Audit Report

Stride

v1.0

September 26, 2022

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Stride Labs, Inc. to perform a security audit of Stride.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repository:

https://github.com/Stride-Labs/stride

Commit hash: `047714ea7336f6fc95b529f015666bd26d4a5c17`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base’s intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Stride is a blockchain that provides liquidity for staked assets. Using Stride, users can earn both staking and DeFi yields across the Cosmos IBC ecosystem.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Non-deterministic iterations can cause consensus failures	Critical	Resolved
2	`GetHostZoneFromHostDenom` incorrectly uppercases user input, which can be used to mint invalid assets	Critical	Resolved
3	Computationally heavy operations in `BeginBlocker` may slow down or stop block production	Critical	Resolved
4	`RegisterHostZone` does not validate `Bech32Prefix` which will lead to staked funds being unredeemable if misconfigured	Major	Resolved
5	`RegisterHostZone` does not ensure that `HostDenom` and `IbcDenom` are unique which may introduce conflicts when returning a hostzone from these values	Major	Resolved
6	Hard-coded admins increase the potential of unauthorized privileged activity	Major	Resolved
7	`OnTimeoutPacket` method not implemented	Major	Resolved
8	Misconfiguring `CurrentEpoch` and `CurrentEpochStartHeight` during genesis initialization would cause overflow issues	Minor	Resolved
9	Participation rewards configured can be negative	Minor	Resolved
10	Duplicates in `HostZoneList` are not removed during genesis initialization	Minor	Resolved
11	Consider verifying a validator's address when adding new validators	Minor	Acknowledged
12	Host denom and IBC denom validations are insufficient	Minor	Resolved
13	Several unhandled errors exist in the codebase	Minor	Resolved
14	Interchainquery `EndBlocker` uses incorrect telemetry key	Minor	Resolved

15	Epochs genesis validation is missing `EpochCountingStarted` validation	Minor	Resolved
16	`k.BeforeEpochStart` is incorrectly called after the epoch start event is emitted	Minor	Resolved
17	`HandleSend` could potentially exceed the block gas limit	Minor	Resolved
18	Casting `uint64` `msg.Amount` into `uint32` for `Itoa` conversion may yield in an overflow	Minor	Resolved
19	Incorrect code comment	Informational	Resolved
20	`DepositRecord`'s `Amount` is signed integer	Informational	Resolved
21	Hard-coded value may impact log readability	Informational	Resolved
22	Excessive logging and print values	Informational	Resolved
23	Replace hard-coded version in `GetAppVersion`	Informational	Resolved
24	Validations on checking negative values for unsigned integers can be removed	Informational	Resolved
25	Potential division by 0 panic if total delegation is 0	Informational	Resolved
26	Validator name validation can be bypassed with a whitespace character	Informational	Resolved
27	`GetDepositRecordByEpochAndChain` is inefficient	Informational	Acknowledged
28	Duplicated code reduces maintainability	Informational	Resolved
29	Late validation in `RedeemStake` is inefficient	Informational	Resolved
30	Unnecessary allocation in `RedeemStake` is inefficient	Informational	Resolved
31	Log not reported in `ValidateBasic` for `MsgAddValidator`	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	**Medium**	-
Code readability and clarity	**Low-Medium**	There are many TODO comments in the codebase that indicate the code is still under active development.
Level of documentation	**High**	-
Test coverage	**Medium-High**	-

Detailed Findings

1. Non-deterministic iterations can cause consensus failures

Severity: Critical

In several instances of the codebase, iterations are done over maps. Since Go map iterations are non-deterministic, this would cause each validator to produce a different app hash, causing a consensus failure and potentially leading to a chain halt.

Affected code lines:

- `x/stakeibc/keeper/unbonding_records.go:101-109` and `147-152`
- `x/stakeibc/keeper/msg_server_rebalance_validators.go:61-63`
- `app/app.go:762` and `849`
- `cmd/strided/root.go:349`

This issue has independently been detected by the client during the audit, but it is still present in the commit hash that was used for the audit.

Recommendation

We recommend sorting the map keys into a slice and iterating over the sorted keys to ensure deterministic results among all validators.

Status: Resolved

2. `GetHostZoneFromHostDenom` incorrectly uppercases user input, which can be used to mint invalid assets

Severity: Critical

In `x/stakeibc/keeper/host_zone.go:57`, the `GetHostZoneFromHostDenom` keeper function automatically uppercases the denom argument in line `59` and compares it against the zone's denom which is also uppercased in line `61`. As the denom argument is mostly supplied by the user, this would cause an unintended validation bypass.

