Blackthorn

Security Review For **Interchain Labs**

Collaborative Audit Prepared For:	**Interchain Labs**
Lead Security Expert(s):	**0x007**
	defsec
	Oblivionis
	sammy
Date Audited:	**February 4 - March 4, 2025**

Introduction

Interchain Labs is a core contributor for the Cosmos ecosystem. The CosmWasm v2 audit focuses on ensuring the VM's safety.

Scope

Repository: CosmWasm/cosmwasm

Audited Commit: b9a149fde2a787b7221918b0651532ca2668d761

Final Commit: 7ec688a452ec0d5e847366a072eac0fbb2ad7ca7

Files:

- packages/check/src/main.rs
- packages/core/src/crypto.rs
- packages/core/src/lib.rs
- packages/crypto/benches/main.rs
- packages/crypto/src/backtrace.rs
- packages/crypto/src/bls12_381/aggregate.rs
- packages/crypto/src/bls12_381/constants.rs
- packages/crypto/src/bls12_381/hash.rs
- packages/crypto/src/bls12_381/mod.rs
- packages/crypto/src/bls12_381/pairing.rs
- packages/crypto/src/bls12_381/points.rs
- packages/crypto/src/ecdsa.rs
- packages/crypto/src/ed25519.rs
- packages/crypto/src/errors.rs
- packages/crypto/src/identity_digest.rs
- packages/crypto/src/lib.rs

- packages/crypto/src/secp256k1.rs
- packages/crypto/src/secp256r1.rs
- packages/derive/src/lib.rs
- packages/go-gen/src/go.rs
- packages/go-gen/src/main.rs
- packages/go-gen/src/schema.rs
- packages/go-gen/src/utils.rs
- packages/schema-derive/src/cw_serde.rs
- packages/schema-derive/src/error.rs
- packages/schema-derive/src/generate_api.rs
- packages/schema-derive/src/lib.rs
- packages/schema-derive/src/query_responses.rs
- packages/schema-derive/src/query_responses/context.rs
- packages/schema/src/casing.rs
- packages/schema/src/export.rs
- packages/schema/src/idl.rs
- packages/schema/src/lib.rs
- packages/schema/src/query_response.rs
- packages/schema/src/remove.rs
- packages/schema/src/schema_for.rs
- packages/std/src/__internal.rs
- packages/std/src/addresses.rs
- packages/std/src/assertions.rs
- packages/std/src/binary.rs

- packages/std/src/checksum.rs
- packages/std/src/coin.rs
- packages/std/src/coins.rs
- packages/std/src/conversion.rs
- packages/std/src/deps.rs
- packages/std/src/encoding.rs
- packages/std/src/errors/backtrace.rs
- packages/std/src/errors/mod.rs
- packages/std/src/errors/recover_pubkey_error.rs
- packages/std/src/errors/std_error.rs
- packages/std/src/errors/system_error.rs
- packages/std/src/errors/verification_error.rs
- packages/std/src/exports.rs
- packages/std/src/forward_ref.rs
- packages/std/src/hex_binary.rs
- packages/std/src/ibc.rs
- packages/std/src/ibc/callbacks.rs
- packages/std/src/ibc/transfer_msg_builder.rs
- packages/std/src/import_helpers.rs
- packages/std/src/imports.rs
- packages/std/src/iterator.rs
- packages/std/src/lib.rs
- packages/std/src/math/conversion.rs
- packages/std/src/math/decimal.rs

- packages/std/src/math/decimal256.rs
- packages/std/src/math/fraction.rs
- packages/std/src/math/int128.rs
- packages/std/src/math/int256.rs
- packages/std/src/math/int512.rs
- packages/std/src/math/int64.rs
- packages/std/src/math/isqrt.rs
- packages/std/src/math/mod.rs
- packages/std/src/math/num_consts.rs
- packages/std/src/math/signed_decimal.rs
- packages/std/src/math/signed_decimal_256.rs
- packages/std/src/math/uint128.rs
- packages/std/src/math/uint256.rs
- packages/std/src/math/uint512.rs
- packages/std/src/math/uint64.rs
- packages/std/src/memory.rs
- packages/std/src/metadata.rs
- packages/std/src/msgpack.rs
- packages/std/src/never.rs
- packages/std/src/pagination.rs
- packages/std/src/panic.rs
- packages/std/src/prelude.rs
- packages/std/src/query/bank.rs
- packages/std/src/query/distribution.rs

- packages/std/src/query/ibc.rs
- packages/std/src/query/mod.rs
- packages/std/src/query/query_response.rs
- packages/std/src/query/staking.rs
- packages/std/src/query/wasm.rs
- packages/std/src/results/contract_result.rs
- packages/std/src/results/cosmos_msg.rs
- packages/std/src/results/empty.rs
- packages/std/src/results/events.rs
- packages/std/src/results/mod.rs
- packages/std/src/results/query.rs
- packages/std/src/results/response.rs
- packages/std/src/results/submessages.rs
- packages/std/src/results/system_result.rs
- packages/std/src/sections.rs
- packages/std/src/serde.rs
- packages/std/src/stdack.rs
- packages/std/src/storage.rs
- packages/std/src/storage_keys/length_prefixed.rs
- packages/std/src/storage_keys/mod.rs
- packages/std/src/testing/assertions.rs
- packages/std/src/testing/message_info.rs
- packages/std/src/testing/mock.rs
- packages/std/src/testing/mod.rs

- packages/std/src/timestamp.rs
- packages/std/src/traits.rs
- packages/std/src/types.rs
- packages/vm-derive/src/hash_function.rs
- packages/vm-derive/src/lib.rs
- packages/vm/benches/main.rs
- packages/vm/examples/heap_profiling.rs
- packages/vm/examples/module_size.rs
- packages/vm/examples/module_size.sh
- packages/vm/examples/multi_threaded_cache.rs
- packages/vm/src/backend.rs
- packages/vm/src/cache.rs
- packages/vm/src/calls.rs
- packages/vm/src/capabilities.rs
- packages/vm/src/compatibility.rs
- packages/vm/src/config.rs
- packages/vm/src/conversion.rs
- packages/vm/src/environment.rs
- packages/vm/src/errors/backtrace.rs
- packages/vm/src/errors/communication_error.rs
- packages/vm/src/errors/mod.rs
- packages/vm/src/errors/region_validation_error.rs
- packages/vm/src/errors/vm_error.rs
- packages/vm/src/filesystem.rs

- packages/vm/src/imports.rs
- packages/vm/src/instance.rs
- packages/vm/src/lib.rs
- packages/vm/src/limited.rs
- packages/vm/src/memory.rs
- packages/vm/src/modules/cached_module.rs
- packages/vm/src/modules/file_system_cache.rs
- packages/vm/src/modules/in_memory_cache.rs
- packages/vm/src/modules/mod.rs
- packages/vm/src/modules/pinned_memory_cache.rs
- packages/vm/src/modules/versioning.rs
- packages/vm/src/parsed_wasm.rs
- packages/vm/src/sections.rs
- packages/vm/src/serde.rs
- packages/vm/src/size.rs
- packages/vm/src/static_analysis.rs
- packages/vm/src/testing/calls.rs
- packages/vm/src/testing/instance.rs
- packages/vm/src/testing/mock.rs
- packages/vm/src/testing/mod.rs
- packages/vm/src/testing/querier.rs
- packages/vm/src/testing/storage.rs
- packages/vm/src/wasm_backend/compile.rs
- packages/vm/src/wasm_backend/engine.rs

