Audit Report

Cosmos Interchain Security

v1.0

June 23, 2023

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 Simply VC Limited to perform a security audit of Cosmos Interchain Security.

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 target:

Repository	https://github.com/cosmos/interchain-security
Commit	`6a856d183cd6fc6f24e856e0080989ab53752102`
Scope	The whole repository was in scope.

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

Cosmos Interchain Security allows a provider chain (like the Cosmos Hub) to provide security to a consumer chain by producing blocks for it.

It achieves this by sharing the set of validators who are in charge of producing blocks. The participating validators would run two nodes, one for the provider chain and one for the consumer chain, and receive fees and rewards on both chains.

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.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Medium-High**	The design of the ICS protocol and its subprotocols utilize multiple complex components. They depend on IBC, several Cosmos SDK modules as well as provider-consumer chain communication.
Code readability and clarity	**Medium-High**	-
Level of documentation	**High**	Documentation is extensive and highly detailed. Some sections are outdated.
Test coverage	**Medium-High**	65.7% test coverage.

Summary of Findings

No	Description	Severity	Status
1	A validator updating its `ConsumerKey` to the same key causes a panic in the related consumer chain	Critical	Resolved
2	Attackers can DOS consumer chains by sending multiple coins to the provider chain's reward address	Critical	Resolved
3	Validators can slow down the provider chain by submitting multiple `AssignConsumerKey` messages in the same block	Major	Resolved
4	The provider chain halts on failure to send packets to a single consumer chain	Major	Resolved
5	Potential loss of rewards during consumer chain removal	Minor	Acknowledged
6	Consumer chains can DOS the provider chain by sending slash packets	Minor	Acknowledged
7	Validators can evade slashing if an equivocation proposal is submitted seven days or later after the infraction	Minor	Acknowledged
8	Consumer chains can expand provider chain unbonding period	Minor	Acknowledged
9	Unbounded loop over proposals in `BeginBlock` could slow down block production	Minor	Acknowledged
10	Unbounded loops over consumer chains in `EndBlock` could slow down block production	Minor	Acknowledged
11	`LastTransmissionBlockHeight` is wrongly updated if the IBC token transfer fails	Minor	Acknowledged
12	Inefficient removal of executed proposals	Informational	Acknowledged
13	Emitting incorrect events	Informational	Acknowledged
14	Redundant checks are inefficient	Informational	Acknowledged
15	Outstanding TODOs	Informational	Acknowledged
16	Miscellaneous comments	Informational	Acknowledged

Detailed Findings

1. A validator updating its `ConsumerKey` to the same key causes a panic in the related consumer chain

Severity: Critical

In `x/ccv/provider/keeper/key_assignment.go:505-523`, during the processing of `ConsumerKey` assignments, the execution appends two `abci.ValidatorUpdate` elements to the `newUpdates` slice for each consumer key to replace.

The first `abci.ValidatorUpdate` is constructed with the old key and `Power` equal to zero and the second one contains the new key with the validator's current `Power`.

If a validator updates its key to the same one, the `newUpdates` slice will contain two elements with the same key and different `Power`, respectively zero and the current one.

This would cause the related consumer chain to panic when trying to update the validator set due to the duplicated `ConsumerKey` in the validator set.

Recommendation

We recommend enforcing validators to update their `ConsumerKey` to a different one.

Status: Resolved

2. Attackers can DOS attack consumer chains by sending multiple coins to the provider chain's reward address

Severity: Critical

During the execution of the consumer chain's `EndBlocker`, the `SendRewardsToProvider` function, defined in `x/ccv/consumer/keeper/distribution.go:103`, gets the balance of all the coins in the `tstProviderAddr` and sends them to the provider chain.

In order to do so, it has to iterate through all the coins found in the reward address and send them one by one through IBC to the provider chain.

Since anyone is allowed to send coins to the reward address, an attacker could create and send a large number of coins with different denoms to it, for example using a chain with the token factory module, in order to attack the mentioned unbounded iteration and DOS attack the chain.

Consequently, the execution of the `EndBlocker` will take more time and resources than expected causing the consumer chain to slow down or in the worst case even halt.

Recommendation

We recommend implementing an allowlist for reward coins in the consumer chain to limit the number of iterations performed in the `EndBlocker`.

