp handling to prevent potential time manipulation attacks that could adversely impact the TWAP data acquisition process.

Threat Model Entry: Insufficient Liquidity in Osmosis Pools Affecting Fee Abstraction Module

System Component: Osmosis Liquidity Pools and Fee Abstraction Module

Description:

The Fee Abstraction Module in the Cosmos ecosystem enables users to pay transaction fees with IBC tokens, relying on Osmosis liquidity pools to convert these tokens into native tokens. The module has its own account where it stores native tokens received from Osmosis via crosschain smart contracts. These tokens are then used for transaction fees when users opt to pay fees in IBC tokens. The efficiency and reliability of this process are contingent upon the liquidity available in Osmosis pools.

Potential Threat:

A lack of sufficient liquidity in Osmosis pools could hinder the Fee Abstraction Module's ability to convert IBC tokens into native tokens efficiently. This insufficiency could lead to increased transaction costs due to slippage or even transaction failures, adversely affecting the user experience and the overall functionality of the Fee Abstraction Module.

Example of Attack from Osmosis Side:

An attacker could exploit the dependency on Osmosis liquidity by either withdrawing a significant amount of liquidity or executing large trades that could lead to severe slippage. This would result in the Fee Abstraction Module being unable to obtain the required amount of native tokens to facilitate transactions, causing delays or failures in transaction processing.

Attack Vector:

- **Liquidity Withdrawal**: An attacker or a group of colluding entities could withdraw a significant amount of liquidity from Osmosis pools, leading to insufficient liquidity for converting IBC tokens into native tokens.
- **Large Trades**: Executing large trades that lead to severe slippage, impacting the exchange rate and the availability of native tokens for the Fee Abstraction Module.

Code to Focus on for this Threat:

The code on Osmosis' side handling cross-chain swaps should be inspected, as well as fee abstraction module to check if there are any recovery or error handling strategies in case there is not enough liquidity on Osmosis.

Important requirements:

The code should ensure that there is adequate liquidity in Osmosis pools before attempting to convert IBC tokens into native tokens. The handling of scenarios where liquidity is insufficient needs to be robust to prevent transaction failures.

Example of Impact:

Insufficient liquidity in Osmosis pools could lead to transaction failures or increased costs due to slippage for users opting to pay fees with IBC tokens. This could result in a degraded user experience, reduced efficiency of the Fee Abstraction Module, and a loss of trust in the system’s ability to facilitate transactions with IBC tokens. The dependency on Osmosis for liquidity becomes a critical point of failure, potentially undermining the financial and operational integrity of the Cosmos ecosystem.

Threat Inspection Results:

The idea laying behind this threat is to affect TWAP values from Osmosis to then affect the calculations in Fee Abstraction Module itself. Due to this, we can conclude here that previous threats already included TWAP exchange rate inspection, and same principles could be applied here. The slippage and pool liquidities in Osmosis are defended in their code, while Fee Abstraction Module is obliged to be protected only against received external values, in this case the TWAP.

Due to these reasons, for this threat inspection, we will only repeat findings regarding exchange rate:
Unvalidated External Exchange Rate Data Usage
Outdated Exchange Rate Risk Due to Potential IBC Relayer Delays

Findings

Title	Project	Severity	Impact	Exploitability	Status
Critical Risk of Non-Uniqueness in KVStore Keys	Fee Abstraction Module by Notional	4 CRITICAL	3 HIGH	3 HIGH	ACKNOWLEDGED
Outdated Exchange Rate Risk Due to Potential IBC Relayer Delays	Fee Abstraction Module by Notional	3 HIGH	3 HIGH	2 MEDIUM	ACKNOWLEDGED
Lack of Frozen Status Check in Cross-chain Swap Execution	Fee Abstraction Module by Notional	3 HIGH	3 HIGH	2 MEDIUM	RISK ACCEPTED
Missing Frozen Host Zone Checks in AnteHandles	Fee Abstraction Module by Notional	3 HIGH	3 HIGH	2 MEDIUM	ACKNOWLEDGED
Unvalidated External Exchange Rate Data Usage	Fee Abstraction Module by Notional	3 HIGH	3 HIGH	2 MEDIUM	ACKNOWLEDGED
Unnecessary Host Zone Freezing Due to Lack of Granular Error Handling	Fee Abstraction Module by Notional	2 MEDIUM	2 MEDIUM	2 MEDIUM	REPORTED
Unspecified recovery address for Crosschain-swaps IBC transfer	Fee Abstraction Module by Notional	2 MEDIUM	3 HIGH	1 LOW	REPORTED
Missing Fee Collector Account Check in abstractionDeductFee Handler	Fee Abstraction Module by Notional	1 LOW	1 LOW	1 LOW	ACKNOWLEDGED
Inaccurate Time Calculation for ICQ TWAP Request	Fee Abstraction Module by Notional	1 LOW	1 LOW	1 LOW	ACKNOWLEDGED

Title	Project	Severity	Impact	Exploitability	Status
Optimize Fee Calculation Process	Fee Abstraction Module by Notional	0 INFORMATIONAL	0 NONE	0 NONE	RISK ACCEPTED
Minor Code Improvements	Fee Abstraction Module by Notional	0 INFORMATIONAL	0 NONE	0 NONE	REPORTED
Optimization Opportunities in KVStore Calls	Fee Abstraction Module by Notional	0 INFORMATIONAL	0 NONE	1 LOW	REPORTED
Code Quality and Naming Improvements	Fee Abstraction Module by Notional	0 INFORMATIONAL	0 NONE	0 NONE	REPORTED

Critical Risk of Non-Uniqueness in KVStore Keys

Title	Critical Risk of Non-Uniqueness in KVStore Keys
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	4 CRITICAL
Impact	3 HIGH
Exploitability	3 HIGH
Status	ACKNOWLEDGED
Issue	https://github.com/osmosis-labs/fee-abstraction/issues/95

Involved artifacts

- x/feeabs/types/keys.go

Description

The existing system contains a significant risk of non-uniqueness for keys: `GetKeyHostZoneConfigByFeeabsIBCDenom`, `GetKeyHostZoneConfigByOsmosisIBCDenom`, and `GetKeyTwapExchangeRate` - keys.go#L38-L48:

```
func GetKeyHostZoneConfigByFeeabsIBCDenom(feeabsIbcDenom string) []byte {
    return append(KeyHostChainChainConfigByFeeAbs, []byte(feeabsIbcDenom)...)
}

func GetKeyHostZoneConfigByOsmosisIBCDenom(osmosisIbcDenom string) []byte {
    return append(KeyHostChainChainConfigByOsmosis, []byte(osmosisIbcDenom)...)
}

func GetKeyTwapExchangeRate(ibcDenom string) []byte {
    return append(OsmosisTwapExchangeRate, []byte(ibcDenom)...)
}
```

The absence of uniqueness in these keys poses a severe risk to the system's integrity and proper functionality. These non-unique keys may lead to various critical issues within the system, such as sending incorrect Interchain Query (ICQ) requests, miscalculations of native coins with incorrect TWAP rate values, and erroneous transfers of native coins to the fee-collector account.