- packages/vm/src/wasm_backend/gatekeeper.rs
- packages/vm/src/wasm_backend/limiting_tunables.rs
- packages/vm/src/wasm_backend/metering.rs
- packages/vm/src/wasm_backend/mod.rs

Repository: CosmWasm/wasmd

Audited Commit: 04cb6e5408cc54c27247b0b327dfa99769d5103c

Final Commit: 5ed2ff8f9c9a3cd1912991c5f34f0530bda4c065

Files:

- benchmarks/app_test.go
- benchmarks/bench_test.go
- benchmarks/cw20_test.go
- benchmarks/testdata/download_releases.sh
- benchmarks/testdata/version.txt
- x/wasm/alias.go
- x/wasm/client/cli/gov_tx.go
- x/wasm/client/cli/gov_tx_test.go
- x/wasm/client/cli/new_tx.go
- x/wasm/client/cli/query.go
- x/wasm/client/cli/tx.go
- x/wasm/client/cli/tx_test.go
- x/wasm/client/cli/utils.go
- x/wasm/common_test.go
- x/wasm/exported/exported.go

- x/wasm/ibc.go
- x/wasm/ibc_test.go
- x/wasm/ioutils/ioutil.go
- x/wasm/ioutils/ioutil_test.go
- x/wasm/ioutils/utils.go
- x/wasm/ioutils/utils_test.go
- x/wasm/keeper/addresses.go
- x/wasm/keeper/addresses_test.go
- x/wasm/keeper/ante.go
- x/wasm/keeper/ante_test.go
- x/wasm/keeper/api.go
- x/wasm/keeper/api_test.go
- x/wasm/keeper/authz_policy.go
- x/wasm/keeper/authz_policy_test.go
- x/wasm/keeper/bench_test.go
- x/wasm/keeper/capabilities.go
- x/wasm/keeper/contract_keeper.go
- x/wasm/keeper/contract_keeper_test.go
- x/wasm/keeper/events.go
- x/wasm/keeper/events_test.go
- x/wasm/keeper/genesis.go
- x/wasm/keeper/genesis_test.go
- x/wasm/keeper/handler_plugin.go
- x/wasm/keeper/handler_plugin_encoders.go

- x/wasm/keeper/handler_plugin_encoders_test.go
- x/wasm/keeper/handler_plugin_test.go
- x/wasm/keeper/ibc.go
- x/wasm/keeper/ibc_test.go
- x/wasm/keeper/keeper.go
- x/wasm/keeper/keeper_cgo.go
- x/wasm/keeper/keeper_no_cgo.go
- x/wasm/keeper/keeper_test.go
- x/wasm/keeper/metrics.go
- x/wasm/keeper/migrations.go
- x/wasm/keeper/msg_dispatcher.go
- x/wasm/keeper/msg_dispatcher_test.go
- x/wasm/keeper/msg_server.go
- x/wasm/keeper/msg_server_test.go
- x/wasm/keeper/options.go
- x/wasm/keeper/options_test.go
- x/wasm/keeper/proposal_handler_legacy.go
- x/wasm/keeper/querier.go
- x/wasm/keeper/querier_test.go
- x/wasm/keeper/query_plugins.go
- x/wasm/keeper/query_plugins_test.go
- x/wasm/keeper/recurse_test.go
- x/wasm/keeper/reflect_test.go
- x/wasm/keeper/relay.go

- x/wasm/keeper/relay_test.go
- x/wasm/keeper/snapshotter.go
- x/wasm/keeper/staking_test.go
- x/wasm/keeper/submsg_test.go
- x/wasm/keeper/test_common.go
- x/wasm/keeper/test_fuzz.go
- x/wasm/keeper/testdata/contracts.go
- x/wasm/keeper/testdata/download_releases.sh
- x/wasm/keeper/testdata/genesis.json
- x/wasm/keeper/testdata/version.txt
- x/wasm/keeper/wasmtesting/extension_mocks.go
- x/wasm/keeper/wasmtesting/gas_register.go
- x/wasm/keeper/wasmtesting/message_router.go
- x/wasm/keeper/wasmtesting/messenger.go
- x/wasm/keeper/wasmtesting/mock_engine.go
- x/wasm/keeper/wasmtesting/mock_keepers.go
- x/wasm/keeper/wasmtesting/msg_dispatcher.go
- x/wasm/keeper/wasmtesting/query_handler.go
- x/wasm/keeper/wasmtesting/store.go
- x/wasm/migrations/v1/store.go
- x/wasm/migrations/v1/store_test.go
- x/wasm/migrations/v2/legacy_types.go
- x/wasm/migrations/v2/params_legacy.go
- x/wasm/migrations/v2/params_legacy_test.go

- x/wasm/migrations/v2/store.go
- x/wasm/migrations/v2/store_test.go
- x/wasm/migrations/v3/legacy_types.go
- x/wasm/migrations/v3/store.go
- x/wasm/migrations/v3/store_test.go
- x/wasm/module.go
- x/wasm/simulation/genesis.go
- x/wasm/simulation/operations.go
- x/wasm/simulation/proposals.go
- x/wasm/types/authz.go
- x/wasm/types/authz_policy.go
- x/wasm/types/authz_test.go
- x/wasm/types/codec.go
- x/wasm/types/context.go
- x/wasm/types/errors.go
- x/wasm/types/errors_test.go
- x/wasm/types/events.go
- x/wasm/types/expected_keepers.go
- x/wasm/types/exported_keepers.go
- x/wasm/types/feature_flag.go
- x/wasm/types/gas_register.go
- x/wasm/types/gas_register_test.go
- x/wasm/types/genesis.go
- x/wasm/types/genesis_test.go

- x/wasm/types/iavl_range_test.go
- x/wasm/types/json_matching.go
- x/wasm/types/json_matching_test.go
- x/wasm/types/keys.go
- x/wasm/types/keys_test.go
- x/wasm/types/params.go
- x/wasm/types/params_test.go
- x/wasm/types/proposal_legacy.go
- x/wasm/types/proposal_legacy.pb.go
- x/wasm/types/proposal_legacy_test.go
- x/wasm/types/test_fixtures.go
- x/wasm/types/tx.go
- x/wasm/types/tx_test.go
- x/wasm/types/types.go
- x/wasm/types/types_test.go
- x/wasm/types/validation.go
- x/wasm/types/wasmer_engine.go

Repository: CosmWasm/wasmvm

Audited Commit: bb5b8c1afc27b5df0cb9162ff44266a830991e68

Final Commit: bb5b8c1afc27b5df0cb9162ff44266a830991e68

Files:

- cmd/demo/main.go
- internal/api/api_test.go

- internal/api/bindings.h
- internal/api/callbacks.go
- internal/api/callbacks_cgo.go
- internal/api/iterator.go
- internal/api/iterator_test.go
- internal/api/lib.go
- internal/api/lib_test.go
- internal/api/link_glibclinux_aarch64.go
- internal/api/link_glibclinux_x86_64.go
- internal/api/link_mac.go
- internal/api/link_mac_static.go
- internal/api/link_muslc_aarch64.go
- internal/api/link_muslc_x86_64.go
- internal/api/link_system.go
- internal/api/link_windows.go
- internal/api/memory.go
- internal/api/memory_test.go
- internal/api/mock_failure.go
- internal/api/mocks.go
- internal/api/testdb/memdb.go
- internal/api/testdb/memdb_iterator.go
- internal/api/testdb/types.go
- internal/api/version.go
- internal/api/version_test.go