For example, the user's input `msg.HostDenom` is passed as the `denom` argument to `GetHostZoneFromHostDenom` in `x/stakeibc/keeper/msg_server_liquid_stake.go:22` when the user wants to liquid stake. A user can provide a mixed case argument such as `"aToM"` which will be validated as `"ATOM"` due to the automatic uppercase handling. As a result, the user can mint invalid `stAssets` such as `"staToM"`, which is incorrect and not accepted when redeeming the staked assets via `RedeemStake`.

Recommendation

We recommend removing the functionality of automatically uppercasing denom argument input in `GetHostZoneFromHostDenom` as case-by-case validation should be enough.

Status: Resolved

3. Computationally heavy operations in `BeginBlocker` may slow down or stop block production

Severity: Critical

`BeginBlocker` and `EndBlocker` are a way for module developers to add automatic execution of logic to their module. This is a powerful tool that should be used carefully, as complex automatic functions can slow down or even halt the chain. There are two modules within the scope of this audit where the `BeginBlocker` or `EndBlocker` contains unbounded loops that can slow or even halt the chain.

Both the `interchainquery` and `epochs` modules contain resource intensive `BeginBlocker` or `EndBlocker` functions:

- `x/interchainquery/keeper/abci.go:18`
- `x/epochs/keeper/abci.go:14`

Recommendation

We recommend reworking the `BeginBlocker` and `Endblocker` functions in order to reduce their computational complexity.

Status: Resolved

4. `RegisterHostZone` does not validate `Bech32Prefix` which will lead to staked funds being unredeemable if misconfigured

Severity: Major

The `RegisterHostZone` function in `x/stakeibc/keeper/msg_server_register_host_zone.go:15` is lacking validations to ensure the `msg.Bech32Prefix` is valid and not empty. This functionality is neither performed in `RegisterHostZone` nor the `ValidateBasic` function for `MsgRegisterHostZone`. This is a critical check because if the field is empty `MsgRedeemStake` will error and block all redemptions for that specific zone since `AccAddressFromBech32` will return an error if `hostZone.Bech32Prefix` is empty. There is no way to update this field or a hostzone so any funds that get staked to a hostzone will not be redeemable.

We classify this issue as major instead of critical, since only the admins can cause it. Still, it leads to permanently locked funds, that would require a chain upgrade to be resolved.

This issue has independently been detected by the client during the audit, but it is still present in the commit hash that was used for the audit.

Recommendation

We recommend implementing a basic check in the `ValidateBasic` for `MsgRegisterHostZone` to ensure the `msg.Bech32Prefix` is not empty. Additionally, a config parameter could be created that holds all the valid `Bech32Prefix`es and that can be checked to ensure that only valid prefixes are being used when creating a new hostzone. This parameter could be updated when new staking functionality is added to support additional `stAssets` that may have a different prefix.

Status: Resolved

5. `RegisterHostZone` does not ensure that `HostDenom` and `IbcDenom` are unique which may introduce conflicts when returning a hostzone from these values

Severity: Major

The `RegisterHostZone` function in `x/stakeibc/keeper/msg_server_register_host_zone.go:15` is lacking validations to ensure the hostzone being registered does not contain duplicate fields with existing hostzones. For example, if `msg.HostDenom` is the same as an existing hostzone it will effectively invalidate any guarantee that `GetHostZoneFromHostDenom` provides. `GetHostZoneFromHostDenom` is used in multiple locations to derive the hostzone from a denom passed in so `msg.HostDenom` should be unique throughout all hostzones. This is the same case with `msg.IbcDenom` and `GetHostZoneFromIBCDenom`.

We classify this issue as major instead of critical, since only the admins can cause it. Still, it leads to permanently locked funds, that would require a chain upgrade to be resolved.

Recommendation

We recommend ensuring that both `msg.HostDenom` and `msg.IbcDenom` is unique throughout all host zones before allowing them to be set in `RegisterHostZone`. Additionally, consider implementing the above recommendation during genesis initialization to make it consistent across the codebase.

Status: Resolved

6. Hard-coded admins increase the potential of unauthorized privileged activity

Severity: Major

In `utils/utils.go:17-21`, a hardcoded slice `ADMINS` is defined to represent the addresses that may perform a privileged activity. This is problematic because it creates a situation where it is difficult to control access and respond if one of the admins is compromised. For example, one of the addresses is labeled as a testnet address which likely has private keys that have been shared amongst the development team. If this address were to remain in `ADMINS` it would present a serious risk to Stride as the address could perform multiple privileged actions.