For instance, if Osmosis's pools #3 (AKT/OSMO) and #4 (AKT/ATOM) share identical `KeyHostChainChainConfigByFeeAbs` and `OsmosisTwapExchangeRate` keys, the system could inadvertently send requests for pool #4 instead of pool #3. This misdirection of system actions can result in financial discrepancies and disrupt the normal operations of the system.

Problem Scenarios

The non-uniqueness of keys like `GetKeyHostZoneConfigByFeeabsIBCDenom`, `GetKeyHostZoneConfigByOsmosisIBCDenom`, and `GetKeyTwapExchangeRate` leads to critical scenarios where the system might operate incorrectly:

1. **Incorrect ICQ Requests**: Due to the non-uniqueness of keys, the system may send ICQ requests with incorrect denominations, potentially querying the wrong data or blockchain. This could result in inaccurate information and decisions made by the system.
2. **Native Coin Calculation Errors**: The miscalculation of native coins, as a result of using the wrong TWAP rate value, can lead to financial discrepancies. Users or the system could end up with less or more native coins than expected, affecting transactions and holdings.
3. **Fee Collection Errors**: Sending the wrong amount of native coins to the fee-collector account due to key non-uniqueness can disrupt fee collection processes. This could lead to financial losses and operational inefficiencies.

Recommendation

To mitigate the risk of non-uniqueness in system keys, we recommend to consider introducing the `poolId` in the key structure for keys: `GetKeyHostZoneConfigByFeeabsIBCDenom`, `GetKeyHostZoneConfigByOsmosisIBCDenom`, and `GetKeyTwapExchangeRate`. This addition will ensure key uniqueness and prevent issues related to incorrect data retrieval or operations.

Outdated Exchange Rate Risk Due to Potential IBC Relayer Delays

Title	Outdated Exchange Rate Risk Due to Potential IBC Relayer Delays
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	3 HIGH
Impact	3 HIGH
Exploitability	2 MEDIUM
Status	ACKNOWLEDGED
Issue	https://github.com/osmosis-labs/fee-abstraction/issues/94

Involved artifacts

- x/feeabs/epoch.go
- x/feeabs/ibc.go

Description

The module utilizes the Inter-Blockchain Communication (IBC) protocol to obtain a Time-Weighted Average Price (TWAP) from the Osmosis chain, which is then used to calculate transaction fees in IBC tokens. The exchange rate information is updated at the end of every epoch, as illustrated in the code snippet below:

```
func (k Keeper) AfterEpochEnd(ctx sdk.Context, epochIdentifier string) {
    switch epochIdentifier {
    case types.DefaultQueryEpochIdentifier:
        k.Logger(ctx).Info("Epoch interchain query TWAP")
        k.executeAllHostChainTWAPQuery(ctx)
    // ...
    }
}
```

In the event of a timeout during the IBC query, the system has a built-in retry mechanism intended to resend the query as illustrated in the following code snippet:

```
func (am IBCModule) OnTimeoutPacket(
    ctx sdk.Context,
    packet channeltypes.Packet,
    relayer sdk.AccAddress,
) error {
    // ...
    // Resend request if timeout
    err := am.keeper.OnTimeoutPacket(ctx)
    if err != nil {
        am.keeper.Logger(ctx).Error(fmt.Sprintf("Error OnTimeoutPacket %s",
err.Error()))
    // ...
    }
    return nil
}
```

This retry mechanism is triggered in the `OnTimeoutPacket` method, which is invoked whenever a timeout occurs while waiting for a response from the Osmosis chain.

Problem Scenarios

In the event of a delay or failure in IBC relayer functionality, the system may be unable to obtain the latest TWAP from the Osmosis chain, leading to the utilization of outdated exchange rate information for fee calculations. IBC relayers are critical for facilitating communication between different blockchains, and their performance can be impacted by network congestion, software bugs, or other operational issues. An outdated exchange rate may result in incorrect fee amounts being charged to users, which could have financial implications and potentially erode trust in the system. On the other side, it could also harm the system itself if the users are paying less than the actual current price should be.

Recommendation

1. **Pausing the system operation**:
    - If such delays occur, the system is in danger of calculating prices with outdated exchange rate, bringing both the users and the system to risk of unfair prices. The system should be paused to prevent loss for the system or users.
2. **User Notification**:
    - In the event of prolonged delays, notify users about the potential outdated exchange rate and its implications, providing them with the option to proceed or cancel their transactions.
3. **Alternative Data Sources**:
    - Explore the possibility of utilizing alternative data sources for exchange rate information to reduce dependency on a single external chain.
4. **Real-Time Monitoring and Alerting**:
    - Implement real-time monitoring and alerting mechanisms to detect and notify administrators of relayer delays or failures promptly, enabling quicker resolution.
5. **Retry Logic Enhancement**:
    - Expand the retry logic to include escalating attempts to obtain the latest exchange rate information, and potentially employ exponential backoff strategies to manage retries efficiently.

These recommendations aim to address the identified risk of outdated exchange rate information, improve system resilience, and enhance the overall user experience.

Lack of Frozen Status Check in Cross-chain Swap Execution

Title	Lack of Frozen Status Check in Cross-chain Swap Execution
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	3 HIGH
Impact	3 HIGH
Exploitability	2 MEDIUM
Status	RISK ACCEPTED
Issue	

Involved artifacts

- x/feeabs/msgserver.go
- x/feeabs/ibc.go

Description

The system exhibits inconsistent behavior in handling frozen host zones when executing cross-chain swaps. In the `executeAllHostChainSwap` method, a check is performed to skip the operation for frozen host zones, as shown below:

```
if hostZoneConfig.Frozen {
    return false
}
```

However, in the `SwapCrossChain` method triggered via the message server, there is no similar check for the frozen status, and the method proceeds to attempt the cross-chain swap regardless:

```
err = k.transferOsmosisCrosschainSwap(ctx, hostChainConfig)
```

This lack of a frozen status check can potentially lead to unintended transactions on frozen host zones, which could have adverse impacts.

Problem Scenarios

Suppose a host zone is frozen due to an error or an abnormal condition detected in the system. Despite this, a user or another system component triggers a cross-chain swap via the message server using the `SwapCrossChain` method. Since there's no check for the frozen status, the method would proceed to execute the cross-chain swap on the frozen host zone, possibly leading to unintended transactions, resource wastage, or even financial loss.