- lib.go
- lib_libwasmvm.go
- lib_libwasmvm_test.go
- lib_test.go
- libwasmvm/bindings.h
- libwasmvm/build.rs
- libwasmvm/cbindgen.toml
- libwasmvm/clippy.toml
- libwasmvm/src/api.rs
- libwasmvm/src/args.rs
- libwasmvm/src/cache.rs
- libwasmvm/src/calls.rs
- libwasmvm/src/db.rs
- libwasmvm/src/error/go.rs
- libwasmvm/src/error/mod.rs
- libwasmvm/src/error/rust.rs
- libwasmvm/src/examples/wasmvmstatic.rs
- libwasmvm/src/gas_meter.rs
- libwasmvm/src/gas_report.rs
- libwasmvm/src/handle_vm_panic.rs
- libwasmvm/src/iterator.rs
- libwasmvm/src/lib.rs
- libwasmvm/src/memory.rs
- libwasmvm/src/querier.rs

- libwasmvm/src/storage.rs
- libwasmvm/src/test_utils.rs
- libwasmvm/src/tests.rs
- libwasmvm/src/version.rs
- libwasmvm/src/vtables.rs
- types/api.go
- types/checksum.go
- types/config.go
- types/config_test.go
- types/env.go
- types/env_test.go
- types/fraction.go
- types/gas.go
- types/ibc.go
- types/ibc_test.go
- types/msg.go
- types/msg_test.go
- types/queries.go
- types/queries_test.go
- types/store.go
- types/submessages.go
- types/submessages_test.go
- types/systemerror.go
- types/systemerror_test.go

- types/types.go
- types/types_test.go
- version_cgo.go
- version_no_cgo.go

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
3	3	10

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Security Experts Dedicated to This Review

@0x007

0x007

@Oblivionis

Oblivionis

@defsec

defsec

@sammy

[illegible]

Issue H-1: Sub-context gas not consumed on non-OutOfGas panics

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/539

Summary

The gas consumption in sub-contexts is not properly accounted for when a non-OutOfGas panic occurs, leading to uncompensated CPU cycles and potential gas exhaustion attacks.

Vulnerability Detail

In the message dispatcher, when a submessage execution panics, the code only handles gas consumption for OutOfGas panics by consuming the full gas limit. However, for other types of panics, the gas consumed in the sub-context before the panic occurred is not charged to the parent context. This creates a discrepancy between actual resource usage and gas accounting.

Impact

1. Inaccurate gas accounting, as some gas usage goes uncharged
2. Potential gas exhaustion attacks where malicious contracts could perform operations without proper gas accounting
3. Loss of revenue for validators due to uncompensated CPU cycles

Code Snippet

Tool Used

Manual Review

Recommendation

Modify the panic handler to always consume the gas used in the sub-context, regardless of panic type:

Discussion

chipshort

Great finding! If I understand correctly, this can be used to consume up to the block gas limit of gas while paying only very little gas (essentially only some fees for calling the contract in the first place). Just to assess the risk: Do I understand correctly that it's not possible to use more than the block gas limit with this exploit? If so, then at least it cannot slow down the chain, so "only" the validator is harmed by it. My reasoning for that is that the subCtx still has the limit, so it would prevent you from consuming more than that and the panic would end the transaction, so you cannot use it multiple times to consume a multiple of the gas limit.

webmaster128

> panic would end the transaction

Not necessarily. With `wasmvmtypes.ReplyError` or `wasmvmtypes.ReplyAlways` you can have a tx with gas limit 1000 which then repeatedly

1. emit a sub message with limit of 800
2. use most of that limit
3. panic
4. reply

The sum of all gas used in 2. should be able to exceed the tx gas limit.

chipshort

> With `wasmvmtypes.ReplyError` or `wasmvmtypes.ReplyAlways` you can have a tx with gas limit 1000 which then repeatedly

AFAICS that's not possible because a panic in a submessage doesn't get caught, so once the first such sub message panics, the tx stops.

webmaster128

Ah, it indeed does get cought with the panic catcher in `dispatchMsgWithGasLimit` (using `recover()`), but due to the re-though of the panic you are right

Issue H-2: Missing contract setup cost in IBC packet handling leads to uncompensated node work

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/540

Summary

The contract setup cost is not properly charged in relay.go functions, leading to uncompensated work for the node.

Vulnerability Detail

In the keeper.go file, contract setup costs are explicitly charged using `k.gasRegister.SetupContractCost()`. However, in relay.go functions like `OnRecvPacket`, this setup cost is not charged, despite similar contract initialization and execution occurring. This creates an inconsistency in gas accounting and results in uncompensated work for the node.

Impact

1. Inaccurate gas accounting for IBC-related contract executions
2. Loss of revenue for validators due to uncompensated setup work
3. Potential economic imbalance between different contract execution paths
4. Incentive misalignment for nodes processing IBC packets

Code Snippet

In keeper.go:

In relay.go (missing setup cost):

Tool Used

Manual Review

Recommendation

Add consistent setup cost charging in relay.go functions:

This change ensures consistent gas accounting across all contract execution paths and proper compensation for node operators.

Discussion

chipshort

Valid issue. I'm not sure what to use for the `msgLen` parameter in all the different entrypoints because the json serialization happens one level below. Might have to think about that for a bit.

Issue H-3: Improper error handling may lead to IBC channel opening despite error

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/543

Summary

The `OnOpenChannel` function in `relay.go` fails to properly handle the case where `res.Err` is non-nil, leading to incorrect error propagation in the IBC channel handshake process. This can result in invalid channel openings being accepted by the chain.

Vulnerability Detail

In the `OnOpenChannel` function, when the contract execution returns a response with `res.Err` set, the function incorrectly returns a nil error. This causes the calling function (`OnChanOpenInit`) to proceed with channel opening, even though the contract has indicated an error condition.

The issue stems from the following problematic logic:

1. The contract returns a response with `res.Err` set to indicate an error
2. `OnOpenChannel` ignores this error and returns nil
3. `OnChanOpenInit` interprets the nil return as success and proceeds with channel opening

This behavior violates the IBC protocol specification, which requires that contract errors during channel opening should prevent the channel from being established.

Impact

1. Malicious or buggy contracts could open channels despite returning errors
2. Breaks the IBC protocol specification for channel handshake
3. Channels may be opened in invalid states

Code Snippet

Tool Used

Manual Review

Recommendation

Modify the error handling to properly propagate contract errors:

Discussion

chipshort

Really nice finding! This could even jeopardize non-malicious contracts that expect an error to abort the channel handshake.

However, I don't fully understand point 4 and 5 in the impact section. I doubt there is a mechanism to reject channel openings in that way and `res.Err` being deterministic or not (it should be) makes no difference, since we don't even use it here.

sammy-tm

I wasn't able to phrase the impacts properly while writing the report as it was last minute. About 5, I was under the impression that `res.Err` captures issues that may arise during local execution of the VM in a node, since it is possible to have distinct `res.Err` on different nodes (which is why it is wrapped in a `makeErrorDeterministic` error in other functions), I assumed that this may cause a consensus failure. But upon reviewing this now I think since `res.Err` is not caught altogether, impact 5 cannot happen. I apologize for the misjudgement.

