Audit Report

Slide SDK

v1.1

September 20, 2024

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security GmbH

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

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Gaia Labs LTD to perform a security audit of Slide SDK.

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/LandslideNetwork/slide-sdk
Commit	`247e4cae90379015a7ee1f742505530406a314b7`
Scope	All packages were in scope.
Fixes verified at commit	`bfc067c8aeeed3e2c6a1331732d29e00804790bc` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

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

Slide SDK defines a custom virtual machine designed for AvalancheGo. This custom VM enables the execution of Cosmos SDK chains on the Avalanche network by emulating the CometBFT consensus mechanism. Through this emulation, the VM allows the execution of Cosmos SDK chains and the interaction with Cosmos modules while being secured by the Avalanche consensus protocol.

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 VM facilitates complex communications with AvalancheGo and implements emulation of ComeBFT functionalities.
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium**	The documentation describes the project folder structure and includes high-level diagram flows.
Test coverage	**Medium**	`go test` reports an average test coverage of `36.9%`. End-to-end tests for ABCI and CosmWasm are implemented.

Summary of Findings

No	Description	Severity	Status
1	Block production halts if the mempool size exceeds `100MB`	Critical	Resolved
2	Casting `uint` to `int` can lead the VM to panic	Critical	Resolved
3	Incorrect cast from `int64` to `uint64` makes the `txSelector` unable to prune transactions in the `PrepareProposal`	Major	Resolved
4	Unlimited transaction retrieval from mempool can lead to DoS	Major	Resolved
5	Overly permissive gRPC default options	Minor	Resolved
6	Incorrect boolean value assigned to `allowShutdown` during VM initialization	Minor	Resolved
7	Incomplete validation in `ValidateBlock`	Minor	Resolved
8	Lack of subscription limits in `BroadcastTxCommit` can lead to DoS	Minor	Resolved
9	Lack of request channel handling in `BroadcastTxSync`	Minor	Resolved
10	Lack of validation for `blockMeta` can lead to `nil` elements in `blockMetas`	Minor	Resolved
11	Partial support for Cosmos modules relying on validator information	Minor	Acknowledged
12	gRPC methods for AvalancheGo should not be exposed	Informational	Resolved
13	Missing type assertion and error handling for messages from `deliverTxSub.Out()`	Informational	Resolved
14	Unnecessary panic in `validatePage` function	Informational	Resolved
15	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Block production halts if the mempool size exceeds `100MB`

Severity: Critical

In the `CreateProposalBlock` method of the `BlockExecutor`, defined in `vm/types/state/executor.go:99-149`, transactions from the mempool are processed by the `PrepareProposal` method and subsequently validated by the `Validate` function to ensure their combined size is smaller than `MaxBlockSizeBytes`, set at `100MB`. This security check is intended to prevent the constructed transaction list from exceeding this size, as it should have been enforced by the `PrepareProposal`.

However, due to the “Incorrect cast from into64 to uint64 makes the txSelector unable to prune transactions in the PrepareProposal” and “Unlimited transaction retrieval from mempool can lead to DoS” issues, this maximum size is not enforced and the proposed block contains all the transactions in the mempool.

Consequently, attackers can exploit this vulnerability by sending more than `100MB` of transactions to the mempool, causing the `Validate` function to return an error. This error is propagated to the `BuildBlock` method which returns an error to AvalancheGo leading to the impossibility for the VM to create new blocks.

Recommendation

We recommend limiting the amount of transactions included in the proposed block.

Status: Resolved

2. Casting `uint` to `int` can lead the VM to panic

Severity: Critical

In `vm/rpc.go:329`, during the execution of the `GenesisChunked RPC` method, the `chunk` argument is defined as `uint` and it is then cast to an `int`, which can result in a negative number due to overflow if the `chunk` value is large.

Consequently, this causes the subsequent conditional statement to evaluate to `false`, resulting in `ResultGenesisChunk` being returned with a negative `ChunkNumber` which is then used to access the `rpc.vm.genChunks` array.

Since array indices must be non-negative, attempting to access the array with a negative index causes a panic, halting the block production of the VM.

Attackers can leverage this behavior by sending arbitrary `chunk` values to the `RPC` endpoint to stop the VM.

Recommendation

We recommend adding additional validation or changing the type system to prevent the overflow and ensure the `chunk` argument remains non-negative.

Additionally, recovery should be implemented to handle panics in the `RPC` endpoints.

Status: Resolved

3. Incorrect cast from `int64` to `uint64` makes the `txSelector` unable to prune transactions in the `PrepareProposal`

Severity: Major

In `vm/types/state/executor.go:121`, the `CreateProposalBlock` method of the `BlockExecutor` sets `maxDataBytes` to `-1`, which is invalid as it must be a positive value since only `maxBytes` can accept a negative value. This results in constructing the `RequestPrepareProposal` struct with `MaxTxBytes` set to `-1`.

