Omni

Omni Undelegation Review

Security Assessment Report

Version: 1.0

May, 2025

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Omni components in scope. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the components in scope. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Omni components contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Omni components in scope.

Overview

Omni is a chain abstraction layer, allowing developers to make their application available to users and liquidity across all rollups, while only deploying their application on a single rollup.

The Undelegation feature in Omni establishes a bridge between EVM execution and Cosmos consensus, allowing users to unstake OMNI tokens. This system operates as a FIFO queue, ensuring zero-balance invariants are maintained in Cosmos accounts. It supports multiple withdrawal triggers, such as reward collection and undelegation, and enforces appropriate unbonding periods.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the omni-network/omni repository.

The scope of this time-boxed review was strictly limited to files at:

1. EVM Withdrawals: Pull 3676.
2. Relax block validation for upgrades: Pull 3692.
3. Undelegations: Pull 3702.
4. Rewards withdrawals: Pull 3800.

Note: third party libraries and dependencies were excluded from the scope of this assessment.

Approach

The security assessment covered components written in Golang.

For the Golang components, the manual review focused on identifying issues associated with the business logic implementation of the libraries and modules. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Go runtime.

Additionally, the manual review process focused on identifying vulnerabilities related to known Golang anti-patterns and attack vectors, such as integer overflow, floating point underflow, deadlocking, race conditions, memory and CPU exhaustion attacks, and various panic scenarios including nil pointer dereferences, index out of bounds, and explicit panic calls.

To support the Golang components of the review, the testing team may use the following automated testing tools:

- golangci-lint: `https://golangci-lint.run/`
- vet: `https://pkg.go.dev/cmd/vet`
- errcheck: `https://github.com/kisielk/errcheck`

Output for these automated tools is available upon request.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 7 issues during this assessment. Categorised by their severity:

- Medium: 1 issue.
- Low: 2 issues.
- Informational: 4 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Omni components in scope. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the components, including optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
OMW-01	`nil` Pointer Dereference In `SendCoinsFromModuleToAccount()`	Medium	Open
OMW-02	`cosmos-sdk` Function `BurnCoins()` Panics On Incorrect Permissions	Low	Open
OMW-03	Hardcoded Minimum Self Delegation In Validator Creation	Low	Open
OMW-04	`nil` Pointer Dereference In `EventProcessor.deliverWithdraw()`	Informational	Open
OMW-05	Fuzz Testing Code Present	Informational	Open
OMW-06	Missing Constructor With `_disableInitializers()` In Upgradeable Contract	Informational	Open
OMW-07	Miscellaneous General Comments	Informational	Open

OMW-01	`nil` Pointer Dereference In `SendCoinsFromModuleToAccount()`		
Asset	`halo/withdraw/wrapper.go`		
Status	Open		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

During validation of the `coins` argument in `SendCoinsFromModuleToAccount()`, a call to `coins.IsValid()` could cause a `nil` pointer dereference.

```
if w.EVMEngineKeeper == nil {
    return errors.New("nil EVMEngineKeeper [BUG]")
} else if !coins.IsValid() { // This ensures amounts are positive
    return errors.New("invalid coins [BUG]")
} else if len(coins) != 1 {
    return errors.New("invalid number of coins, only 1 supported [BUG]")
} else if coins[0].Denom != sdk.DefaultBondDenom {
    return errors.New("invalid coin denom, only bond denom supported [BUG]")
}
```

The `nil` pointer dereference occurs deep within the call stack:

- `coins.IsValid()`
- `coins.Validate()`
- `coins.IsPositive()`
- `coin.Amount.Sign()`

Here, `coin.Amount` is a `math.Int` that contains the member `i *big.Int`. Ultimately, the call to `coin.Amount.i.Sign()` may trigger an unexpected panic if the pointer is `nil`.

This issue is rated high impact because it would cause an unexpected `panic` in a user facing function. However, to reach the panic `UndelegateCoinsFromModuleToAccount()` must be called by CosmosSDK with a coin including a `nil` pointer in the `Amount`. This is not expected to occur within CosmosSDK and so the likelihood is considered low.