4 is a 'potential' impact, which can be ignored if no such mechanism exists.

Issue M-1: Gas overcharging in validate Address()

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/531

Summary

The `validateAddress` function in `wasmd/x/wasm/keeper/api.go` is overcharging gas in two scenarios:

1. When `AccAddressFromBech32` fails, it charges full validation cost instead of just canonicalization cost
2. The total validation cost includes duplicate charges for address format verification which is performed twice

Vulnerability Detail

The `validateAddress` function charges `costValidate` (which equals `DefaultGasCostHumanAddress + DefaultGasCostCanonicalAddress`) in all cases. However:

1. When `AccAddressFromBech32` fails, it should only charge `costCanonical` since no humanization is performed
2. `costValidate` includes costs for both canonicalization and humanization, but `AccAddressFromBech32` already performs address format verification internally, making the final humanization cost redundant

Impact

Every address validation operation is overcharging gas:

- Failed validations charge 9 gas (5 + 4) when they should charge 4 gas
- Successful validations include duplicate charges for address format verification

While the impact per transaction is small (a few gas units), this affects every transaction that involves address validation, leading to systematic overcharging across the network.

Code Snippet

```
go:wasmd/x/wasm/keeper/api.go
func validateAddress(human string) (uint64, error) {
canonicalized, err := sdk.AccAddressFromBech32(human)
if err != nil {
return costValidate, err // Overcharges by using costValidate instead of
 ↪  costCanonical
}
// AccAddressFromBech32 already calls VerifyAddressFormat, so we can just humanize
 ↪  and compare
if canonicalized.String() != human {
return costValidate, errors.New("address not normalized")
}
return costValidate, nil
}
```

Tool Used

Manual Review

Recommendation

Update `costValidate` so it accounts for address format verification only once. Also, Update the `validateAddress` function to charge appropriate gas costs:

```
go:wasmd/x/wasm/keeper/api.go
func validateAddress(human string) (uint64, error) {
canonicalized, err := sdk.AccAddressFromBech32(human)
if err != nil {
return costCanonical, err // Only charge canonicalization cost on failure
}
// AccAddressFromBech32 already calls VerifyAddressFormat, so we only need
 ↪  costCanonical + String() cost
if canonicalized.String() != human {
return costValidate, errors.New("address not normalized")
}
return costValidate, nil // Charge for both operations but avoid double
↪  verification cost
}
```

Discussion

chipshort

I'd say overcharging is more of a low/info severity. Undercharging is much more problematic. Nevertheless, this is a very valid finding and should be addressed.

chipshort

Created a PR here: https://github.com/CosmWasm/wasmd/pull/2193

Issue M-2: Deprecated context.Consensus Params() usage creates compatibility risk

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/535

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The codebase uses the deprecated `ctx.ConsensusParams()` method from `sdk.Context` in the `LimitSimulationGasDecorator` ante handler, which has been removed in Cosmos SDK 0.52. This creates a future compatibility issue as the project will break when upgrading to newer SDK versions.

Vulnerability Detail

In Cosmos SDK 0.52, `sdk.Context` has been significantly refactored, with consensus parameters moved out of the global context state. The code in `LimitSimulationGasDecorator.AnteHandle()` directly calls `ctx.ConsensusParams()` to retrieve block gas limits, which will not be available in this manner in SDK 0.52+.

According to the Cosmos SDK migration documentation, the global state from context has been replaced with module-specific environments through `appmodule.Environment`. Consensus parameters should now be obtained from the consensus module directly.

Reference : Document

Impact

In Cosmos SDK 0.52, `sdk.Context` has been significantly refactored, with consensus parameters moved out of the global context state. The code in `LimitSimulationGasDecorator.AnteHandle()` directly calls `ctx.ConsensusParams()` to retrieve block gas limits, which will not be available in this manner in SDK 0.52+.

Code Snippet

ante.go#L102-L103

```
// AnteHandle that limits the maximum gas available in simulations only.
// A custom max value can be configured and will be applied when set. The value
↪  should not
// exceed the max block gas limit.
// Different values on nodes are not consensus breaking as they affect only
// simulations but may have effect on client user experience.
//
// When no custom value is set then the max block gas is used as default limit.
func (d LimitSimulationGasDecorator) AnteHandle(ctx sdk.Context, tx sdk.Tx,
↪  simulate bool, next sdk.AnteHandler) (sdk.Context, error) {
    if !simulate {
        // Wasm code is not executed in checkTX so that we don't need to limit it
↪  further.
        // Tendermint rejects the TX afterwards when the tx.gas > max block gas.
        // On deliverTX we rely on the tendermint/sdk mechanics that ensure
        // tx has gas set and gas < max block gas
        return next(ctx, tx, simulate)
    }

    // apply custom node gas limit
    if d.gasLimit != nil {
        return next(ctx.WithGasMeter(storetypes.NewGasMeter(*d.gasLimit)), tx,
↪  simulate)
    }

    // default to max block gas when set, to be on the safe side
    params := ctx.ConsensusParams()
    if maxGas := params.GetBlock().MaxGas; maxGas > 0 {
        return
↪  next(ctx.WithGasMeter(storetypes.NewGasMeter(storetypes.Gas(maxGas))), tx,
↪  simulate)
    }
    return next(ctx, tx, simulate)
}
```

Tool Used

Manual Review

Recommendation

Refactor the code to use the new pattern for accessing consensus parameters.

Discussion

chipshort

Interchain Labs decided to skip 0.52 and reverted the big refactor they did there. This function is no longer marked as deprecated in the latest v0.53.0-beta.3 release, so I think it can stay as-is.

defsec

marked as acknowledged.

Issue M-3: Missing version validation in On ChanOpenInit IBC handler implementation

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/536

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The `OnChanOpenInit` function in the IBCHandler implementation does not properly validate or handle empty version strings per IBC protocol requirements, potentially allowing incompatible channel handshakes to succeed when they should fail.

Vulnerability Detail

According to the IBC specification, when `OnChanOpenInit` is called with an empty version string, the implementation should:

1. Return a default version string representing the version(s) it supports, OR
2. Return an error if there is no default version string for the application

However, the current implementation in the IBCHandler simply passes the version string to the WASM contract without validating whether it's empty or providing a default:

```
msg := wasmvmtypes.IBCChannelOpenMsg{
    OpenInit: &wasmvmtypes.IBCOpenInit{
        Channel: wasmvmtypes.IBCChannel{
            // ...other fields...
            // DESIGN V3: this may be "" ??
            Version: version,
            // ...
        },
    },
}
```

The comment `// DESIGN V3: this may be "" ??` indicates awareness of the issue but no proper handling has been implemented.

