Security Audit Report

Q4 2024 Buyback & Burn

Authors: Darko Deuric

Last revised 25 December, 2024

Table of Contents

Audit Dashboard

Target Summary

- **Type**: Protocol and Implementation
- **Platform**: CosmosSDK, Golang
- **Artifacts:** https://github.com/Stride-Labs/stride/tree/buyback-and-burn

Auditors

- Darko Deuric

Engagement Summary

- **Dates**: 17 Dec 2024 - 23 Dec 2024.
- **Method**: Code Review

Severity Summary

Finding Severity	**#**
Critical	1
High	1
Medium	1
Low	
Informational	
Total	**3**

Threat Inspection

Auction Threat Model

- **Anyone can create or update auction** ✅
  Not possible; only admin address has permissions.
- **Auction name is not unique** ✅
  An auction with `AUC_NAME` cannot be created if there is already an auction with the same `AUC_NAME`.
- **Coins are not sold in FCFS manner** ✅
  The auction operates on a first-come, first-serve basis. When `PlaceBid` is called, the caller automatically wins if they satisfy the minimum selling conditions.
- **Auction relies on outdated prices** ✅
  Prices used for auctions generally couldn't be outdated due to regular ICQ updates.
- **Base denom can be bought with a token other than payment token (STRD)** ✅
  Here, `auction.PaymentDenom` is always enforced.
- **Place bid action fails** ❌
  - It fails due to:
    - `tokenPrice.UpdatedAt` not updated; the price is treated as stale after `PriceExpirationTimeoutSec`, which breaks the bidding logic. `GetTokenPriceForQuoteDenom` returns error
    - `bid.PaymentTokenAmount` incorrectly compared with the `auction.SellingDenom` token - has been resolved in a recent commit
    - tokenPrice.UpdatedAt Not Updated Leading to Potential Failures and Incorrect Unit Comparison in Auction Bid Validation for reference
- **More base denoms than intended are sold on auction** ✅
  Bidders cannot buy more than the auction balance, and they can buy exactly `bid.SellingTokenAmount`
- **Bidder overpays the selling denom** ❓
  It's possible to overpay without receiving a refund. This might be expected behavior, but it needs confirmation.
- **Bidder underpays the selling denom** ❌
  There is a check for bids that are too low, but the `LT` condition is reversed, allowing underpayment to succeed. Incorrect Price Validation Logic in Auction Bids for reference.

ICQ & Price Threat Model

- **Panic in BeginBlocker** ✅
  Generally, no direct code path leads to a panic.
- **Non-determinism in BeginBlocker** ✅
  The check for update intervals uses `currentTime := ctx.BlockTime()`, which is deterministic.
- **ICQ is never executed for certain** `TokenPrice` ✅
  ICQ should be executed at least once, if `lastUpdate.IsZero()`.
- **ICQ is not regularly executed for certain** `TokenPrice` ❌
  It's expected to execute every `UpdateIntervalSec` if `!tokenPrice.QueryInProgress`. However, `tokenPrice.UpdatedAt` is not updated, causing unexpected behavior. tokenPrice.UpdatedAt Not Updated Leading to Potential Failures for reference

- **Price is not expressed in the right units - wrong price interpretation** ?
  The multiplication `bid.SellingTokenAmount.ToLegacyDec().Mul(discountedPrice)` requires `discountedPrice` to be expressed in terms of `SellingDenom/PaymentDenom` (e.g., `uatom/ustrd`). This imposes a critical requirement: the spot prices must always be expressed as `baseDenom/quoteDenom` (e.g., `uatom/usdc` or `ustrd/usdc`) to ensure correct calculations.
- **Price of 1 baseToken in terms of quoteToken could be zero** ✓
  `baseTokenPrice / quoteTokenPrice` cannot be zero, as `GetTokenPriceForQuoteDenom` would fail

Findings

Finding	Severity	Status
Incorrect Price Validation Logic in Auction Bids	CRITICAL	RESOLVED
tokenPrice.UpdatedAt Not Updated Leading to Potential Failures	HIGH	RESOLVED
Incorrect Unit Comparison in Auction Bid Validation	MEDIUM	RESOLVED

Incorrect Price Validation Logic in Auction Bids

Description:
In the `fcfsBidHandler` function, the condition used to validate the bid price is incorrect. The logic currently uses `LT` (less than) to compare the bid's offered payment amount with the required minimum payment, which reverses the intended check. This effectively validates bids that underpay instead of ensuring that bids meet or exceed the required minimum.

Impact:
The reversed condition allows bids to pass validation even if they grossly underpay for the auctioned assets. As a result, a malicious user can purchase the entire auction balance for as little as **1 ustrd**, leading to a complete loss of the auctioned assets' intended value. This presents a **critical security issue**.