Recommendations

Consider pushing an update to the CosmosSDK to ensure that the `coins.IsValid()` checks for `nil` pointers before deferencing the underlying value in `Amount`.

Alternatively, consider using the `IsNil()` on the `coin.Amount` to check for a `nil` pointer before calling `IsValid()`.

```go
if w.EVMEngineKeeper == nil {
    return errors.New("nil EVMEngineKeeper [BUG]")
} else if len(coins) != 1 {
    return errors.New("invalid number of coins, only 1 supported [BUG]")
} else if coins[0].Amount.IsNil() { // @audit prevents amount from being nil
    return errors.New("nil coins [BUG]")
} else if !coins.IsValid() { // This ensures amounts are positive
    return errors.New("invalid coins [BUG]")
} else if coins[0].Denom != sdk.DefaultBondDenom {
    return errors.New("invalid coin denom, only bond denom supported [BUG]")
}
```

OMW-02	`cosmos-sdk` Function `BurnCoins()` Panics On Incorrect Permissions		
Asset	`/halo/withdraw/wrapper.go`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

In the function `SendCoinsFromModuleToAccount()`, the call to `BurnCoins()` will panic if `senderModule` argument does not have the `Burn` permission. This could cause an unexpected coin transfer failure.

Recommendations

Check if `senderModule` exists and has adequate permissions before calling `BurnCoins()`.

OMW-03	Hardcoded Minimum Self Delegation In Validator Creation		
Asset	`evmstaking/keeper/keeper.go`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The EVM staking module hardcodes the minimum self delegation amount to `1` token unit, potentially allowing validators to operate with minimal stake. This appears to be non-production code based on the comment.

```
math.NewInt(1)) // Stub out minimum self delegation for now, just use 1.
```

A proper minimum self delegation threshold helps ensure validators are economically incentivized to act honestly. The current implementation does not have any configurable threshold, potentially breaking economic incentives.

Recommendations

Replace the hardcoded value with a configurable parameter:

- Add a minimum self delegation parameter to the module's parameters, which can be set through governance.
- Modify the `deliverCreateValidator()` function to use this parameter instead of the hardcoded value.

OMW-04	`nil` Pointer Dereference In `EventProcessor.deliverWithdraw()`
Asset	`/halo/evmdistribution/evmdistribution.go`
Status	**Open**
Rating	Informational

Description

The `EventProcessor.Deliver()` does not check if the event parsed from log is `nil` before calling `deliverWithdraw()`. This issue has been given an informational severity, since the event is verified before the function is called in `/octane/evmengine/ke`
and is executed inside a `catch()` function.

Recommendations

Implement the check inside the function as well to prevent future misuse.

OMW-05	Fuzz Testing Code Present
Asset	`evmengine/keeper/abci.go`
Status	**Open**
Rating	Informational

Description

The Octane module's `PrepareProposal()` function includes the `maybeFuzzPayload` functionality which, when explicitly enabled via the `FlagFuzzOctane` feature flag, introduces random mutations to execution payloads. These mutations include altering parent hashes, adding invalid withdrawals, and creating invalid blob bundles.

This function is guarded by a feature flag check that prevents execution in production environments. The current implementation returns the unmodified payload when the flag is disabled, ensuring normal operation in production.

However, test code should typically be separated from production code to improve maintainability and reduce the risk of configuration errors.

Recommendations

Consider refactoring the test functionality into a separate testing package to follow best practices.

OMW-06	Missing Constructor With `_disableInitializers()` In Upgradeable Contract
Asset	`Distribution.sol`
Status	**Open**
Rating	Informational

Description

The `Distribution` contract is designed to be upgradeable, but it does not include a constructor that invokes `_disableInitializers()`. While not necessarily problematic if deployment scripts are correctly configured, this omission deviates from OpenZeppelin's recommended best practices for upgradeable contracts.