Impact

This vulnerability could lead to:

1. **Incompatible Channel Establishment:** Channels may be established between applications with incompatible versions
2. **Protocol Violations**: Violates the IBC specification requirements for version negotiation

Code Snippet

ibc.go#L47-L48

```
func (i IBCHandler) OnChanOpenInit(
    ctx sdk.Context,
    order channeltypes.Order,
    connectionHops []string,
    portID string,
    channelID string,
    chanCap *capabilitytypes.Capability,
    counterParty channeltypes.Counterparty,
    version string,
) (string, error) {
    // ensure port, version, capability
    if err := ValidateChannelParams(channelID); err != nil {
        return "", err
    }
    contractAddr, err := keeper.ContractFromPortID(portID)
    if err != nil {
        return "", errorsmod.Wrapf(err, "contract port id")
    }

    msg := wasmvmtypes.IBCChannelOpenMsg{
        OpenInit: &wasmvmtypes.IBCOpenInit{
            Channel: wasmvmtypes.IBCChannel{
                Endpoint:              wasmvmtypes.IBCEndpoint{PortID: portID,
↪   ChannelID: channelID},
                CounterpartyEndpoint: wasmvmtypes.IBCEndpoint{PortID:
↪   counterParty.PortId, ChannelID: counterParty.ChannelId},
                Order:                 order.String(),
                // DESIGN V3: this may be "" ??
                Version:       version,
                ConnectionID: connectionHops[0],
            },
        },
    }
    // No version validation or default version assignment
    // ...
}
```

Tool Used

Manual Review

Recommendation

Implement proper version validation and handling according to the IBC specification.

Discussion

chipshort

I doubt that this is something that can/should be handled on the VM level. The VM simply does not know what version the contract supports. But we currently already have a check that prevents an empty version from being returned: https://github.com/CosmWasm/wasmd/blob/eb231283dbc38f433a8a8e634a5eef6d0fa5cbce/x/wasm/ibc.go allowbreak #L81-L83

So, we are already following the spec you linked:

> If there is no default version string for the application, it should return an error if the provided version is an empty string.

If the provided version is empty, we either return the version the contract returns (i.e. the default version) or error if it is empty.

Issue L-1: CosmWasm contracts can compile with migrate_with_info entry point

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/529

Summary

CosmWasm contracts can compile with the `migrate_with_info` entry point which could lead to unexpected erros and confusion.

Vulnerability Detail

In the new 2.2.0 update, the `migrate` entry point has been updated to accept either 2 or 3 arguments based on the dev's choice. This has been implemented by using a `migrate_with_info` function under the belt. However, the contract can also compile with the `migrate_with_info` entry point which can lead to confusion and unintended errors if the dev implements both entry points in their contract.

Impact

If a dev implements both `migrate` (with 2 args) and `migrate_with_info` entry points in the contract, each having different underlying logic, the contract will compile successfully. However, when the contract admin calls `migrate_with_info`, `migrate` (2 args) will be called instead because of how `migrate_with_info` is implemented in `calls.rs`.

Recommendation

This has no High or Medium impact as the contract admin can just deploy another contract without `migrate_with_info` and then call the `migrate` function. However, it would be best practice to catch this issue earlier during compilation.

Discussion

chipshort

Valid finding and we fixed it already in this PR.

Issue L-2: Missing IBC callback entrypoints in contract analysis

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/530

Description

In the CosmWasm VM's static analysis of smart contracts, the `Entrypoint` enum is missing important IBC callback entrypoints that contracts can implement. While these callbacks are not required for all IBC-enabled contracts (hence not included in `REQUIRED_IBC_EXPORTS`), the static analysis should still be able to identify them when present.

The issue is found in the `static_analysis.rs` file where the `Entrypoint` enum defines the known contract entrypoints:

```
#[derive(PartialEq, Eq, Debug, Clone, Copy, Hash, EnumString, Display, AsRefStr)]
pub enum Entrypoint {
#[strum(serialize = "instantiate")]
Instantiate,
// ... other standard entrypoints ...
#[strum(serialize = "ibc_channel_open")]
IbcChannelOpen,
#[strum(serialize = "ibc_channel_connect")]
IbcChannelConnect,
#[strum(serialize = "ibc_channel_close")]
IbcChannelClose,
#[strum(serialize = "ibc_packet_receive")]
IbcPacketReceive,
#[strum(serialize = "ibc_packet_ack")]
IbcPacketAck,
#[strum(serialize = "ibc_packet_timeout")]
IbcPacketTimeout,
} // @audit-info ibc callbacks missing
```

The enum is missing two optional but important IBC callback entrypoints:

1. `ibc_source_callback`
2. `ibc_destination_callback`

These callbacks are used by contracts that need to perform additional validation or setup during the IBC channel handshake process.

The issue affects the contract analysis (`analyze_code`) performed in `cache.rs` when analyzing a contract's capabilities.

Impact

1. The static analysis cannot identify contracts that implement these optional callbacks
2. Tools and UIs that rely on the contract analysis may not show complete information about a contract's IBC capabilities
3. This could make it harder to distinguish between contracts that implement different levels of IBC functionality

Mitigation

Add the missing IBC callback entrypoints to the `Entrypoint` enum

Discussion

chipshort

Valid finding. I don't think there are any external users of the contract analysis, so I don't see much risk in this.

I prepared a PR for this: https://github.com/CosmWasm/cosmwasm/pull/2438

Issue L-3: Integer type mismatch between CosmWasm and Cosmos SDK

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/532

Summary

There is a type mismatch in how integers are handled between CosmWasm and Cosmos SDK:

- **CosmWasm** uses `uint128` for coin variables
- **Cosmos SDK** uses `uint256` for integer types

This inconsistency creates friction when developing applications that interact with both systems, particularly during testing and when handling token amounts.

Vulnerability Detail

There is a type mismatch in how integers are handled between CosmWasm and Cosmos SDK:

- **CosmWasm** uses `uint128` for coin variables
- **Cosmos SDK** uses `uint256` for integer types

This inconsistency creates friction when developing applications that interact with both systems, particularly during testing and when handling token amounts.

Impact

1. **Development Friction:** Developers must constantly convert between different integer types
2. **Error Potential:** Type conversion can introduce subtle bugs, especially when dealing with large values

Code Snippet

contract.rs#L83-L84

```
pub fn transfer(
    deps: DepsMut,
    _env: Env,
    info: MessageInfo,
    recipient: String,
    send: Uint128,
) -> StdResult<Response> {
    let rcpt_raw = deps.api.addr_canonicalize(&recipient)?;
    let sender_raw = deps.api.addr_canonicalize(info.sender.as_str())?;

    let balance = may_load_map(deps.storage, PREFIX_BALANCE,
 ↪  &sender_raw)?.unwrap_or_default();
    save_map(
        deps.storage,
        PREFIX_BALANCE,
        &sender_raw,
        balance.checked_sub(send)?,
    )?;
    let balance = may_load_map(deps.storage, PREFIX_BALANCE,
 ↪  &rcpt_raw)?.unwrap_or_default();
    save_map(deps.storage, PREFIX_BALANCE, &rcpt_raw, balance + send)?;

    let res = Response::new()
        .add_attribute("action", "transfer")
        .add_attribute("from", info.sender)
        .add_attribute("to", recipient)
        .add_attribute("amount", send.to_string());
    Ok(res)
}
```

Tool Used

Manual Review

Recommendation

Modify the Coin implementation to better support Uint256.

Discussion

chipshort

Valid point, we already opened an issue for this in January.

Issue L-4: Missing HTML character filtering in ValidateLabel enables XSS attacks

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/534

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The `ValidateLabel` function does not filter HTML/JavaScript special characters, allowing XSS payloads to pass validation and potentially enabling cross-site scripting attacks when these labels are displayed in web interfaces.

Vulnerability Detail

The current implementation of `ValidateLabel` checks for:

- Non-empty strings
- Maximum length constraints
- No leading/trailing whitespace
- Printable characters only

However, it fails to filter or sanitize HTML/JavaScript special characters such as <, >, ", ', /, and others that can be used in XSS attacks. This means strings like `<script>alert('XSS')</script>` or `<img src=x onerror=alert(1)>` will pass validation successfully.