A test could be constructed to simulate a frozen host zone, then trigger a cross-chain swap via the `SwapCrossChain` method to observe and validate the behavior, demonstrating the lack of frozen status check and the potential adverse impact.

Recommendation

Implement Frozen Status Check:

- Update the `SwapCrossChain` method to include a check for the frozen status of the host zone before proceeding with the cross-chain swap, similar to the check in the `executeAllHostChainSwap` method.

```
if hostChainConfig.Frozen {
    return &types.MsgSwapCrossChainResponse{}, types.ErrHostZoneFrozen
}
```

Consistent Error Handling:

- Ensure consistent error handling logic across all methods and components that interact with host zones, to prevent unintended behaviors and maintain system integrity.

Testing and Validation:

- Conduct thorough testing to validate the updated logic under various scenarios, ensuring that the system behaves as expected and successfully prevents unintended transactions on frozen host zones.

Logging and Monitoring:

- Enhance logging and monitoring to capture and alert on attempts to execute operations on frozen host zones, aiding in early detection and resolution of issues.

User Feedback:

- Provide clear feedback to users when an operation is blocked due to a host zone being frozen, improving user experience and understanding of system behavior.

These recommendations aim to address the inconsistency in handling frozen host zones, reduce the risk of unintended transactions, and improve the overall robustness and user experience of the system.

Status Note

The development team is planning to remove the message server from the product version thus won’t be resolving this issue.

Missing Frozen Host Zone Checks in AnteHandles

Title	Missing Frozen Host Zone Checks in AnteHandles
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	3 HIGH
Impact	3 HIGH
Exploitability	2 MEDIUM
Status	ACKNOWLEDGED
Issue	https://github.com/osmosis-labs/fee-abstraction/issues/96

Involved artifacts

- x/feeabs/ante/decorate.go
- x/feeabs/keeper/keeper.go

Description

The `FeeAbstractionDeductFeeDecorator` and `FeeAbstractionMempoolFeeDecorator` `AnteHandles` lack the necessary checks to verify whether host zones are frozen. These `AnteHandles` calculate native coins from IBC coins (DeductFee, Mempool), and they potentially supply outdated Twap Rates for these calculations. The absence of frozen host zone checks in these `AnteHandles` creates the risk of incorrect calculations, which could result in the wrong amount of native coins being sent from the `feeabs` module to the `fee_collector`.

Problem Scenarios

The issue becomes evident when transactions involving `FeeAbstractionDeductFeeDecorator` and `FeeAbstractionMempoolFeeDecorator` `AnteHandles` are processed. In the absence of frozen host zone checks, the following problems may arise:

1. **Incorrect Coin Calculations:** If a host zone is frozen, the rate information might be outdated or unavailable. Without checks, the `AnteHandles` could proceed with calculations based on erroneous or outdated data, leading to incorrect coin conversions.
2. **Wrong Amounts Sent:** The lack of frozen host zone checks can result in `FeeAbstractionDeductFeeDecorator` `AnteHandle` sending the wrong amount of native coins

from the `feeabs` module to the `fee_collector` account, which could have financial implications for the system.

Recommendation

To mitigate the risk of incorrect calculations and ensure the accuracy of coin conversions in the `FeeAbstractionDeductFeeDecorator` and `FeeAbstractionMempoolFeeDecorator` `AnteHandles`, we recommend the following:

1. **Add Frozen Host Zone Checks:** Integrate checks within the `AnteHandles` to verify whether host zones are frozen before performing coin calculations. This check should ensure that calculations are based on up-to-date and accurate rate information.
2. **Error Handling:** Implement robust error handling mechanisms to address cases where host zones are frozen, ensuring that transactions are appropriately halted or altered to prevent erroneous calculations.
3. **Testing and Validation:** After making the necessary code modifications, conduct thorough testing and validation to verify that the checks are effectively preventing incorrect coin conversions in scenarios involving frozen host zones.
4. **Documentation:** Update the codebase with clear documentation outlining the addition of frozen host zone checks in the `AnteHandles`. This documentation will assist developers and maintainers in understanding the changes made to address the issue.

By introducing frozen host zone checks in these `AnteHandles`, the system can ensure the accuracy of coin calculations, minimize the risk of incorrect financial transactions, and enhance the overall integrity of the fee abstraction process. These recommendations aim to improve system reliability and financial accuracy.

Unvalidated External Exchange Rate Data Usage

Title	Unvalidated External Exchange Rate Data Usage
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	3 HIGH
Impact	3 HIGH
Exploitability	2 MEDIUM
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- x/feeabs/keeper.go
- x/feeabs/ibc.go

Description

The system relies on Time-Weighted Average Price (TWAP) exchange rate data obtained from an external blockchain, Osmosis, to perform essential financial calculations. This data is fetched through interchain queries over IBC, specifically in the `OnAcknowledgementPacket` function. The `twapRate` is then saved to the store using the `SetTwapRate` function, and later used in the `CalculateNativeFromIBCCoins` function to convert IBC Coins to native coins. A snippet of the code for saving and retrieving the `twapRate` is shown below:

```
func (k Keeper) SetTwapRate(ctx sdk.Context, ibcDenom string, osmosisTWAPExchangeRate
sdk.Dec) {
    store := ctx.KVStore(k.storeKey)
    bz, _ := osmosisTWAPExchangeRate.Marshal()
    key := types.GetKeyTwapExchangeRate(ibcDenom)
    store.Set(key, bz)
}

twapRate, err := k.GetTwapRate(ctx, chainConfig.IbcDenom)
nativeFeeAmount := twapRate.MulInt(coin.Amount).RoundInt()
```

Problem Scenarios

The system does not implement any validation checks to ensure the rationality or correctness of the `twapRate` data obtained from Osmosis. This lack of validation could potentially lead to inaccurate or unrealistic financial calculations, as outlined in the following scenario:

1. The system fetches the `twapRate` from Osmosis through an interchain query.
2. Due to an anomaly, error, or malicious manipulation on Osmosis, an incorrect or unrealistic `twapRate` value is returned.
3. The system saves and later uses this incorrect `twapRate` in the `CalculateNativeFromIBCCoins` function without any validation.
4. The incorrect `twapRate` leads to inaccurate conversion of IBC Coins to native coins, potentially causing financial discrepancies or losses.

Recommendation

To address this vulnerability and ensure the accuracy and reliability of financial calculations, the following steps are recommended:

1. **Validation Checks:** Implement validation checks on the `twapRate` data obtained from Osmosis to ensure it falls within a reasonable and expected range. This could include checks against predefined upper and lower bounds, or comparison with other reliable sources of exchange rate data.
2. **Error Handling:** Enhance error handling to catch and manage any issues during the `twapRate` fetching and validation process. This could include logging errors for further investigation and alerting administrators for manual intervention.
3. **Fallback Mechanism:** Establish a fallback mechanism to use a reliable, internally sourced or previously validated, exchange rate in case the data from Osmosis is found to be erroneous or falls outside the acceptable range.