Status: Resolved

3. Validators can slow down the provider chain by submitting multiple `AssignConsumerKey` messages in the same block

Severity: Major

The `AssignConsumerKey` function, defined in `x/ccv/provider/keeper/key_assignment.go:378`, enables validators to assign themselves a different `consumerKey` for approved consumer chains.

In order to perform this action the `consumerAddrsToPrune` `AddressList` grows by one element in line `428`.

Since this `AddressList` is iterated over in the `EndBlocker` in `x/ccv/provider/keeper/relay.go:95`, it could be utilized by attackers to slow down the provider chain.

To execute such an attack, malicious actors could craft transactions with multiple `AssignConsumerKey` messages and spam the provider chain with those transactions. The `consumerAddrsToPrune` `AddressList` will grow of the same cardinality as the `AssignConsumerKey` sent messages.

Consequently, the execution of the `EndBlocker` will take more time and resources than expected causing the provider chain to slow down or in the worst case even halt.

We classify this issue as major instead of critical since the number of iterations is bounded by the maximum number of messages in a block.

Recommendation

We recommend allowing validators to execute `AssignConsumerKey` only once in a block, and only for approved consumer chains.

Status: Resolved

The Informal Systems team has mitigated this issue by limiting the number of `VSCMaturedPacket` packets processed in a block.

4. The provider chain halts on failure to send packets to a single consumer chain

Severity: Major

During the execution of the provider chain's `EndBlock` function, the `SendVSCPacketsToChain` function, defined in `x/ccv/provider/keeper/relay.go:182`, panics if it fails to send a packet to a consumer chain.

This implies that an issue relevant only to a consumer chain will make the provider chain panic. Likewise, consumers will also panic in `x/ccv/consumer/keeper/relay.go:180` because they will not be able to send packets to the provider.

This evidences a single point of failure – an error occurring on a single packet for a single consumer can halt the entire ICS network.

We classify this issue as major instead of critical because in the current version consumer chains, relayers, and light clients are assumed to be non-malicious.

Recommendation

We recommend isolating failures of every consumer-provider relation, rather than panicking in the provider chain upon packet sending errors to consumer chains.

Status: Resolved

5. Potential loss of rewards during consumer chain removal

Severity: Minor

In `x/ccv/consumer/keeper/distribution.go`, the reward distribution process takes place at the end of each block on the consumer chain. When the number of blocks for transmission is exceeded, the accumulated rewards are sent to the provider.

If a proposal to remove or stop a consumer chain is successfully executed, the code at `x/ccv/provider/keeper/proposal.go:155-232` automatically handles the removal process. This includes tasks such as cleaning up the state, closing the channel, releasing unbonding operations, and deleting all related data.

However, it does not explicitly check whether the rewards associated with the consumer chain have been distributed before removing the chain. This may lead to a loss of rewards that have not been sent to the provider.

Recommendation

We recommend ensuring that all rewards have been distributed during the consumer chain removal process.

Status: Acknowledged

6. Consumer chains can DOS the provider chain by sending slash packets

Severity: Minor

In `x/ccv/provider/keeper/throttle.go:274-276`, a panic occurs if the cardinality of the queue of throttled packets is bigger than the defined `MaxThrottledPackets` hard cap.

This behavior can be exploited by a malicious consumer chain, making the provider unable to process any further packets from other consumers.

To execute such an attack, the malicious consumer chain can spam the provider chain with slash packets in order to fill the queue and cause a panic. Since the validation of slash packets in `x/ccv/provider/keeper/relay.go` does not disregard duplicate or other invalid slash packets, the attack can be performed for example by simply sending a valid slash packet multiple times.

This could cause other consumer chains to be removed from the ICS because the provider will not receive relevant maturity notifications before the timeout.

We classify this issue as minor instead of major because in the current version consumer chains, relayers, and light clients are assumed to be non-malicious.

Recommendation

We recommend implementing a coordinated queuing mechanism between provider and consumer chains able to offload excessive packets to the consumer chains and disregard invalid slash packets before enqueuing them.