While the validation function itself is not directly vulnerable, it creates a critical security gap if validated labels are later rendered in HTML contexts without proper escaping.

Proof Of Concept

```
package main

import (
    "fmt"
```

```go
    "strings"
    "unicode"
)

// Error types for validation
type wrappedError struct {
    msg string
}

func (e wrappedError) Error() string {
    return e.msg
}

func (e wrappedError) Wrap(msg string) error {
    return wrappedError{msg: e.msg + ": " + msg}
}

func (e wrappedError) Wrapf(format string, args ...interface{}) error {
    return wrappedError{msg: e.msg + ": " + fmt.Sprintf(format, args...)}
}

var (
    ErrEmpty   = wrappedError{msg: "empty"}
    ErrLimit   = wrappedError{msg: "limit exceeded"}
    ErrInvalid = wrappedError{msg: "invalid"}
)

const MaxLabelSize = 50

// ValidateLabel ensure label constraints
func ValidateLabel(label string) error {
    if label == "" {
        return ErrEmpty.Wrap("is required")
    }
    if len(label) > MaxLabelSize {
        return ErrLimit.Wrapf("cannot be longer than %d characters", MaxLabelSize)
    }
    if label != strings.TrimSpace(label) {
        return ErrInvalid.Wrap("label must not start/end with whitespaces")
    }
    labelWithPrintableCharsOnly := strings.Map(func(r rune) rune {
        if unicode.IsPrint(r) {
            return r
        }
        return -1
    }, label)
    if label != labelWithPrintableCharsOnly {
        return ErrInvalid.Wrap("label must have printable characters only")
    }
    return nil
}

func main() {
    // XSS Payload Test Cases
    payloads := []string{
        // Basic XSS
        "<script>alert('XSS')</script>",
        "<img src=x onerror=alert('XSS')>",
        "<svg onload=alert('XSS')>",

        // JavaScript protocol
        "javascript:alert('XSS')",

        // Event handlers
        "<body onload=alert('XSS')>",
        "<a href='' onclick=alert('XSS')>Click me</a>",

        // CSS-based attacks
        "<div style=\"background-image: url('javascript:alert(`XSS`)')\">",

        // Attribute-breaking
        "\" onmouseover=\"alert('XSS')\" \"",
        "' onmouseover='alert(\"XSS\")' '",

        // Template injection
        "{{7*7}}",
        "${7*7}",

        // DOM-based XSS
        "<img src=1 href=1 onerror=\"javascript:alert(document.cookie)\">",

        // Bypassing WAF/filters
        "<scr<script>ipt>alert('XSS')</scr</script>ipt>",
        "<SCRIPT SRC=http://xss.rocks/xss.js></SCRIPT>",

        // Mixed case to bypass filters
        "<ScRiPt>alert('XSS')</sCrIpT>",

        // HTML5 vectors
        "<svg><animate xlink:href=#xss attributeName=href
↪   values=javascript:alert(1) /><a id=xss><text x=20 y=20>XSS</text></a>",

        // Exotic payloads
        "<math><mtext><table><mglyph><style><!--</style><img
↪   title=\"--&gt;&lt;/mglyph&gt;&lt;img src=1
↪   onerror=alert(1)&gt;\"></table></mtext></math>",

        // Unicode/encoding tricks
        " script alert(1) /script ", // Fullwidth characters
        "<script> =''‚alert(1)</script>", // With special unicode chars

        // Non-alphanumeric XSS
        "\"><iframe/src=javascript:alert(2)>",

        // JSON injection
        "\", \"malicious\": true, \"hack\": \"true",

        // Special character tests
        "Label' OR '1'='1",
        "Label\" OR \"1\"=\"1",
    }

    fmt.Println("XSS PAYLOAD VALIDATION TEST")
    fmt.Println("===========================\n")
    for i, payload := range payloads {
        err := ValidateLabel(payload)
        status := "  PASSED" // The payload passed validation (could be XSS
↪   vulnerability)
        if err != nil {
            status = "  REJECTED" // Good - payload was rejected
        }

        fmt.Printf("%02d. %s: %s\n", i+1, status, payload)
        if err != nil {
            fmt.Printf("    Error: %s\n", err)
        } else {
            fmt.Printf("    WARNING: This payload would pass validation\n")
        }
        fmt.Println()
    }

    fmt.Println("\nSUMMARY")
    fmt.Println("=======")
    fmt.Println("The ValidateLabel function only checks for:")
    fmt.Println("- Non-empty strings")
    fmt.Println("- Max length")
    fmt.Println("- No leading/trailing whitespace")
    fmt.Println("- Printable characters only")
    fmt.Println("\nIt DOES NOT filter out XSS payloads!")
    fmt.Println("Additional HTML escaping is required when rendering labels.")
}
```

Output:

```
01.   PASSED: <script>alert('XSS')</script>
    WARNING: This payload would pass validation

02.   PASSED: <img src=x onerror=alert('XSS')>
    WARNING: This payload would pass validation

03.   PASSED: <svg onload=alert('XSS')>
```

```
    WARNING: This payload would pass validation

04.  PASSED: javascript:alert('XSS')
    WARNING: This payload would pass validation

05.  PASSED: <body onload=alert('XSS')>
    WARNING: This payload would pass validation

06.  PASSED: <a href='' onclick=alert('XSS')>Click me</a>
    WARNING: This payload would pass validation

07.  REJECTED: <div style="background-image: url('javascript:alert(`XSS`)')">
    Error: limit exceeded: cannot be longer than 50 characters

08.  PASSED: " onmouseover="alert('XSS')" "
    WARNING: This payload would pass validation

09.  PASSED: ' onmouseover='alert("XSS")' '
    WARNING: This payload would pass validation

10.  PASSED: {{7*7}}
    WARNING: This payload would pass validation

11.  PASSED: ${7*7}
    WARNING: This payload would pass validation

12.  REJECTED: <img src=1 href=1 onerror="javascript:alert(document.cookie)">
    Error: limit exceeded: cannot be longer than 50 characters

13.  PASSED: <scr<script>ipt>alert('XSS')</scr</script>ipt>
    WARNING: This payload would pass validation

14.  PASSED: <SCRIPT SRC=http://xss.rocks/xss.js></SCRIPT>
    WARNING: This payload would pass validation

15.  PASSED: <ScRiPt>alert('XSS')</sCrIpT>
    WARNING: This payload would pass validation
```

```
16.  REJECTED: <svg><animate xlink:href=#xss attributeName=href
 ↪  values=javascript:alert(1) /><a id=xss><text x=20 y=20>XSS</text></a>
    Error: limit exceeded: cannot be longer than 50 characters

17.  REJECTED: <math><mtext><table><mglyph><style><!--</style><img
 ↪  title="--&gt;&lt;/mglyph&gt;&lt;img src=1
 ↪  onerror=alert(1)&gt;"></table></mtext></math>
    Error: limit exceeded: cannot be longer than 50 characters

18.  PASSED:  script alert(1) /script
    WARNING: This payload would pass validation

19.  PASSED: <script> ='',alert(1)</script>
    WARNING: This payload would pass validation

20.  PASSED: "><iframe/src=javascript:alert(2)>
    WARNING: This payload would pass validation

21.  PASSED: ", "malicious": true, "hack": "true
    WARNING: This payload would pass validation

22.  PASSED: Label' OR '1'='1
    WARNING: This payload would pass validation

23.  PASSED: Label" OR "1"="1
    WARNING: This payload would pass validation
```