Recommendation

If admins must be used over governance functionality, it is best practice to implement a more robust solution such as creating functionality for the removal/update of `ADMINS` if one were to get compromised. This could be implemented through params that are set at genesis. Alternatively, ldflags could be used in Go to compile the source with only the admins that are required for the destination chain, i. e. testnets and mainnets could have different constant values.

Status: Resolved

The team mentioned that they swapped the admin to a secure account and reduced admin privileges to barebones functionality (most previously gated messages are now governance-controlled).

7. `OnTimeoutPacket` method not implemented

Severity: Major

In `x/stakeibc/module_ibc.go:143`, `OnTimeoutPacket` is not implemented and will panic on any timeout. Operations involving token transfers and burns could potentially lead to a state in which a chain updates the state as based on a successful transfer, but ignores the timeout. This can cause balances that aren't synchronized between chains, which can ultimately lead to users not being able to redeem their tokens successfully, or even being able to redeem more than what they should.

Recommendation

We recommend implementing custom logic that handles different possible packet timeouts in order to ensure that the state is correctly synced cross-chain. As an example, the transfer module implements this method, see here.

Status: Resolved

8. Misconfiguring `CurrentEpoch` and `CurrentEpochStartHeight` during genesis initialization would cause overflow issues

Severity: Minor

In `x/epochs/types/genesis.go:51`, the genesis initialization parameters are validated via the `Validate` function to prevent incorrect configurations. There are no validations that verify the value of `CurrentEpoch` and `CurrentEpochStartHeight` to not be negative values though. If they are configured as negative values, it would cause unintended consequences when converting them into unsigned values using the `uint64` function. For example, the epoch number in `x/stakeibc/keeper/hooks.go:36` would underflow and become a tremendously large value, which is incorrect.

Recommendation

We recommend verifying the values of `CurrentEpoch` and `CurrentEpochStartHeight` to be positive values.

Status: Resolved

9. Participation rewards configured can be negative

Severity: Minor

In `x/mint/types/params.go:177-191`, there is no validation that makes sure the participation rewards decimal value is not a negative value. The participation rewards value is used in `x/mint/keeper/keeper.go:163` when distributing the minted coins via the `DistributeMintedCoin` keeper function. A misconfigured participation rewards value would cause the execution to panic in `x/mint/keeper/keeper.go:135` due to a negative coin amount. As a result, hooks that should be executed after an epoch ends (see `x/mint/keeper/hooks.go:16`) would keep failing.

Recommendation

We recommend checking `v.ParticipationRewards.IsNegative()` in `x/mint/types/params.go:190`.

Status: Resolved

10. Duplicates in `HostZoneList` are not removed during genesis initialization

Severity: Minor

In `x/stakeibc/types/genesis.go:26`, the `stakeibc` genesis validate functionality does not remove duplicates from the `HostZoneList` slice. As the `SetHostZone` keeper function in `x/stakeibc/keeper/host_zone.go:42` uses the `ChainId` as the key identifier, having duplicate chain id values in the `HostZoneList` slice would cause the final index with the same chain id value to be stored in the storage. As a result, previous host zone configurations with duplicate chain id values would be overwritten and ignored completely.

Recommendation

We recommend deduping the `HostZoneList` array similar to how `EpochTrackerList` and `PendingClaimsList` array is validated in `x/stakeibc/types/genesis.go:33-49`.

Status: Resolved

11. Consider verifying a validator's address when adding new validators

Severity: Minor

In `x/stakeibc/keeper/msg_server_add_validator.go:34`, the validator's address supplied is not validated to be a valid address before saving it into the host zone's validator storage. As a result, the keeper function `DelegateOnHost` in `x/stakeibc/keeper/msg_server_submit_tx.go:68` would fail since the for loop would send a transaction to all validators in the hostzone (see line `92`) through `SubmitTx`.

Recommendation

We recommend validating a validator's address before saving it into the host zone.

Status: Acknowledged

The team states that fixing this issue requires issuing an ICQ to the host zone to query the validator set and check the new validator against it. They plan to implement this fix shortly after launch.

12. Host denom and IBC denom validations are insufficient

Severity: Minor

In `x/stakeibc/types/message_register_host_zone.go:57`, `68`, and `71`, the `ValidateBasic` function performs several validation checks towards the host denom and

IBC denom input. The validations are not very strict though. They currently only validate that the host denom is not an empty string and the IBC denom starts with the `"ibc"` prefix. As such, there is a possibility that the denoms are still invalid after the validation, for example a denom may contain whitespace.