Recommendations

Add a constructor to the `Distribution` contract that calls `_disableInitializers()` to prevent the implementation contract from being initialized directly:

```
constructor() {
    _disableInitializers();
}
```

This change aligns with OpenZeppelin's recommended pattern for upgradeable contracts and adds an additional layer of security that does not rely on external deployment processes.

OMW-07	Miscellaneous General Comments
Asset	All files
Status	Open
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Unchecked Pointer Arguments To Functions**

   Related Asset(s): /halo/evmengine/keeper/msg_server.go, /halo/evmengine/keeper/abci.go

   Pointer arguments to functions in `msg_server.go` and `abci.go` lack `nil` checks. For example in `ExecutionPayload()`, `msg` is not verified. Similarly, in `abci.go`, the `req` argument of `PrepareProposal()` is not validated.

   Due to how handlers currently work in Omni, these pointers are expected to be non-`nil`.

   Implement checks and return errors in cases where the arguments could be `nil` pointers.

2. **Misleading Comment**

   Related Asset(s): evmengine/keeper/keeper.go

   The following comment is misleading:

   ```
   // withdrawalsEqual returns true only if two non-nil slices contain identical
   // withdrawals in identical order.
   func withdrawalsEqual(w1, w2 []*etypes.Withdrawal) bool {
   ```

   Since this function will return true if both slices are `nil`.

   Consider rewording to:

   ```
   // withdrawalsEqual returns true if both slices have the same length and
   // contain identical withdrawals in identical order. Both nil and empty slices
   // are considered equal. Returns false if either slice contains nil elements.
   ```

3. **Error Handling**

   Related Asset(s): evmengine/keeper/keeper.go

   Error is returned directly which is inconsistent with the error handling pattern used in other parts of the code.

   ```
   headHash, err := head.Hash()
      if err != nil {
          return engine.ExecutableData{}, err
      }
   ```

   To maintain consistency with the coding pattern in the rest of the file, wrap errors using `errors.Wrap()` to provide additional context.

4. **Missing Input Validation In `updateExecutionHead()` Function**

   Related Asset(s): evmengine/keeper/db.go

   The `updateExecutionHead()` function is responsible for updating the execution head record with data from a new executable payload.

Validate that the block hash is not a zero hash, as is done in `InsertGenesisHead`:

```
if payload.BlockHash == (common.Hash{}) {
    return errors.New("invalid zero execution block hash")
}
```

5. **Open `TODO`**

   Related Asset(s): evmengine/keeper/abci.go

   There is an unaddressed open `TODO`

   ```
   // TODO(corver): Add support for typed errors.
   ```

6. **Excess Fee Burned**

   Related Asset(s): Distribution.sol

   When a user calls the withdraw function with a `msg.value` greater than the required fee (0.1 ether), the entire amount is burned.

   Consider modifying the `_burnFee()` function to only burn the exact fee amount and return any excess to the caller.

7. **Configurable Fee In Distribution Contract**

   Related Asset(s): Distribution.sol

   In `Distribution.sol`, the withdrawal fee is implemented as a constant (`uint256 public constant Fee = 0.1 ether`), this lacks flexibility to adapt to changing economic conditions or network usage patterns.

   Consider making the fee configurable.

8. **Legacy Zero Used**

   Related Asset(s): evmstaking/keeper/keeper.go

   `NewMsgCreateValidator` is called with zero as the commission.

   ```
   stypes.NewCommissionRates(math.LegacyZeroDec(), math.LegacyZeroDec(), math.LegacyZeroDec())
   ```

   Consider if this should be modified for production.

9. **Dust Is Burnt**

   Related Asset(s): withdraw/wrapper.go

   Dust (under 1 gwei) is added to counter and not withdrawn, this should be documented clearly as it would be uneconomical for users to withdraw small amounts.

   ```
     if err != nil {
     return errors.Wrap(err, "to gwei conversion")
   }
   dustCounter.Add(float64(dust))
   ```

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact			
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Low	Low	Medium
	Low	Medium	High
		Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.