This could be implemented with two FIFO queues: a bigger one on the consumer side and a smaller one on the provider side. The provider can then throttle messages according to its queue's capacity and exceeding messages are kept in the consumer chain queue. If the consumer chain keeps sending slash packets, the provider could remove the consumer.

Status: Acknowledged

The client states that this issue will be addressed in future versions with the transition to the untrusted consumer chain paradigm. Currently, the issue is mitigated by the requirement of governance approval for consumer chains to join ICS.

7. Validators can evade slashing if an equivocation proposal is submitted seven days or later after the infraction

Severity: Minor

When a validator performs a double-signing infraction on a consumer chain, equivocation slashing should be proposed and voted upon. Since the voting period lasts two weeks and the unbonding period is currently set to three weeks, an equivocation slashing proposal submitted seven days or later after the infraction takes place allows the validator to unbond and evade slashing.

Recommendation

We recommend preventing and reverting the unbonding process for validators that are targeted by an equivocation slashing proposal during its voting period.

Status: Acknowledged

8. Consumer chains can expand provider chain unbonding period

Severity: Minor

The default unbonding period for consumer chains `DefaultConsumerUnbondingPeriod` is set in `x/ccv/consumer/types/params.go:37` to one day less than the default unbonding period, which is currently three weeks.

Since a validator's unbonding matures only after all consumer chains' unbondings mature, the `DefaultConsumerUnbondingPeriod` is chosen to ensure that the validators can unbond without any delays.

However, since the consumer unbonding period is not enforced in the code to be less than the provider's, consumer chains configured with a bigger unbonding period will delay the provider's unbonding period.

Recommendation

We recommend enforcing an upper bound on the consumer unbonding period to avoid delays in validator unbonding on the provider chain.

Status: Acknowledged

9. Unbounded loop over proposals in `BeginBlock` could slow down block production

Severity: Minor

In `x/ccv/provider/keeper/proposal.go:370` and `504`, an unbounded loop is used to iterate over the `ConsumerAdditionProposals` and `ConsumerRemovalProcess` list. This loop has no set limit for the number of times it can run. Since there are no restrictions on the number of consumer chains that can be supported by the provider chain, a large number of proposals could slow down or halt the chain.

We are reporting this with Minor severity since proposals go through governance voting and the likelihood of having multiple proposals with the same `SpawnTime` or `StopTime` is low.

Recommendation

We recommend implementing a queue in order to be able to consume proposals in batches if needed.

Status: Acknowledged

The client states that the issue is mitigated by the requirement of governance approval for consumer chains to join the ICS.

10. Unbounded loops over consumer chains in `EndBlock` could slow down block production

Severity: Minor

When the `EndBlock` function is executed on the provider chain, there are several instances where the `GetAllConsumerChains` function is called to retrieve a list of all consumer chains. As there are no restrictions on the number of consumer chains that a provider can support, this slice can potentially be very large.

The `GetAllConsumerChains` function is called in:

- `x/ccv/provider/keeper/proposal.go:60`,
- `x/ccv/provider/keeper/proposal.go:173`, and
- `x/ccv/provider/keeper/proposal.go:224`.

These unbounded lists are then iterated over to perform various operations for all active consumer chains. Specifically, there are iterations over `leadingVSCMaturedData`, `pendingPackets`, and `MustApplyKeyAssignmentToValUpdates`.

If the cardinality of these lists increases, block production may slow down, possibly even halting the chain.

Operations on time-critical applications running on the network such as auctions or governance proposal execution may be delayed when a consumer chain floods the `EndBlocker` with fraudulent VSC matured packets, leading to a delay in block production.

We are reporting this issue as Minor since consumer chains can only be added through governance.

Recommendation

We recommend benchmarking the impact of a growing number of consumer chains on block production time and setting an upper boundary on the number of consumer chains a provider can support.

Status: Acknowledged

The client states that this issue is mitigated by the small number of consumer chains that will be supported in this version.

11. `LastTransmissionBlockHeight` is wrongly updated if the IBC token transfer fails

Severity: Minor

During the execution of the `EndBlockRD` function, defined in `x/ccv/consumer/keeper/distribution.go:21`, the `LastTransmissionBlockHeight` is updated with the current timestamp even if the IBC token transfer fails.

This implies that rewards are not re-sent in the next block but in the next epoch, leading to lower rewards for unstaking validators. Also, a misleading value is stored.