Then the `PrepareProposalHandler` of the Cosmos SDK baseapp is executed, `MaxTxBytes`, which is represented as `int64`, is cast to `uint64` and then passed to the `SelectTxForProposal` method of the `txSelector` to prune transactions exceeding the maximum bytes and gas limits.

However, casting `-1` from `int64` to `uint64` results in `MaxTxBytes` being `18446744073709551615` bytes, which is approximately `18446744TB`.

Consequently, an exceedingly large transaction limit could lead to processing an enormous transaction list, allowing attackers to perform denial-of-service (DoS) attacks by spamming a large number of transactions.

Recommendation

We recommend ensuring `maxDataBytes` is set to a valid, positive value to prevent improper casting and potential overloading of the transaction list.

Status: Resolved

4. Unlimited transaction retrieval from mempool can lead to DoS

Severity: Major

In `vm/types/state/executor.go:99` the `CreateProposalBlock` method of the `BlockExecutor` retrieves transactions from the mempool by invoking the `ReapMaxBytesMaxGas` function, passing `-1` for both `maxBytes` and `maxGas` parameters.

However, this effectively instructs the `CListMempool` implementation to ignore any size and gas limits when fetching transactions, resulting in all transactions in the mempool being handled in a single batch.

This behavior can cause denial-of-service (DoS) attacks and timeouts, as the node may become overloaded by attempting to process all transactions in the mempool within a single block.

Recommendation

We recommend setting appropriate limits for `maxBytes` and `maxGas` parameters when calling the `ReapMaxBytesMaxGas` method.

Status: Resolved

5. Overly permissive gRPC default options

Severity: Minor

The `Serve` function, defined In `landslidevm.go:73-160`, is responsible for starting the gRPC server and it accepts various `grpc.ServerOption` arguments.

When not specified, these arguments default to `DefaultServerOptions`, defined in `landslidevm.go:51-65`.

However, this default configuration is excessively permissive and could lead to potential denial-of-service (DoS) attacks.

Specifically, setting `MaxRecvMsgSize` to `math.MaxInt` changes the default maximum receive message size from `4MB` to `9223372TB`.

Similarly, `MaxConcurrentStreams` is set to `math.MaxUint32`, allowing an excessively large number of concurrent streams.

We are reporting this issue with minor severity since the aforementioned server is expected to be only exposed locally to interact with the AvalancheGo node.

Recommendation

We recommend limiting the `MaxRecvMsgSize` and `MaxConcurrentStreams` to prudential values.

Status: Resolved

6. Incorrect boolean value assigned to `allowShutdown` during VM initialization

Severity: Minor

When initializing the VM, the `New` function is called, which subsequently invokes the `NewViaDB` function to configure some VM state parameters. One of these parameters is `allowShutdown`, which stores information about whether a node is ready to shut down via the `Shutdown` operation.

However, the default assigned value is `True`, while the logic does not allow the parameter value to be changed to `False` anywhere in the code.

Consequently, when calling the `Serve` operation, the check performed in `landslidevm.go:110` via the `CanShutdown` function will always return `True`, which will result in incorrect execution of the syscall management logic.

Recommendation

We recommend setting the default value of the `allowShutdown` variable to `False`. Changing the value to `True` should only happen when calling the `Shutdown` operation.

Status: Resolved

7. Incomplete validation in `ValidateBlock`

Severity: Minor

In `vm/types/state/utils.go:65`, the `ValidateBlock` function appears to be incomplete, performing fewer validations compared to its counterpart in CometBFT.

While it is understandable that certain checks, such as those related to validators and evidence, are not applicable to this virtual machine, other checks, such as those for block height, state hashes, and block time, should be included.

Recommendation

We recommend extending the `ValidateBlock` function to incorporate additional checks for block height, state hashes, and block time.

Status: Resolved

8. Lack of subscription limits in `BroadcastTxCommit` can lead to DoS

Severity: Minor

In the `BroadcastTxCommit` method of the `RPC`, defined in `vm/rpc.go:141`, there are no enforced limits for `MaxSubscriptionClients` and `MaxSubscriptionsPerClient`, unlike the implementation in CometBFT.

As a result, attackers could exploit this by spamming a large number of `BroadcastTxCommit` requests, potentially causing a denial-of-service (DoS) attack.

Recommendation

We recommend implementing maximum limits for `MaxSubscriptionClients` and `MaxSubscriptionsPerClient` in the `BroadcastTxCommit` method.

Status: Resolved

9. Lack of request channel handling in `BroadcastTxSync`

Severity: Minor

In `vm/rpc.go:231-247` the `BroadcastTxSync` method of the `RPC` does not utilize `rpctypes.Context` to manage scenarios where a request is canceled or timed out.