Impact

If validated labels are rendered in HTML contexts without proper escaping or sanitization, attackers can:

1. **Execute arbitrary JavaScript** in users' browsers
2. **Steal sensitive information** such as authentication tokens or cookies
3. **Perform actions on behalf of users** without their knowledge

Code Snippet

```go
func ValidateLabel(label string) error {
    if label == "" {
        return errorsmod.Wrap(ErrEmpty, "is required")
    }
    if len(label) > MaxLabelSize {
        return ErrLimit.Wrapf("cannot be longer than %d characters", MaxLabelSize)
    }
    if label != strings.TrimSpace(label) {
        return ErrInvalid.Wrap("label must not start/end with whitespaces")
    }
    labelWithPrintableCharsOnly := strings.Map(func(r rune) rune {
        if unicode.IsPrint(r) {
            return r
        }
        return -1
    }, label)
    if label != labelWithPrintableCharsOnly {
        return ErrInvalid.Wrap("label must have printable characters only")
    }
    return nil
}
```

Tool Used

Manual Review

Recommendation

Implement one or more of these approaches:

1. **HTML Entity Encoding:** Modify the validation to encode or reject HTML special characters:

```go
func ValidateLabel(label string) error {
    // Existing checks...
    // Check for potential XSS characters
    if strings.ContainsAny(label, "<>'\"/") {
        return ErrInvalid.Wrap("label contains forbidden characters")
    }
    return nil
}
```

2. **Context-Based Approach:** Since the function only validates input syntax, ensure proper HTML escaping is always applied when rendering labels:

```
// When displaying labels:
htmlEscapedLabel := html.EscapeString(label)
```

Discussion

chipshort

We don't really promote the use of labels in HTML and have no control over where people use labels. Sanitizing it is up to the user, which also allows much more specific sanitization. You want to save it in a database? Better make sure you cannot have an SQL injection! You want to use it in HTML? Better make sure to prevent XSS. But all of that is not our responsibility. That being said, I think it does make sense to restrict it to printable characters, so the `unicode.IsPrint` check sounds like a good idea.

Issue L-5: Using deprecated paramsKeeper in wasmApp struct will break in future SDK versions

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/537

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The `WasmApp` struct still includes the deprecated `ParamsKeeper` which will be fully removed in future Cosmos SDK releases. According to SDK development plans, the `x/params` module has been deprecated in v53 and will be removed entirely in a following release.

Vulnerability Detail

The application currently includes and depends on the `x/params` module, which has been officially deprecated in Cosmos SDK v53. As noted in cosmos/cosmos-sdk#23834, the params module is slated for complete removal in an upcoming release. The current implementation still imports the params packages and includes a ParamsKeeper in the WasmApp struct:

```
// imports the deprecated modules
"github.com/cosmos/cosmos-sdk/x/params"
paramsclient "github.com/cosmos/cosmos-sdk/x/params/client"
paramskeeper "github.com/cosmos/cosmos-sdk/x/params/keeper"
paramstypes "github.com/cosmos/cosmos-sdk/x/params/types"
paramproposal "github.com/cosmos/cosmos-sdk/x/params/types/proposal"

// WasmApp struct still includes a ParamsKeeper
type WasmApp struct {
    // ... other fields
    ParamsKeeper          paramskeeper.Keeper
    // ... other keepers
}
```

This indicates the application hasn't been migrated to use the new recommended parameter handling mechanisms introduced in recent Cosmos SDK versions.

Impact

Continued use of deprecated modules has several negative consequences:

The application will break when upgrading to future SDK versions where `x/params` is completely removed.

Code Snippet

```
// In imports
"github.com/cosmos/cosmos-sdk/x/params"
paramsclient "github.com/cosmos/cosmos-sdk/x/params/client"
paramskeeper "github.com/cosmos/cosmos-sdk/x/params/keeper"
paramstypes "github.com/cosmos/cosmos-sdk/x/params/types"
paramproposal "github.com/cosmos/cosmos-sdk/x/params/types/proposal"

// In WasmApp struct definition
type WasmApp struct {
    // ...
    ParamsKeeper          paramskeeper.Keeper
    // ...
}
```

Tool Used

Manual Review

Recommendation

Migrate away from the `x/params` module.

Discussion

chipshort

This is valid, but as far as I can see, we only use it for the migration that migrates away from the `x/params` module.

The issue you linked, mentions:

> are there any legacy interfaces that folks use that we still should export for continuity?

Given that, I think it makes more sense to wait until they remove it in the SDK and then we know for sure which parts need to be removed on our side. Because as of right now almost all the `AppModules` and the IBC Keepers still require a `Subspace` for said migration, which we can (as far as I can see) only get through the ParamsKeeper.

Issue L-6: StoreCodeUnchecked bypasses compilation cost verification

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/538

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The `StoreCodeUnchecked` function bypasses compilation cost verification, creating a potential vector for resource exhaustion.

Vulnerability Detail

In the CosmWasm library, when storing contract code on-chain, the standard path validates the WASM code and calculates the compilation cost to ensure it doesn't exceed the maximum allowable limit. However, the `StoreCodeUnchecked` function circumvents these checks, allowing potentially expensive-to-compile contract code to be stored without enforcing resource consumption limits.

Impact

Bypasses the economic security model designed to prevent resource abuse.

Code Snippet

lib_libwasmvm.go#L95-L96)

```
//
// For example, the code for all ERC-20 contracts should be the same.
// This function stores the code for that contract only once, but it can
// be instantiated with custom inputs in the future.
//
// Returns both the checksum, as well as the gas cost of compilation (in CosmWasm
 ↪  Gas) or an error.
```

```go
func (vm *VM) StoreCode(code WasmCode, gasLimit uint64) (Checksum, uint64, error) {
    gasCost := compileCost(code)
    if gasLimit < gasCost {
        return nil, gasCost, types.OutOfGasError{}
    }

    checksum, err := api.StoreCode(vm.cache, code, true)
    return checksum, gasCost, err
}

// SimulateStoreCode is the same as StoreCode but does not actually store the code.
// This is useful for simulating all the validations happening in StoreCode without
↪  actually
// writing anything to disk.
func (vm *VM) SimulateStoreCode(code WasmCode, gasLimit uint64) (Checksum, uint64,
↪  error) {
    gasCost := compileCost(code)
    if gasLimit < gasCost {
        return nil, gasCost, types.OutOfGasError{}
    }

    checksum, err := api.StoreCode(vm.cache, code, false)
    return checksum, gasCost, err
}

// StoreCodeUnchecked is the same as StoreCode but skips static validation checks.
// Use this for adding code that was checked before, particularly in the case of
↪  state sync.
func (vm *VM) StoreCodeUnchecked(code WasmCode) (Checksum, error) {
    return api.StoreCodeUnchecked(vm.cache, code)
}

func (vm *VM) RemoveCode(checksum Checksum) error {
    return api.RemoveCode(vm.cache, checksum)
}
```

Tool Used

Manual Review

Recommendation

Add similar compilation cost on the function like a `StoreCode`.