These measures would significantly enhance the system's defense against erroneous external data, thereby enhancing its financial accuracy and reliability.

Unnecessary Host Zone Freezing Due to Lack of Granular Error Handling

Title	Unnecessary Host Zone Freezing Due to Lack of Granular Error Handling
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	2 MEDIUM
Impact	2 MEDIUM
Exploitability	2 MEDIUM
Status	REPORTED
Issue	

Involved artifacts

- x/feeabs/ibc.go

Description

The system, as per the provided code snippets, exhibits a behavior where all host zones are attempted to be frozen in the event of an error acknowledgment received from inter-chain queries. The relevant logic is encapsulated in the `OnAcknowledgementPacket` method and the helper method `FrozenHostZoneByIBCDenom`. Below is the code snippet from `OnAcknowledgementPacket` method where the system iterates over all host zones to freeze them on encountering an error acknowledgment:

```
case *channeltypes.Acknowledgement_Error:
    k.IterateHostZone(ctx, func(hostZoneConfig types.HostChainFeeAbsConfig) (stop
bool) {
        err := k.FrozenHostZoneByIBCDenom(ctx, hostZoneConfig.IbcDenom)
        if err != nil {
            k.Logger(ctx).Error(fmt.Sprintf("Failed to frozen host zone %s",
err.Error()))
        }
        return false
    })
    k.Logger(ctx).Error(fmt.Sprintf("failed to send packet ICQ request %v",
resp.Error))
```

Problem Scenarios

The current approach of freezing all host zones upon receiving an error acknowledgment can lead to a complete halt in system operations even if the error occurs to a single host zone. This approach lacks granularity and could significantly disrupt the user experience and system functionality.

Recommendation

Enhanced Acknowledgment Structure:

- Modify the acknowledgment packet structure to include detailed error information, specifically identifying the affected host zone. This will allow for more targeted error handling.

Granular Error Handling:

- Update the error handling logic in the `OnAcknowledgementPacket` method to parse the enhanced acknowledgment, identify the affected host zone, and freeze only that specific host zone.

```
case *channeltypes.Acknowledgement_Error:
    errInfo := resp.ErrorInfo
    if errInfo != nil {
        err := k.FrozenHostZoneByIBCDenom(ctx, errInfo.HostZoneID)
        if err != nil {
            k.Logger(ctx).Error(fmt.Sprintf("Failed to freeze host zone %s: %s",
 errInfo.HostZoneID, err.Error()))
        }
    }
    // ...
```

Testing and Validation:

- Conduct extensive testing to validate the new error handling logic under various scenarios to ensure it behaves as expected and successfully isolates the impact of errors.

User Communication:

- In the event of host zone freezing, communicate the status and any potential impact to users, maintaining transparency and managing user expectations.

Unspecified recovery address for Crosschain-swaps IBC transfer

Title	Unspecified recovery address for Crosschain-swaps IBC transfer
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	2 MEDIUM
Impact	3 HIGH
Exploitability	1 LOW
Status	REPORTED
Issue	

Involved artifacts

- x/feeabs/types/build_memo.go

Description

The Fee Abstraction Module is designed to facilitate the swapping of IBC tokens for native tokens via Osmosis' smart contracts, specifically utilizing the Crosschain-swaps contract. This contract, in turn, leverages another contract named Swaprouter to execute the actual token swap. Post-swap, the tokens are moved to an Osmosis managed account, awaiting transfer back to the original sender in the form of native tokens.

A critical component in this process is the memo field in the IBC transfer, which carries essential information for a successful transfer, swap, and receipt, including a particular entry named `on_failed_delivery`. As per Osmosis explanation in README.md:
”The `on_failed_delivery` field can be set to `do_nothing` or a local recovery addr via `{"local_recovery_addr": "osmo1..."}`. If set to `do_nothing`, the contract will not track the packet, and the user will not be able to recover the funds if the packet fails. If set to a local recovery addr, the contract will track the packet, and the specified address will be able to execute `{"recover": {}}` on the crosschain swaps contract to recover the funds.”

In the Fee Abstraction Module, the memo for the transfer is generated as follows::

```
func BuildCrossChainSwapMemo(outputDenom string, contractAddress string,
receiverAddress string, chainName string) (string, error) {
    receiver := fmt.Sprintf("%s/%s", chainName, receiverAddress)
    swap := Swap{
        OutPutDenom: outputDenom,
        Slippage: Twap{
            Twap: TwapRouter{
                SlippagePercentage: "20",
                WindowSeconds:      10,
            },
        },
        Receiver:          receiver,
        OnFailedDelivery: "do_nothing",
    }
...
```

Here, the recovery action is consistently set to `do_nothing`, which presents a risk of fund loss if the IBC transfer from Osmosis' Crosschain-swaps to the originating chain fails.

Problem Scenarios

Based on the Osmosis' Crosschain-swaps contract documentation, a failed IBC transfer could unfold as follows:

1. The Fee Abstraction Module initiates an IBC transfer to the Osmosis chain for executing a token swap.
2. The swap occurs, and a new IBC transfer from Osmosis back to the sender chain is triggered by the smart contract.
3. The transfer fails, and the packet is lost.
4. Due to the absence of a specified recovery address in the memo field, there's no mechanism to return the funds back to the original sender.

Recommendation

To mitigate the risk of fund loss due to transfer failures, it's advisable to specify a recovery address in the `on_failed_delivery` field, rather than leaving it as `do_nothing`. This could be the module's address or another designated account address to handle failed IBC transfers from Osmosis back to the sender chain, thereby providing a fallback mechanism to recover funds in adverse scenarios.

Missing Fee Collector Account Check in abstractionDeductFeeHandler

Title	Missing Fee Collector Account Check in abstractionDeductFeeHandler
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	1 LOW
Impact	1 LOW
Exploitability	1 LOW
Status	ACKNOWLEDGED
Issue	https://github.com/osmosis-labs/fee-abstraction/issues/97

Involved artifacts

- x/feeabs/ante/decorate.go

Description

The `abstractionDeductFeeHandler` lacks a check to verify if the `fee_collector` module account has been set. In contrast, such a check exists in the `normalDeductFeeAnteHandle` - decorate.go#L55-L57:

```
if addr := fadfd.accountKeeper.GetModuleAddress(types.FeeCollectorName); addr == nil
{
    return ctx, fmt.Errorf("fee collector module account (%s) has not been set",
types.FeeCollectorName)
}
```

The absence of this check in the `abstractionDeductFeeHandler` can potentially lead to a panic when sending coins from the `feeabs` module to the `fee_collector` in the `DeductFees` function - decorate.go#166

```
err := bankKeeper.SendCoinsFromAccountToModule(ctx, accAddress,
types.FeeCollectorName, fees)
```

This scenario could result in a panic if the `fee_collector` module account is not set.

