/cosmos-vulnerability-scanner

Source: `~/.claude/skills/tob-building-secure-contracts/skills/cosmos-vulnerability-scanner/SKILL.md`

---

name: cosmos-vulnerability-scanner
description: Scans Cosmos SDK blockchains for 9 consensus-critical vulnerabilities including non-determinism, incorrect signers, ABCI panics, and rounding errors. Use when auditing Cosmos chains or CosmWasm contracts.

Cosmos Vulnerability Scanner

1. Purpose

Systematically scan Cosmos SDK blockchain modules and CosmWasm smart contracts for platform-specific security vulnerabilities that can cause chain halts, consensus failures, or fund loss. This skill encodes 9 critical vulnerability patterns unique to Cosmos-based chains.

2. When to Use This Skill

- Auditing Cosmos SDK modules (custom x/ modules)
- Reviewing CosmWasm smart contracts (Rust)
- Pre-launch security assessment of Cosmos chains
- Investigating chain halt incidents
- Validating consensus-critical code changes
- Reviewing ABCI method implementations

3. Platform Detection

File Extensions & Indicators

- **Go files**: `.go`, `.proto`
- **CosmWasm**: `.rs` (Rust with cosmwasm imports)

Language/Framework Markers

```
// Cosmos SDK indicators
import (
    "github.com/cosmos/cosmos-sdk/types"
    sdk "github.com/cosmos/cosmos-sdk/types"
    "github.com/cosmos/cosmos-sdk/x/..."
)

// Common patterns
keeper.Keeper
sdk.Msg, GetSigners()
BeginBlocker, EndBlocker
CheckTx, DeliverTx
protobuf service definitions
```

```
// CosmWasm indicators
use cosmwasm_std::*;
```

```
#[entry_point]
pub fn execute(deps: DepsMut, env: Env, info: MessageInfo, msg: ExecuteMsg)
```

Project Structure

- `x/modulename/` - Custom modules
- `keeper/keeper.go` - State management
- `types/msgs.go` - Message definitions
- `abci.go` - BeginBlocker/EndBlocker
- `handler.go` - Message handlers (legacy)

Tool Support

- **CodeQL**: Custom rules for non-determinism and panics
- **go vet**, **golangci-lint**: Basic Go static analysis
- **Manual review**: Critical for consensus issues

---

4. How This Skill Works

When invoked, I will:

1. **Search your codebase** for Cosmos SDK modules
2. **Analyze each module** for the 9 vulnerability patterns
3. **Report findings** with file references and severity
4. **Provide fixes** for each identified issue
5. **Check message handlers** for validation issues

---

5. Example Output

When vulnerabilities are found, you'll get a report like this:

```
=== COSMOS SDK VULNERABILITY SCAN RESULTS ===

Project: my-cosmos-chain
Files Scanned: 6 (.go)
Vulnerabilities Found: 2
```

[CRITICAL] Incorrect GetSigners()

---

5. Vulnerability Patterns (9 Patterns)

I check for 9 critical vulnerability patterns unique to CosmWasm. For detailed detection patterns, code examples, mitigations, and testing strategies, see [VULNERABILITY_PATTERNS.md](resources/VULNERABILITY_PATTERNS.md).

Pattern Summary:

1. **Missing Denom Validation** ⚠ CRITICAL - Accepting arbitrary token denoms
2. **Insufficient Authorization** ⚠ CRITICAL - Missing sender/admin validation
3. **Missing Balance Check** ⚠ HIGH - Not verifying sufficient balances
4. **Improper Reply Handling** ⚠ HIGH - Unsafe submessage reply processing
5. **Missing Reply ID Check** ⚠ MEDIUM - Not validating reply IDs
6. **Improper IBC Packet Validation** ⚠ CRITICAL - Unvalidated IBC packets
7. **Unvalidated Execute Message** ⚠ HIGH - Missing message validation
8. **Integer Overflow** ⚠ HIGH - Unchecked arithmetic operations
9. **Reentrancy via Submessages** ⚠ MEDIUM - State changes before submessages

For complete vulnerability patterns with code examples, see [VULNERABILITY_PATTERNS.md](resources/VULNERABILITY_PATTERNS.md).
5. Scanning Workflow

Step 1: Platform Identification
1. Identify Cosmos SDK version (`go.mod`)
2. Locate custom modules (`x/*/`)
3. Find ABCI methods (`abci.go`, BeginBlocker, EndBlocker)
4. Identify message types (`types/msgs.go`, `.proto`)

Step 2: Critical Path Analysis
Focus on consensus-critical code:
- BeginBlocker / EndBlocker implementations
- Message handlers (execute, DeliverTx)
- Keeper methods that modify state
- CheckTx priority logic

````
### Step 3: Non-Determinism Sweep
**This is the highest priority check for Cosmos chains.**

```bash
# Search for non-deterministic patterns
grep -r "range.*map\[" x/
grep -r "\bint\b\|\buint\b" x/ | grep -v "int32\|int64\|uint32\|uint64"
grep -r "float32\|float64" x/
grep -r "go func\|go routine" x/
grep -r "select {" x/
grep -r "time.Now()" x/
grep -r "rand\." x/
````

For each finding:

1. Verify it's in consensus-critical path
2. Confirm it causes non-determinism
3. Assess severity (chain halt vs data inconsistency)

Step 4: ABCI Method Analysis

Review BeginBlocker and EndBlocker:

- ☐ Computational complexity bounded?
- ☐ No unbounded iterations?
- ☐ No nested loops over large collections?
- ☐ Panic-prone operations validated?
- ☐ Benchmarked with maximum state?

Step 5: Message Validation

For each message type:

- ☐ GetSigners() address matches handler usage?
- ☐ All error returns checked?
- ☐ Priority set in CheckTx if critical?
- ☐ Handler registered (or using v0.47+ auto-registration)?
````

Step 6: Arithmetic & Bookkeeping

- [ ] sdk.Dec operations use multiply-before-divide?
- [ ] Rounding favors protocol over users?
- [ ] Custom bookkeeping synchronized with x/bank?
- [ ] Invariant checks in place?