Discussion

chipshort

This is kind of expected. As the comment indicates, this function is basically only intended for state sync, where it doesn't make sense to charge gas. But I think, we should add a note to the docs about it not charging gas.

defsec

Marked as an acknowledged.

chipshort

Opened a PR for this: https://github.com/CosmWasm/wasmvm/pull/634

Issue L-7: Missing #[non_exhaustive] attribute on GovMsg enum

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/541

Summary

The `GovMsg` enum in the CosmWasm standard library is missing the `#[non_exhaustive]` attribute that is consistently applied to all other message enum types in the codebase.

Vulnerability Detail

In the provided codebase, all message enums such as `BankMsg`, `WasmMsg`, `StakingMsg`, and `DistributionMsg` are marked with the `#[non_exhaustive]` attribute, which signals to consumers that new variants might be added in the future. This encourages users to handle potential future variants in match statements.

However, the `GovMsg` enum lacks this attribute:

```
#[cfg(feature = "stargate")]
#[derive(Serialize, Deserialize, Clone, Debug, PartialEq, Eq, JsonSchema)]
#[serde(rename_all = "snake_case")]
pub enum GovMsg {
    /// This maps directly to [MsgVote]
    Vote {
        proposal_id: u64,
        /// The vote option.
        option: VoteOption,
    },
    /// This maps directly to [MsgVoteWeighted]
    #[cfg(feature = "cosmwasm_1_2")]
    VoteWeighted {
        proposal_id: u64,
        options: Vec<WeightedVoteOption>,
    },
}
```

Impact

Adding new governance message types would break existing code that doesn't handle unknown variants.

Code Snippet

/cosmwasm/packages/std/src/results/cosmos_msg.rs#L372-L373

```
#[cfg(feature = "stargate")]
#[derive(Serialize, Deserialize, Clone, Debug, PartialEq, Eq, JsonSchema)]
#[serde(rename_all = "snake_case")]
pub enum GovMsg {
    Vote {
        proposal_id: u64,
        option: VoteOption,
    },
    #[cfg(feature = "cosmwasm_1_2")]
    VoteWeighted {
        proposal_id: u64,
        options: Vec<WeightedVoteOption>,
    },
}
```

Tool Used

Manual Review

Recommendation

Add the `#[non_exhaustive]` attribute to the `GovMsg` enum to maintain consistency with other message types.

Discussion

chipshort

Good point. Fixed in https://github.com/CosmWasm/cosmwasm/pull/2426

defsec

Fixed

Issue L-8: Empty public key and signature can batch-verify any ed25519 message

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/542

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The `ed25519_batch_verify` function in `packages/crypto/src/ed25519.rs` deviate from `ed25519_verify`, which may lead to potential issues.

Vulnerability Detail

The `ed25519_batch_verify` function in `packages/crypto/src/ed25519.rs`

> Three Variants are supported in the input for convenience:
> - Equal number of messages, signatures, and public keys: Standard, generic functionality.
> - One message, and an equal number of signatures and public keys: Multiple digital signature (multisig) verification of a single message.
> - One public key, and an equal number of messages and signatures: Verification of multiple messages, all signed with the same private key.

It allows the following three special cases:

- The ”one-message, with zero signatures and zero public keys” case, is considered valid.
- The ”one-public key, with zero messages and zero signatures” case, is considered valid.
- The ”no messages, no signatures and no public keys” case, is considered valid.

This introduces a discrepancy in the protocol: when developers use `ed25519_verify`, the above three cases are considered invalid; however, when they use `ed25519_batch_verify`, the above three cases are considered valid. This could lead to security vulnerabilities in code written by unsuspecting library users.

Impact

Uninformed developers might write code that allows empty signatures + non-empty message to pass.

Code Snippet

https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/blob/main/cosmwasm/packages/crypto/src/ed25519.rs#L61

```
if messages_len == signatures_len && messages_len == public_keys_len { // We're
 ↪  good to go
} else if messages_len == 1 && signatures_len == public_keys_len {
    // Replicate message, for multisig
    messages = messages.repeat(signatures_len);
} else if public_keys_len == 1 && messages_len == signatures_len {
    // Replicate pubkey
    public_keys = public_keys.repeat(messages_len);
} else {
    return Err(CryptoError::batch_err(
        "Mismatched / erroneous number of messages / signatures / public keys",
    ));
}
```

Tool Used

Manual Review

Recommendation

In practice, none of the above three signatures have any real significance, nor are they common practice in the industry. If you don't wish to fix it, you could also update the documentation to clarify this situation.

Discussion

chipshort

While I understand the concern about potentially incorrect usage, we think the current implementation makes sense. All the special cases you mention can be summed up as: When you provide zero signatures then that means all signatures are valid.

I don't understand how that is supposed to be inconsistent with `ed25519_verify`. With `ed25519_verify` you have no way to provide zero signatures. You always have to provide one, so none of the special cases apply here.

All these special cases are already documented in `cosmwasm-crypto`, but we should probably also add this documentation to the `Api` trait, so contract developers have easy access to it.

Issue L-9: Missing error handling for multi-connection hop IBC channels

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/545

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The IBC querier in the provided code incorrectly handles channels with multiple connection hops by silently taking only the first connection and discarding the rest, without returning any error or indication to the calling contract that the channel uses multiple hops.

Vulnerability Detail

IBC channels can be configured with multiple connection hops, allowing for relaying across multiple chains. The current implementation in both the `ListChannels` and `Channel` query handlers only extracts and returns the first connection hop:

```
ConnectionID: ch.ConnectionHops[0]
```

and

```
ConnectionID: got.ConnectionHops[0]
```

This implementation:

1. Assumes that all channels have exactly one connection hop
2. Silently discards any additional connection hops
3. Provides no indication to the calling contract that the channel might be using multiple hops
4. Does not validate the length of `ConnectionHops` before accessing index 0

Impact

Smart contracts receive incomplete information about channel configuration.

Code Snippet

```
// In ListChannels handler:
channels = append(channels, wasmvmtypes.IBCChannel{
    // ...other fields...
    ConnectionID: ch.ConnectionHops[0], // Only using first hop
})

// In Channel handler:
channel = &wasmvmtypes.IBCChannel{
    // ...other fields...
    ConnectionID: got.ConnectionHops[0], // Only using first hop
}
```

Tool Used

Manual Review

Recommendation

Return error for multi-hop channels.

Discussion

chipshort

This is clearly documented in the cosmwasm-std documentation.

Issue L-10: Incorrect error in ToWasmVMGas

Source: https://github.com/sherlock-audit/2025-02-interchain-labs-cosmwasm-v2-audit/issues/546

Summary

The `ToWasmVMGas` method in the gas register implementation incorrectly uses `ErrorOutOfGas` when gas consumption exceeds the limit, but it should use `ErrorGasOverflow` instead. This can lead to incorrect error handling, particularly when the gas limit is consumed on OOG error in the sub-message flow.

Vulnerability Detail

In the `ToWasmVMGas` method of the gas register, the following logic is used:

```
if gas > math.MaxUint64 {
    panic(storetypes.ErrorOutOfGas{Descriptor: "overflow"})
}
```

Code Snippet

Tool Used

Manual Review

Recommendation

Replace `ErrorOutOfGas` with `ErrorGasOverflow` in the `ToWasmVMGas` method:

Discussion

chipshort

Valid point. I doubt this is a big problem in practice since it would require an insane amount of gas. I opened a PR here: https://github.com/CosmWasm/wasmd/pull/2195

Disclaimers

Blackthorn does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.