Recommendation

We recommend using ValidateDenom() and ValidateIBCDenom() to verify the host denom and IBC denom to be valid respectively.

Status: Resolved

13. Several unhandled errors exist in the codebase

Severity: Minor

In `x/stakeibc/module_ibc.go:97-100`, the addresses of the interchain accounts are formatted, but the error is never handled.

In `x/records/keeper/deposit_record.go:93`, the iterator can be checked to see if there is an error by calling `Error`.

In `x/stakeibc/keeper/host_zone.go:157`, the iterator can be checked to see if there is an error by calling `Error`. Additionally, the function that is passed as a second argument can fail and its error is not reported.

In `x/records/module_ibc.go`, there are several functions that do not check errors. If there are errors during the execution of the functions, these errors will not be caught.

Recommendation

We recommend handling the errors that are potentially returned in `x/stakeibc/module_ibc.go:97-100`.

In `x/records/keeper/deposit_record.go:93`, we recommend returning an error signaling the case in which the iterator failed.

In `x/stakeibc/keeper/host_zone.go:157`, return the index of elements that succeeded in applying the function, and the first error (if any) that was returned either from the function itself or from the iterator.

We recommend checking and handling the returned error values in `x/records/module_ibc.go:58`, `118`, `129`, `140`, `151`, and `233`.

Status: Resolved

14. Interchainquery `EndBlocker` uses incorrect telemetry key

Severity: Minor

In `x/interchainquery/keeper/abci.go:19`, the `EndBlocker` function is incorrectly using the `MetricKeyBeginBlocker` telemetry key, which is intended to be used in `BeginBlocker`.

Recommendation

We recommend using the `MetricKeyEndBlocker` telemetry key instead.

Status: Resolved

15. Epochs genesis validation is missing `EpochCountingStarted` validation

Severity: Minor

In `x/epochs/types/genesis.go:51`, the `Validate` function allows initial epochs with a `true` `EpochCountingStarted` variable to pass validation, which will cause these epochs to never start.

Recommendation

We recommend enforcing `EpochCountingStarted` to be `false` for all initial state epochs.

Status: Resolved

16. `k.BeforeEpochStart` is incorrectly called after the epoch start event is emitted

Severity: Minor

In `x/epochs/keeper/abci.go` in the `BeginBlocker` function, the `k.BeforeEpochStart` hook is currently being called after the `EventTypeEpochStart` event has been emitted, which can cause unexpected side effects.

Recommendation

We recommend calling `k.BeforeEpochStart` before emitting the `EventTypeEpochStart` event.

Status: Resolved

17. `HandleSend` could potentially exceed the block gas limit

Severity: Minor

With a large enough number of `userRedemptionRecords`, the `HandleSend` function in `x/stakeibc/keeper/ibc_handlers.go`, which iterates over `userRedemptionRecords`, could potentially exceed the block gas limit.

Recommendation

We recommend adding a `userRedemptionRecords` count limit that will not exceed the block gas limit.

Status: Resolved

18. Casting `uint64` `msg.Amount` into `uint32` for `Itoa` conversion may yield in an overflow

Severity: Minor

In `x/stakeibc/keeper/msg_server_liquid_stake.go`, with a large enough number of `msg.Amount`, the conversion `int(msg.Amount)` could lead to an overflow in case the platform uses `int32` as `int`. In that case, an overflow could happen if a user is requesting liquid staking greater than `u32::MAX`, leading to a small amount sent to the module account. We consider the impact to be minor because the balance is properly tracked and the issue can be reverted.

Ref: https://go.dev/ref/spec#Numeric_types

Recommendation

We recommend checking in the `ValidateBasic` method that the amount lies between `0` and `u32::MAX`.

Status: Resolved

19. Incorrect code comment

Severity: Informational

The `ValidateBasic` function for the `MsgRegisterHostZone` message in `x/stakeibc/types/message_register_host_zone.go:74` contains a misleading comment that may impact the readability and maintainability of the codebase. Currently, the comment states that the `msg.TransferChannelId` cannot be empty and must begin with the string `"transfer"`. The check below performs a validation to ensure that the `msg.TransferChannelId` is prefixed with the string `"channel"`.

Recommendation

We recommend updating the comment to reflect the code's functionality by changing `"transfer"` to `"channel"`.

Status: Resolved

20. `DepositRecord`'s `Amount` is signed integer

Severity: Informational

In `proto/records/genesis.proto:54`, `Amount` has type `int64`. Given that amount should always be a positive number, there seems to be no reason to use a signed type.

Recommendation

We recommend changing the `Amount` in `DepositRecord` to `uint64`.