Recommendation

We recommend not updating the `LastTransmissionBlockHeight` timestamp if the IBC token transfer fails.

Status: Acknowledged

12. Inefficient removal of executed proposals

Severity: Informational

The `BeginBlock` function, which is called at the beginning of every block, calls `BeginBlockInit` in `x/ccv/provider/keeper/proposal.go:367` to get the pending consumer addition proposals and then deletes executed proposals in line `391`.

However, even if there is no pending proposal to execute, the `DeletePendingAdditionProps` function is called to fetch a `KVStore` from the `MultiStore` every block, which is inefficient.

This inefficiency can be removed by changing the function to delete a single executed proposal and moving it inside the for loop.

The same issue applies to the `DeletePendingRemovalProps` function called from `BeginBlockCCR` in `x/ccv/provider/keeper/proposal.go:501`.

Recommendation

We recommend changing the `DeletePendingAdditionProps` and `DeletePendingRemovalProps` to delete a single executed proposal and moving them inside the for loops.

Status: Acknowledged

13. Emitting incorrect events is misleading

Severity: Informational

In case of unmarshalling packet data failure in the `OnRecvPacket` function, the `ack` value will contain an error acknowledgment which is used to emit an event with `AttributeKeyAckSuccess` in both `x/ccv/consumer/ibc_module.go:237` and `x/ccv/provider/ibc_module.go:205`. This is misleading for client applications and users, since a success flag can be returned along with an error.

Similarly, the `DistributeRewardsInternally` function in `x/ccv/consumer/keeper/distribution.go` distributes rewards from the `feeCollector` to `ConsumerRedistributeName` and `ConsumerToSendToProviderName` according to the `DefaultConsumerRedistributeFrac`. By the time the `sendRewardsToProvider` function is called, `feeCollector` has a zero balance. Thus, the emitted event in `x/ccv/consumer/keeper/distribution.go:138` will always contain 0 emitted `fpTokens`, which again is misleading for client applications and users

Recommendation

We recommend emitting the correct event in case of unmarshalling packet data failure and the actual distributed `fpTokens` value calculated in `DistributeRewardsInternally`.

Status: Acknowledged

14. Redundant checks are inefficient

Severity: Informational

The codebase contains redundant checks:

- The version check in `x/ccv/consumer/ibc_module.go:95` is unnecessary as it is already invoked in line `42`.
- The `ValidateBasic` function in `x/ccv/provider/client/cli/tx.go:56` is unnecessary as it is already invoked in the `GenerateOrBroadcastTxWithFactory` function in line `60`.

Recommendation

We recommend removing the aforementioned redundant checks.

Status: Acknowledged

15. Outstanding TODOs

Severity: Informational

There are multiple TODOs in the codebase that imply that the codebase is still in development:

- `x/ccv/types/expected_keepers.go:82`,
- `x/ccv/provider/ibc_module.go:185`, and
- `x/ccv/consumer/keeper/genesis.go:21`.

Recommendation

We recommend resolving all TODOs in the codebase before the production release.

Status: Acknowledged

16. Miscellaneous comments

Severity: Informational

Across the codebase, instances of unused/commented code and inaccurate comments have been found. This can negatively impact the maintainability of the codebase.

Recommendation

The following are some recommendations to improve the overall code quality and readability:

- Remove unused errors in `x/ccv/provider/types/errors.go`.
- Remove unused errors in `x/ccv/types/errors.go`.

- Remove unused events in `x/ccv/types/events.go`.
- Remove the unused function `ValidateString` in `x/ccv/types/shared_params.go`.
- Remove the commented code in `x/ccv/consumer/module.go:44`.
- Remove the commented code in `x/ccv/consumer/module.go:49`.
- Remove the commented code in the `prepForZeroHeightGenesis` function in `app/consumer-democracy/export.go:55-70`, `77-99`, `105-121`, and `128-172`.
- Change the error message in `x/ccv/provider/types/proposal.go:186` from “spawn time cannot be zero” to “stop time cannot be zero”.
- Change the error message in `x/ccv/consumer/types/genesis.go:165` from “cannot have 0 maturity time” to “cannot have 0 id”.

Status: Acknowledged