Consequently, the method does not properly handle cases where the client cancels the request or the request times out as implemented in CometBFT.

Recommendation

We recommend modifying the `BroadcastTxAsync` method to properly handle request cancellations and timeouts by catching `<-ctx.Context().Done()`.

Status: Resolved

10. Lack of validation for `blockMeta` can lead to `nil` elements in `blockMetas`

Severity: Minor

In the `BlockchainInfo` function, defined in `vm/rpc.go:302`, the result of `rpc.vm.blockStore.LoadBlockMeta(height)` is directly appended to the `blockMetas` slice without any validation.

Consequently, if `LoadBlockMeta` returns `nil`, for example in `cometbft@v0.38.6/store/store.go:20`, this could lead to `blockMetas` containing `nil` elements.

Recommendation

We recommend implementing a validation check that `blockMeta` is not `nil` before appending it to the `blockMetas` slice.

Status: Resolved

11. Partial support for Cosmos modules relying on validator information

Severity: Minor

Several Cosmos SDK modules, such as `distribution`, `slashing`, and `staking`, rely on information about validators, including the block proposer and voters.

However, since this part is emulated, this VM does not handle validators and the block voting procedure. For example, the proposer for all blocks is hardcoded to the zero address.

As a result, the business logic in these modules could fail to operate correctly since they process mocked data instead of actual validator information. For example, validators can be registered and staked, and tokens will be distributed in each block, but the underlying operations will not reflect true validator activities.

Recommendation

We recommend documenting this behavior and evaluating the fork of affected Cosmos modules to handle some processes differently.

Status: Acknowledged

The client states that the VM does not have a real Tendermint validator's signature and that this should always be kept in mind when running a new Cosmos chain on the VM.

The client has also agreed to include a section in the documentation to clarify this point.

12. gRPC methods for AvalancheGo should not be exposed

Severity: Informational

The VM provides multiple gRPC endpoints which are required for the communication with the AvalancheGo node. Those endpoints are linked with critical methods like `BlockAccept`, `BuildBlock`, and `Initialize` whose execution should be restricted to the sole AvalancheGo node.

However, there is no documentation for node operators to instruct them which endpoint should or should not be exposed.

Recommendation

We recommend creating documentation for node operators to guide them on which endpoints should be publicly exposed and which ones should remain private.

Status: Resolved

13. Missing type assertion and error handling for messages from `deliverTxSub.Out()`

Severity: Informational

In `vm/rpc.go:188`, the function assumes that the message from `deliverTxSub.Out()` will always be of type `types.EventDataTx`.

However, without a type assertion and proper error handling, this assumption can lead to panic if an unexpected message type is received.

Recommendation

We recommend adding a type assertion with error handling to ensure that only messages of type `types.EventDataTx` are processed.

Status: Resolved

14. Unnecessary panic in `validatePage` function

Severity: Informational

In `vm/rpc.go:480`, the `validatePage` function uses panic to handle the case where `perPage` is less than `1`.

Although `validatePerPage` is called before `validatePage` to ensure `perPage` is valid, this design is fragile. If a developer forgets to call `validatePerPage`, the server can panic, leading to potential DoS attacks.

Additionally, relying on panic for error handling in production code can cause the entire node to terminate.

Recommendation

We recommend refactoring the `validatePage` function to return an error instead of panicking. This will make the code more robust and prevent potential production issues due to unhandled panics. Ensure all parent functions handle the returned error appropriately.

Status: Resolved

15. Miscellaneous comments

Severity: Informational

Miscellaneous recommendations can be found below.

Recommendation

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

- Review and address each `TODO` context to ensure that all contexts satisfy the requirements (e.g., `vm/rpc.go:106`, `114`, `128` and `vm/types/state/executor.go`:`167`, `210`, `387`, `723`, `746`)
- Remove casting to `int` on `http/reader/reader_server.go:24`
- Update all dependencies, especially the version of the Go compiler (it has vulnerability GO-2024-2887) and `cosmos-sdk` module (it has vulnerability GO-2024-2571), to the latest versions
- Resolve all known overflow issues for 32-bit platforms (e.g., `vm/rpc.go:557`, `626`)
- In `vm/rpc.go:376`, the code checks if `height` is less than or equal to zero. Although this will return an error if the `height` is negative, the presence of negative height values by design can lead to undefined behavior because height should only be positive by definition.
- Do not ignore errors (e.g., `vm/vm.go:224`), instead, a log message could be emitted
- Review and address each `TODO` comment to ensure that all expected security checks and functionalities are properly implemented (e.g. `vm/vm.go:67`, `207`, `vm/rpc.go:210`, `242`, `347`, `352`, `420`, `756`)

Status: Resolved