Problem Scenarios

The issue manifests when processing transactions through the `abstractionDeductFeeHandler`. In cases where the `fee_collector` module account has not been set, the `DeductFees` function attempts to send coins from the `feeabs` module to the `fee_collector`. This action can result in a panic, as the necessary `fee_collector` module account does not exist.

Recommendation

We recommended to introduce a check in the `abstractionDeductFeeHandler` to verify the existence of the `fee_collector` module account before attempting to send coins to it. This check should be consistent with the one present in the `normalDeductFeeAnteHandle`.

Inaccurate Time Calculation for ICQ TWAP Request

Title	Inaccurate Time Calculation for ICQ TWAP Request
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	1 LOW
Impact	1 LOW
Exploitability	1 LOW
Status	ACKNOWLEDGED
Issue	https://github.com/osmosis-labs/fee-abstraction/issues/98

Involved artifacts

- x/feeabs/keeper/abci.go
- x/feeabs/keeper/ibc.go

Description

The codebase contains an issue related to the calculation of the time for which the Inter-Chain Query (ICQ) request is created for Time-Weighted Average Price (TWAP). This issue can lead to incorrect TWAP calculations, especially when the transition between epochs occurs.

Problem Scenarios

The problem arises from the method of calculating the ICQ TWAP request start time. This calculation is currently based on the block time minus the epoch duration. However, this approach may result in incorrect start times for the ICQ TWAP requests, particularly when an epoch transition happens.

New epoch start time (abci.go#L42):

```
epochEndTime := epochInfo.CurrentEpochStartTime.Add(epochInfo.Duration)
```

ICQ TWAP start time (ibc.go#L264):

```
startTime := ctx.BlockTime().Add(-queryTwapEpochInfo.Duration)
```

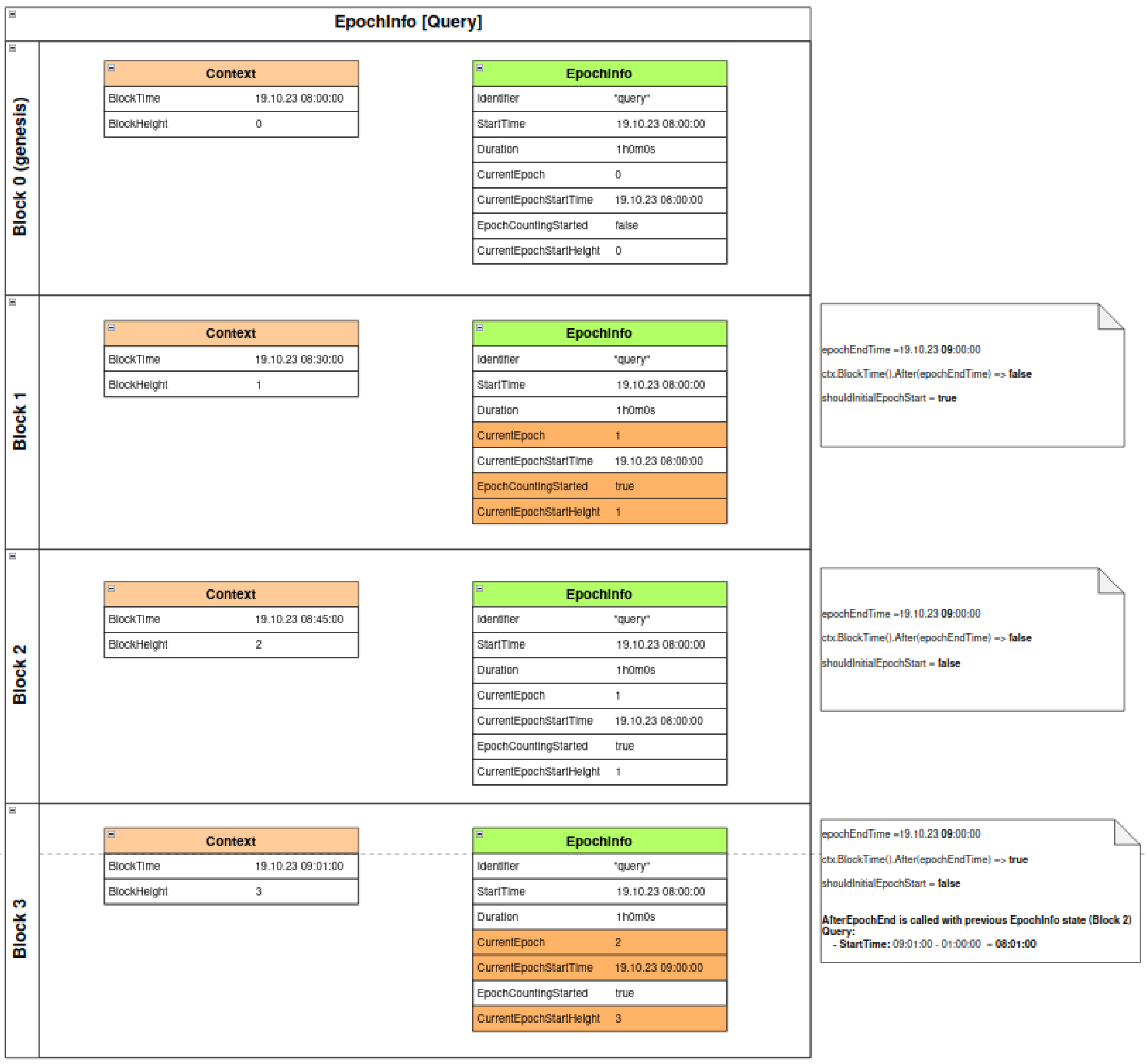
For example, consider two epochs, #1 and #2:

- Epoch #1 starts at 10:00 and spans five blocks:
  - Block #1 (block time: 10:00)
  - Block #2 (block time: 10:09)

    - Block #3 (block time: 10:30)
    - Block #4 (block time: 10:45)
    - Block #5 (block time: 10:55)
- Block #6, which is outside of epoch #1, is created at 11:10, initiating epoch #2. However, the start time for epoch #2 is calculated as 11:00 since the epoch duration is set to 1 hour.

After the epoch transition (from epoch #1 to epoch #2), the `AfterEpochEnd` function is called, and an ICQ TWAP request is created based on the current block time and the epoch duration (11:10 - 1 hour -> 10:10). This means that the `ArithmeticTwapToNow` would be calculated for the period from 10:10 to 11:10, effectively excluding blocks #1 and #2 from the TWAP calculation.

Below is a diagram of the change of EpochInfo by Blocks, on which the problem is also shown.