Status: Resolved

21. Hard-coded value may impact log readability

Severity: Informational

In `x/stakeibc/keeper/ibc_handlers.go:407`, there is a reference to a hard-coded denom `stuatom`. This will impact the log readability during the unbonding of denoms that are not `stuatom`.

Recommendation

We recommend removing the hard-coded `stuatom` denom to avoid confusing log values when Stride is dealing with other `stAssets`.

Status: Resolved

22. Excessive logging and print values

Severity: Informational

`GetEpochInfo` in `x/epochs/keeper/epoch.go:13` contains excessive debugging logs with unrelated print statements.

Recommendation

We recommend removing these debugging logs before Stride launches to production.

Status: Resolved

23. Replace hard-coded version in `GetAppVersion`

Severity: Informational

`GetAppVersion` in `x/records/module_ibc.go:261` returns a hard-coded value for the version.

Recommendation

We recommend modifying the implementation to dynamically retrieve the app version instead of a hard-coded value.

Status: Resolved

24. Validations on checking negative values for unsigned integers can be removed

Severity: Informational

In `x/stakeibc/types/message_add_validator.go:67` and `x/stakeibc/types/message_change_validator_weight.go:52`, the if statement attempts to check whether the `msg.Weight` values are a negative number. As unsigned numbers don't have negative values (they will overflow if they do), these checks are redundant and can be removed.

Recommendation

We recommend removing both checks as mentioned above.

Status: Resolved

25. Potential division by 0 panic if total delegation is 0

Severity: Informational

In `x/stakeibc/keeper/msg_server_rebalance_validators.go:74`, there's a chance that the total delegation value returned might be 0. If that happens, the execution would panic in the next line due to a division by 0 attempt. This is not a security concern since a division by zero panic is equivalent to an error, but it is best practice to return errors instead of relying on panics.

Recommendation

We recommend returning an error if the total delegation value is 0.

Status: Resolved

26. Validator name validation can be bypassed with a whitespace character

Severity: Informational

In `x/stakeibc/types/message_add_validator.go:56`, the name validation verifies the provided `msg.Name` argument is valid by checking the length of it to be non-zero. This validation can be bypassed by providing a whitespace character which causes the length to be 1 while causing the validator name to be invalid.

Recommendation

We recommend trimming whitespace input before performing the validation to prevent invalid whitespace input.

Status: Resolved

27. `GetDepositRecordByEpochAndChain` is inefficient

Severity: Informational

The function in `x/stakeibc/keeper/deposit_record.go:109` retrieves all the records first to then perform an action on each of them until a certain condition is met.

Recommendation

We recommend passing a function that can be applied to each record in order to short-circuit the execution when the condition is met.

Status: Acknowledged

The team mentioned they will make this change shortly after launch in a software upgrade. They mentioned that it's not very risky now because the number of deposit records is capped at 3N where N is the number of host zones. It is planned for 3N to be 45 in 2022 (no more than 15 host zones this year).

28. Duplicated code reduces maintainability

Severity: Informational

In `x/stakeibc/keeper/ibc_handlers.go:94`, the logic is duplicated for a case variant. This reduces the maintainability of the codebase. More importantly, there is a risk for not updating both parts consistently in case of upgrades of these code blocks.

Recommendation

We recommend deduplicating the logic of `case "/cosmos.staking.v1beta1.MsgDelegate"` with the one in line `118`.

Status: Resolved

29. Late validation in `RedeemStake` is inefficient

Severity: Informational

In `x/stakeibc/keeper/msg_server_redeem_stake.go:81`, there is a check to confirm whether the user has or has not performed a redemption in an epoch. However, this check could happen much earlier, in line `25`, to short-circuit the execution and make the code more efficient.

Recommendation

We recommend moving the validation right after line `25` to reduce the computation on redemptions that won't succeed in the current epoch.

Status: Resolved

30. Unnecessary allocation in `RedeemStake` is inefficient

Severity: Informational

In `x/stakeibc/keeper/msg_server_redeem_stake.go:118`, if `hostZoneUnbondings` is an empty map, the code will still allocate a new one, which is inefficient.

Recommendation

We recommend changing the if statement from `len(hostUnboundings) == 0` to `hostUnboundings == nil`.

Status: Resolved

31. Log not reported in `ValidateBasic` for `MsgAddValidator`

Severity: Informational

In `x/stakeibc/types/message_add_validator.go:64`, the error message yields a string which is not used.

Recommendation

We recommend calling `log.{Debug, Println, ...}` on that string and reporting the error message.

Status: Resolved