Resolution:
The condition should be updated to use `GT` (greater than) instead of `LT`, ensuring that bids offering less than the required minimum payment are rejected. The issue has been resolved in this PR.

```
if bid.SellingTokenAmount.ToLegacyDec().
        Mul(bidsFloorPrice).
        GT(bid.PaymentTokenAmount.ToLegacyDec()) {
    // Reject bid as it doesn't meet the minimum price
}
```

Severity: Critical
This issue undermines the integrity of the auction and can result in severe financial losses, as it enables malicious users to exploit the system and acquire assets for significantly less than their intended value.

tokenPrice.UpdatedAt Not Updated Leading to Potential Failures

Description:
The `tokenPrice.UpdatedAt` field is used throughout the code to track the last update time of a token's price. However, it is not updated after the initial zero value, causing it to remain static. This behavior can lead to two issues:

1. **Frequent Price Updates in `BeginBlocker`:** Since `tokenPrice.UpdatedAt` is not updated, the `BeginBlocker` function may unnecessarily trigger frequent price updates for tokens, increasing computational load and potentially leading to performance issues.
2. **Bid Failures in `placeBid`:** The `GetTokenPriceForQuoteDenom` function fails if the `tokenPrice.UpdatedAt` exceeds the `PriceExpirationTimeoutSec`, setting flags like `foundBaseTokenStalePrice` or `foundQuoteTokenStalePrice` to `true`. This failure prevents valid bid placements due to stale price data.

Impact:

- Frequent, redundant price updates in `BeginBlocker` could strain system resources and reduce efficiency.
- Valid `placeBid` transactions may fail due to incorrectly flagged stale prices, negatively impacting user experience and the auction process.

Resolution:
The issue has been resolved in this PR, where `tokenPrice.UpdatedAt` is updated appropriately after each price retrieval (`OsmosisClPoolCallback` function).

Severity: High
While not directly exploitable, the issue significantly impacts system efficiency and user functionality, making it critical to ensure proper updates to `tokenPrice.UpdatedAt`.

Incorrect Unit Comparison in Auction Bid Validation

Description:
In the `fcfsBidHandler` function, the `bid.PaymentTokenAmount` (e.g., `ustrd`) was wrongly compared against `auction.sellingAmountAvailable` (e.g., `uatom`). This comparison used mismatched units, leading to invalid logic when validating bids.

Impact:
The incorrect comparison caused legitimate `MsgPlaceBid` transactions to fail, even when they met all other auction requirements. This effectively blocked users from participating in auctions.

Resolution:
The issue has been resolved in a subsequent PR by replacing the incorrect comparison with `bid.SellingTokenAmount.GT(sellingAmountAvailable)`, ensuring that the units match (`uatom` to `uatom`) and the validation logic is correct.

Severity: Medium
This issue disrupts the core functionality of the auction mechanism but does not lead to unauthorized actions or fund loss.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability, etc.) set forth in the associated Services Agreement. This report provided in connection with the Services set forth in the Services Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This audit report is provided on an “as is” basis, with no guarantee of the completeness, accuracy, timeliness or of the results obtained by use of the information provided. Informal has relied upon information and data provided by the client, and is not responsible for any errors or omissions in such information and data or results obtained from the use of that information or conclusions in this report. Informal makes no warranty of any kind, express or implied, regarding the accuracy, adequacy, validity, reliability, availability or completeness of this report. This report should not be considered or utilized as a complete assessment of the overall utility, security or bugfree status of the code.

This audit report contains confidential information and is only intended for use by the client. Reuse or republication of the audit report other than as authorized by the client is prohibited.

This report is not, nor should it be considered, an “endorsement”, “approval” or “disapproval” of any particular project or team. This report is not, nor should it be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts with Informal to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor does it provide any indication of the client’s business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should it be leveraged as investment advice of any sort.

Blockchain technology and cryptographic assets in general and by definition present a high level of ongoing risk. Client is responsible for its own due diligence and continuing security in this regard.

Appendix A: 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:

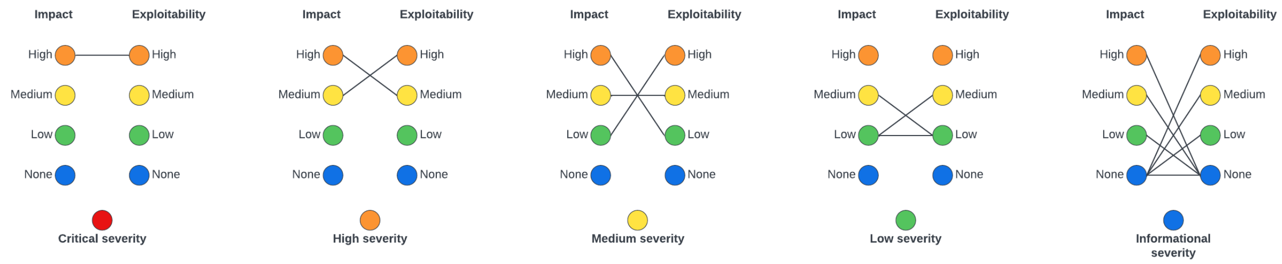
- *Actors* can be any entity that interacts with the system: other blockchains, system users, validators, relayers, but also uncontrollable phenomena (e.g. network delays or partitions).
- *Actions* can be

  - *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**
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