EpochInfo [Query]
Block 0 (genesis)
Context
BlockTime 19.10.23 08:00:00
BlockHeight 0
EpochInfo
Identifier "query"
StartTime 19.10.23 08:00:00
Duration 1h0m0s
CurrentEpoch 0
CurrentEpochStartTime 19.10.23 08:00:00
EpochCountingStarted false
CurrentEpochStartHeight 0
Block 1
Context
BlockTime 19.10.23 08:30:00
BlockHeight 1
EpochInfo
Identifier "query"
StartTime 19.10.23 08:00:00
Duration 1h0m0s
CurrentEpoch 1
CurrentEpochStartTime 19.10.23 08:00:00
EpochCountingStarted true
CurrentEpochStartHeight 1
epochEndTime =19.10.23 09:00:00
ctx.BlockTime().After(epochEndTime) => false
shouldInitialEpochStart = true
Block 2
Context
BlockTime 19.10.23 08:45:00
BlockHeight 2
EpochInfo
Identifier "query"
StartTime 19.10.23 08:00:00
Duration 1h0m0s
CurrentEpoch 1
CurrentEpochStartTime 19.10.23 08:00:00
EpochCountingStarted true
CurrentEpochStartHeight 1
epochEndTime =19.10.23 09:00:00
ctx.BlockTime().After(epochEndTime) => false
shouldInitialEpochStart = false
Block 3
Context
BlockTime 19.10.23 09:01:00
BlockHeight 3
EpochInfo
Identifier "query"
StartTime 19.10.23 08:00:00
Duration 1h0m0s
CurrentEpoch 2
CurrentEpochStartTime 19.10.23 09:00:00
EpochCountingStarted true
CurrentEpochStartHeight 3
epochEndTime =19.10.23 09:00:00
ctx.BlockTime().After(epochEndTime) => true
shouldInitialEpochStart = false
AfterEpochEnd is called with previous EpochInfo state (Block 2)
Query:
- StartTime: 09:01:00 - 01:00:00 = 08:01:00

1 Changes "query" EpochInfo by Blocks

Recommendation

To address this issue we propose one of the following solutions:

Reevaluate Start Time Calculation: Consider changing the method for calculating the start time for ICQ TWAP requests. This could involve basing the calculation on epoch transitions rather than block time. Or,

Review Epoch Transition Logic: Review the logic responsible for transitioning between epochs and ensure that the start time of a new epoch is based on the current block time. This can help align epoch boundaries with actual time.

Optimization Opportunities in KVStore Calls

Title	Optimization Opportunities in KVStore Calls
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	1 LOW
Status	REPORTED
Issue	

Involved artifacts

- x/feeabs/ante/decorate.go
- x/feeabs/ibc.go
- x/feeabs/keeper/config.go
- x/feeabs/keeper/proposal.go

Description

During the code review, it was identified that there are several places within the codebase where KVStore calls could be optimized. These optimizations can help reduce redundant queries and improve the efficiency of the code.

Problem Scenarios

The issues regarding KVStore calls and their potential optimizations are as follows:

1. **Redundant Calls to HasHostZoneConfig and GetHostZoneConfig:** In some parts of the code, there are redundant calls to `HasHostZoneConfig` followed by `GetHostZoneConfig` (decorate.go#L45-L50, decorate.go#L254-L256).

```
hasHostChainConfig := fadfd.feeabsKeeper.HasHostZoneConfig(ctx, feeDenom)
if !hasHostChainConfig {
    return fadfd.normalDeductFeeAnteHandle(ctx, tx, simulate, next, feeTx)
}

hostChainConfig, _ := fadfd.feeabsKeeper.GetHostZoneConfig(ctx, feeDenom)
```

   This is unnecessary, as it's sufficient to call `GetHostZoneConfig` and check the 'found' field to determine if the key exists.

```
hostChainConfig, found := fadfd.feeabsKeeper.GetHostZoneConfig(ctx, feeDenom)
if !found {
    return fadfd.normalDeductFeeAnteHandle(ctx, tx, simulate, next, feeTx)
}
```

The same can be found for EpochInfo at ibc.go#L256-L263.

2. **Unnecessary Use of GetHostZoneConfig:** In certain instances, `GetHostZoneConfig` is called to check the existence of a key. It's more efficient to direc tly000 call `HasHostZoneConfig` (proposal.go#L10-L13, proposal.go#L24-L27, proposal.go#L38-L41).
3. **Double Call to GetHostZoneConfig:** In the `DeleteHostZoneProposal` and `DeleteHostZoneConfig` functions, `GetHostZoneConfig` is called twice, leading to redundant queries. This can be optimized by calling `GetHostZoneConfig` once, potentially in the `DeleteHostZoneConfig` function (proposal.go#L24-L27, config.go#L62).

Recommendation

As explained above.

These optimizations aim to improve code efficiency, reduce redundant queries, and enhance the overall performance and maintainability of the codebase.

Optimize Fee Calculation Process

Title	Optimize Fee Calculation Process
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Status	RISK ACCEPTED
Issue	

Involved artifacts

- x/feeabs/ante/decorate.go

Description

In the process of calculating native coin fees, the coins first get split between the ones with zero and non-zero amounts, as it is pretty clear according to the variable naming in the code and the code semantics:

```
// split feeRequired into zero and non-zero coins(nonZeroCoinFeesReq,
zeroCoinFeesDenomReq), split feeCoins according to
        // nonZeroCoinFeesReq, zeroCoinFeesDenomReq,
        // so that feeCoins can be checked separately against them.
        nonZeroCoinFeesReq, zeroCoinFeesDenomReq := getNonZeroFees(feeRequired)

        // feeCoinsNonZeroDenom contains non-zero denominations from the feeRequired
        // feeCoinsNonZeroDenom is used to check if the fees meets the requirement
imposed by nonZeroCoinFeesReq
        // when feeCoins does not contain zero coins' denoms in feeRequired
        feeCoinsNonZeroDenom, feeCoinsZeroDenom := splitCoinsByDenoms(feeCoins,
zeroCoinFeesDenomReq)

        feeCoinsLen := feeCoins.Len()
```

However, few lines after, the calculation is performed over the complete set of coins by calling the `CalculateNativeFromIBCCoins` with `feeCoins` as a parameter:

```
if feeCoinsNonZeroDenom.Len() == 1 {
            feeDenom := feeCoinsNonZeroDenom.GetDenomByIndex(0)
            hasHostChainConfig := famfd.feeabsKeeper.HasHostZoneConfig(ctx, feeDenom)
            if hasHostChainConfig {
                hostChainConfig, _ := famfd.feeabsKeeper.GetHostZoneConfig(ctx,
feeDenom)
                nativeCoinsFees, err :=
famfd.feeabsKeeper.CalculateNativeFromIBCCoins(ctx, feeCoins, hostChainConfig)
                if err != nil {
                    return ctx, sdkerrors.Wrapf(errorstypes.ErrInsufficientFee,
"insufficient fees")
                }
                feeCoinsNonZeroDenom = nativeCoinsFees
            }
        }
```

The code could be optimized by calling the function over only non zero coins, thus saving time and memory, as the zero amount coins won't affect the end result.