---

6. Reporting Format

Finding Template

```
## [CRITICAL] Non-Deterministic Map Iteration in EndBlocker

**Location**: `x/dex/abci.go:45-52`

**Description**:
The EndBlocker iterates over an unordered map to distribute rewards, causing different
validators to process users in different orders and produce different state roots. This will
halt the chain when validators fail to reach consensus.

**Vulnerable Code**:
```go
// abci.go, line 45
func EndBlocker(ctx sdk.Context, k keeper.Keeper) {
    rewards := k.GetPendingRewards(ctx)  // Returns map[string]sdk.Coins
    for user, amount := range rewards {  // NON-DETERMINISTIC ORDER
        k.bankKeeper.SendCoins(ctx, moduleAcc, user, amount)
    }
}
```

Attack Scenario:

1. Multiple users have pending rewards
2. Different validators iterate in different orders due to map randomization
3. If any reward distribution fails mid-iteration, state diverges
4. Validators produce different app hashes
5. Chain halts - cannot reach consensus
```

Recommendation: Sort map keys before iteration:

```
func EndBlocker(ctx sdk.Context, k keeper.Keeper) {
    rewards := k.GetPendingRewards(ctx)

    // Collect and sort keys for deterministic iteration
    users := make([]string, 0, len(rewards))
    for user := range rewards {
        users = append(users, user)
    }
    sort.Strings(users)  // Deterministic order

    // Process in sorted order
    for _, user := range users {
        k.bankKeeper.SendCoins(ctx, moduleAcc, user, rewards[user])
    }
}
```

References:

- building-secure-contracts/not-so-smart-contracts/cosmos/non_determinism
- Cosmos SDK docs: Determinism

```

---

## 7. Priority Guidelines

### Critical - CHAIN HALT Risk
- Non-determinism (any form)
- ABCI method panics
- Slow ABCI methods
- Incorrect GetSigners (allows unauthorized actions)

### High - Fund Loss Risk
- Missing error handling (bankKeeper.SendCoins)
- Broken bookkeeping (accounting mismatch)
- Missing message priority (oracle/emergency messages)

### Medium - Logic/DoS Risk
```

```
- Rounding errors (protocol value leakage)
- Unregistered message handlers (functionality broken)

---

## 8. Testing Recommendations

### Non-Determinism Testing
```bash
# Build for different architectures
GOARCH=amd64 go build
GOARCH=arm64 go build

# Run same operations, compare state roots
# Must be identical across architectures

# Fuzz test with concurrent operations
go test -fuzz=FuzzEndBlocker -parallel=10
```

ABCI Benchmarking

```
func BenchmarkBeginBlocker(b *testing.B) {
    ctx := setupMaximalState()  // Worst-case state
    b.ResetTimer()

    for i := 0; i < b.N; i++ {
        BeginBlocker(ctx, keeper)
    }

    // Must complete in < 1 second
    require.Less(b, b.Elapsed()/time.Duration(b.N), time.Second)
}
```

Invariant Testing

```
// Run invariants in integration tests
func TestInvariants(t *testing.T) {
    app := setupApp()
```
```

```
    // Execute operations
    app.DeliverTx(...)

    // Check invariants
    _, broken := keeper.AllInvariants()(app.Ctx)
    require.False(t, broken, "invariant violation detected")
}
```

---

9. Additional Resources

- **Building Secure Contracts**: `building-secure-contracts/not-so-smart-contracts/cosmos/`
- **Cosmos SDK Docs**: https://docs.cosmos.network/
- **CodeQL for Go**: https://codeql.github.com/docs/codeql-language-guides/codeql-for-go/
- **Cosmos Security Best Practices**: https://github.com/cosmos/cosmos-sdk/blob/main/docs/docs/learn/advanced/17-determinism.md

---

10. Quick Reference Checklist

Before completing Cosmos chain audit:

Non-Determinism (CRITICAL):

- ☐ No map iteration in consensus code
- ☐ No platform-dependent types (int, uint, float)
- ☐ No goroutines in message handlers/ABCI
- ☐ No select statements with multiple channels
- ☐ No rand, time.Now(), memory addresses
- ☐ All serialization is deterministic

ABCI Methods (CRITICAL):

- ☐ BeginBlocker/EndBlocker computationally bounded
- ☐ No unbounded iterations
- ☐ No nested loops over large collections
- ☐ All panic-prone operations validated
- ☐ Benchmarked with maximum state

Message Handling (HIGH):

- ☐ GetSigners() matches handler address usage
- ☐ All error returns checked
- ☐ Critical messages prioritized in CheckTx
- ☐ All message types registered

Arithmetic & Accounting (MEDIUM):

- ☐ Multiply before divide pattern used
- ☐ Rounding favors protocol
- ☐ Custom bookkeeping synced with x/bank
- ☐ Invariant checks implemented

Testing:

- ☐ Cross-architecture builds tested
- ☐ ABCI methods benchmarked
- ☐ Invariants checked in CI
- ☐ Integration tests cover all messages

Revision #4
Created 2026-02-18 08:40:03 UTC by John
Updated 2026-05-31 20:01:39 UTC by John