Recommendation

As explained above.

Status note

The team plans to enable multiple denoms to be selected for fee payments, and thus leaves the whole `feeCoins` set to be processed, even though it now includes only one non-zero denom.

Minor Code Improvements

Title	Minor Code Improvements
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Status	REPORTED
Issue	

Involved artifacts

- x/feeabs/ante/decorate.go
- x/feeabs/ibc.go

Description

During the code review, several minor code improvements were identified in the Fee Abstraction module. These improvements aim to enhance code readability, maintainability, and adherence to best practices.

Problem Scenarios

The identified code improvements encompass several areas:

1. The variable `feeTx.GetFee()` is defined as `fee` (decorate.go#59) in the `normalDeductFeeAnteHandle` function, but it is not consistently used as such in the code. Instead, the code continues to use `feeTx.GetFee()`, which is unnecessary (decorate.go#86, decorate.go#87, decorate.go#94).
2. Similar to the previous point, in the `abstractionDeductFeeHandler` function, the variable `feeTx.GetFee()` is defined as `fee` (decorate.go#102) and should be used in various places (decorate.go#129 - already defined, decorate.go#140, decorate.go#153).
3. In the `DeductFees` function, it is recommended to use `fees.Validate` instead of `fees.IsValid` for accurate error propagation (decorate.go#L162-L164). Using `Validate` preserves error messages, providing better context in case of validation failures.

4. The `params.OsmosisQueryTwapPath` (ibc.go#L67) value can be supplied before the `for` loop or potentially passed as a function parameter, considering that in the `handleOsmosisIbcQuery` function, the `params` are already used (ibc.go#L267). This change improves code efficiency.
5. It is advised to define a constant for the value `time.Minute * 5` to enhance code maintainability and avoid redundancy (ibc.go#L87, ibc.go#L227).
6. Redundant and maybe incorrect use of `IsZero` in this code. First, even if fee is zero, the event gets emitted:

```
// deduct the fees
    if !feeTx.GetFee().IsZero() {
        err = DeductFees(fadfd.bankKeeper, ctx, deductFeesFrom, feeTx.GetFee())
        if err != nil {
            return ctx, err
        }
    }

    events := sdk.Events{sdk.NewEvent(sdk.EventTypeTx,
        sdk.NewAttribute(sdk.AttributeKeyFee, feeTx.GetFee().String()),
    )}
    ctx.EventManager().EmitEvents(events)
```

It would be more suitable to emit special event, or an error in case of zero fee.
In addition to that, the check of the fee amount is also executed inside `DeductFees` when `Validate` function is being called:

```
// Validate checks that the Coins are sorted, have positive amount, with a
valid and unique
// denomination (i.e no duplicates). Otherwise, it returns an error.
func (coins Coins) Validate() error {
    switch len(coins) {
    case 0:
        return nil

    case 1:
        if err := ValidateDenom(coins[0].Denom); err != nil {
            return err
        }
        if !coins[0].IsPositive() {
            return fmt.Errorf("coin %s amount is not positive", coins[0])
        }
        return nil
...
```

7. Replace this code:

```
fee := feeTx.GetFee()
    if len(fee) == 0 {
        return fadfd.normalDeductFeeAnteHandle(ctx, tx, simulate, next, feeTx)
    }
    feeDenom := fee.GetDenomByIndex(0)
    hasHostChainConfig := fadfd.feeabsKeeper.HasHostZoneConfig(ctx, feeDenom)
    if !hasHostChainConfig {
        return fadfd.normalDeductFeeAnteHandle(ctx, tx, simulate, next, feeTx)
    }
```

with this to remove duplicate code:

```
fee := feeTx.GetFee()
feeDenom := ""

if len(fee) > 0 {
    feeDenom = fee.GetDenomByIndex(0)
}

if len(fee) == 0 || !fadfd.feeabsKeeper.HasHostZoneConfig(ctx, feeDenom) {
    return fadfd.normalDeductFeeAnteHandle(ctx, tx, simulate, next, feeTx)
}
```

8. Change this code:

```
if byPass {
    return next(ctx, tx, simulate)
}

if feeCoinsLen == 0 {
    if len(zeroCoinFeesDenomReq) != 0 {
        return next(ctx, tx, simulate)
    }
    return ctx, sdkerrors.Wrapf(errorstypes.ErrInsufficientFee, "insufficient
fees; got: %s required 12: %s", feeCoins, feeRequired)
}

if len(feeCoinsZeroDenom) > 0 {
    return next(ctx, tx, simulate)
}
```

to this to avoid duplicate code:

```
if byPass || (feeCoinsLen == 0 && len(zeroCoinFeesDenomReq) != 0) ||
len(feeCoinsZeroDenom) > 0 {
    return next(ctx, tx, simulate)
}

if feeCoinsLen == 0 {
    return ctx, sdkerrors.Wrapf(errorstypes.ErrInsufficientFee, "insufficient
fees; got: %s required 12: %s", feeCoins, feeRequired)
}
```

9. Redundant check here. Before every call of `CalculateNativeFromIBCCoins` (where this function is being called) there is a check to see if `ibcCoins.len == 1`, so one of the checks can be safely removed, either before the call or the one inside `verifyIBCCoins`.

Recommendation

As explained above.

These code improvements are intended to enhance code readability, maintainability, and adherence to best practices.

Code Quality and Naming Improvements

Title	Code Quality and Naming Improvements
Project	Fee Abstraction Module by Notional
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Status	REPORTED
Issue	

Involved artifacts

- x/feeabs/types/query.pb.go
- x/feeabs/types/keys.go
- x/feeabs/keeper/epoch.go
- x/feeabs/keeper/config.go

Description

During the code review, several minor code-related issues were identified, including typographical errors, incorrect comments, unused functions, and the need to correct function names for clarity.

Problem Scenarios

The specific issues within the codebase are as follows:

1. **Typo in Protobuf Definition:** The name `QueryHostChainConfigRespone` has a typographical error. It should be corrected to `QueryHostChainConfigResponse` to maintain consistency in naming conventions. This issue was found in the query.pb.go#78 file.
2. **Incorrect Comments:** There are instances where copy-paste have resulted in incorrect comments. These comments are located at keys.go#L32-L33.
3. **Unused Functions:** The functions `NumBlocksSinceEpochStart` (epoch.go#L111), `DeleteEpochInfo` (epoch.go#L68) and `IteratorHostZone` (config.go#L84) appear to be unused and can be safely removed.
4. **Function Name Corrections:** The function names `FrozenHostZoneByIBCDenom` and `UnFrozenHostZoneByIBCDenom` should be corrected to `FreezeHostZoneByIBCDenom` and `UnfreezeHostZoneByIBCDenom`, respectively, to improve clarity and adherence to naming conventions. These corrections are needed in the config.go file.

Recommendation

As explained above.

These recommendations aim to improve the codebase's correctness, clarity, and maintainability by addressing minor issues and inconsistencies.

Appendix: Vulnerability Classification

For classifying vulnerabilities identified in the findings of this report, we employ the simplified version of Common Vulnerability Scoring System (CVSS) v3.1, which is an industry standard vulnerability metric. For each identified vulnerability we assess the scores from the *Base Metric Group*, the Impact score, and the Exploitability score. The *Exploitability score* reflects the ease and technical means by which the vulnerability can be exploited. That is, it represents characteristics of the *thing that is vulnerable*, which we refer to formally as the *vulnerable component*. The *Impact score* reflects the direct consequence of a successful exploit, and represents the consequence to the *thing that suffers the impact*, which we refer to formally as the *impacted component*. In order to ease score understanding, we employ CVSS Qualitative Severity Rating Scale, and abstract numerical scores into the textual representation; we construct the final *Severity score* based on the combination of the Impact and Exploitability sub-scores.

As blockchains are a fast evolving field, we evaluate the scores not only for the present state of the system, but also for the state that deems achievable within 1 year of projected system evolution. E.g., if at present the system interacts with 1-2 other blockchains, but plans to expand interaction to 10-20 within the next year, we evaluate the impact, exploitability, and severity scores wrt. the latter state, in order to give the system designers better understanding of the vulnerabilities that need to be addressed in the near future.

Impact Score

The Impact score captures the effects of a successfully exploited vulnerability on the component that suffers the worst outcome that is most directly and predictably associated with the attack.

Impact Score	**Examples**
High	Halting of the chain; loss, locking, or unauthorized withdrawal of funds of many users; arbitrary transaction execution; forging of user messages / circumvention of authorization logic
Medium	Temporary denial of service / substantial unexpected delays in processing user requests (e.g. many hours/days); loss, locking, or unauthorized withdrawal of funds of a single user / few users; failures during transaction execution (e.g. out of gas errors); substantial increase in node computational requirements (e.g. 10x)
Low	Transient unexpected delays in processing user requests (e.g. minutes/a few hours); Medium increase in node computational requirements (e.g. 2x); any kind of problem that affects end users, but can be repaired by manual intervention (e.g. a special transaction)
⊜ **None**	Small increase in node computational requirements (e.g. 20%); code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation

Exploitability Score

The Exploitability score reflects the ease and technical means by which the vulnerability can be exploited; it represents the characteristics of the vulnerable component. In the below table we list, for each category, examples of actions by actors that are enough to trigger the exploit. In the examples below:

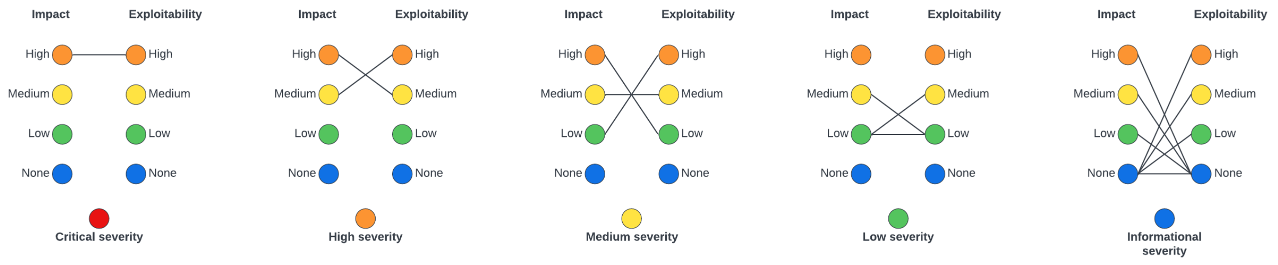
- *Actors* can be any entity that interacts with the system: other blockchains, system users, validators, relayers, but also uncontrollable phenomena (e.g. network delays or partitions).
- *Actions* can be

  - *legitimate*, e.g. submission of a transaction that follows protocol rules by a user; delegation/ redelegation/bonding/unbonding; validator downtime; validator voting on a single, but alternative block; delays in relaying certain messages, or speeding up relaying other messages;
  - *illegitimate*, e.g. submission of a specially crafted transaction (not following the protocol, or e.g. with large/incorrect values); voting on two different alternative blocks; alteration of relayed messages.
- We employ also a *qualitative measure* representing the amount of certain class of power (e.g. possessed tokens, validator power, relayed messages): *small* for < 3%; *medium* for 3-10%; *large* for 10-33%, *all* for >33%. We further quantify this qualitative measure as relative to the largest of the system components. (e.g. when two blockchains are interacting, one with a large capitalization, and another with a small capitalization, we employ *small* wrt. the number of tokens held, if it is small wrt. the large blockchain, even if it is large wrt. the small blockchain)

Exploitability Score	**Examples**
High	illegitimate actions taken by a small group of actors; possibly coordinated with legitimate actions taken by a medium group of actors
Medium	illegitimate actions taken by a medium group of actors; possibly coordinated with legitimate actions taken by a large group of actors
Low	illegitimate actions taken by a large group of actors; possibly coordinated with legitimate actions taken by all actors
▤ **None**	illegitimate actions taken in a coordinated fashion by all actors

Severity Score

The severity score combines the above two sub-scores into a single value, and roughly represents the probability of the system suffering a severe impact with time; thus it also represents the measure of the urgency or order in which vulnerabilities need to be addressed. We assess the severity according to the combination scheme represented graphically below.



As can be seen from the image above, only a combination of high impact with high exploitability results in a Critical severity score; such vulnerabilities need to be addressed ASAP. Accordingly, High severity score receive vulnerabilities with the combination of high impact and medium exploitability, or medium impact, but high exploitability.

Severity Score	**Examples**
▥ **Critical**	Halting of chain via a submission of a specially crafted transaction

Severity Score	Examples
High	Permanent loss of user funds via a combination of submitting a specially crafted transaction with delaying of certain messages by a large portion of relayers
Medium	Substantial unexpected delays in processing user requests via a combination of delaying of certain messages by a large group of relayers with coordinated withdrawal of funds by a large group of users
Low	2x increase in node computational requirements via coordinated withdrawal of all user tokens
▤ **Informational**	Code inefficiencies; bad code practices; lack/incompleteness of tests; lack/ incompleteness of documentation; any exploit for which a coordinated illegitimate action of all actors is necessary

Disclaimer