Omni Halo Security Review

Auditors

Dtheo, Lead Security Researcher

Justin Traglia, Lead Security Researcher

Shotes, Lead Security Researcher

Report prepared by: Lucas Goiriz

October 10, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Omni is the platform for building chain abstracted applications. By linking into each rollup, developers can source liquidity and users from the entire Ethereum ecosystem.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Omni Halo according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 14 days in total, Omni engaged with Spearbit to review the omni-halo protocol. In this period of time a total of **32** issues were found.

Summary

Project Name	Omni
Repository	omni-halo
Commit	fd388534
Type of Project	Infrastructure
Audit Timeline	Aug 26th to Oct 4th

Note that the researchers worked part-time on this review during the specified timeline.

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	5	5	0
Low Risk	9	8	1
Gas Optimizations	3	1	2
Informational	15	10	5
Total	**32**	**24**	**8**

5 Findings

5.1 Medium Risk

5.1.1 CoinGecko API sanitization exposes DoS risks

Severity: Medium Risk

Context: coingecko.go#L59

Description: `(c Client) getPrice()` and its calling functions do not account for issues or inconsistencies in Coingecko's token price API making the fee oracle process vulnerable to multiple forms of unsafe API data handling. This data source is used downstream as the canonical data source for omni's token price oracle contracts which exposes the Omni network to the external centralization risks of which Coingecko is subject to.

- A divide-by-zero panic in lib/contracts/feeoraclev1/feeparams.go#L109 if the `to` price returned by Coingecko is 0. This panic is not caught by any `recover()` function in the call stack and will bring the fee oracle process down.
- A custom panic in lib/tokens/coingecko/coingecko.go#L65) if Coingecko returns anything other than "omni-network" or "ethereum", including an empty string. This panic will also cause a DoS of the fee oracle process.
- If Coingecko returns a valid but empty json there will be no error in this function and multiple levels of calling functions will also not catch the error. This can result in stale prices without there being any indication of error and potentially an nil price on initialization.

Recommendation: Thoroughly sanitize data from external sources. Furthermore, it is recommend wrapping external API call stacks with `recover()` will prevent entire node DoS if panics happen on untrusted data.

Omni: Fixed in 8ba74963 as recommended.

Spearbit: Fix verified.

5.1.2 `guageBuffered()` contains unsafe read of `b.buffer`

Severity: Medium Risk

Context: buffer.go#L121-L136

Description: `b.buffer` is passed to `guageBuffered()` which references the buffer as `prices`. Access of this buffer should be protected by a mutex, in this case a read lock. This is called periodically in the `(b *Buffer) stream` ticker cadence. This opens up Omni's fee oracle to memory corruption.

Recommendation: Create a thread safe `guageBuffered()` method.

Omni: Fixed in 8cc196bb as recommended.

Spearbit: Fix verified.

5.1.3 `cache::Get()` unsafely returns a reference to `c.network` without a lock

Severity: Medium Risk

Context: cache.go#L34

Description: `(c *cache) Get()` returns a pointer to `c.network` and releases its lock upon the function return. This function is called periodically by `(k Keeper) getLatestPortals()` which will deference the pointer when it calls `network.GetPortals()`. This makes the node vulnerable to race conditions and thus memory corruption.

Recommendation: Return a copy of the network object or create thread safe getter routines for the `cache` object.

Omni: Fixed in 60904774 as recommended.

Spearbit: Fix verified.

5.1.4 Potential double-vote if validator runs out of storage

Severity: Medium Risk

Context: voter.go#L316-L328

Description: In `Voter::Vote()`, if `v.saveUnsafe()` (which saves the voting state to disk) returns an error because the node is out of storage, the changes to `v.latest` and `v.available` will still persist. Since the vote was created, we believe it will still be propagated, but the updated state will not be saved to disk. In this situation, there is potential for a double-vote when the node restarts.

When the node starts back up, it will load the outdated state. It calls `getFromHeightAndOffset()` to get `skipBeforeOffset`. Because the state on disk is expected to be out-of-date, `skipBeforeOffset` will come from the latest "approved" attestation. We believe there is an opportunity for the latest propagated attestation to be newer than the latest approved on-chain attestation; otherwise, why would it compare against the value from the local state?

```
// WriteFileAtomic creates a temporary file with data and provided perm and
// swaps it atomically with filename if successful.
func WriteFileAtomic(filename string, data []byte, perm os.FileMode) (err error) {
```

The `v.safeUnsafe()` function uses `WriteFileAtomic()` to save the state to disk. `WriteFileAtomic()` will attempt to create a temporary file. Since there is no available storage space, the write operation will fail when trying to write to the temporary file. This failure will result in an error being returned by the function and the original file will remain unchanged.

Recommendation: If saving the state to disk fails, stop the validator instead of restarting the stream.

Omni: Fixed in 519e3342 by aborting the application if persistence fails.

Spearbit: Fix verified.

5.1.5 `insertValidatorSet()` can insert an invalid validator into `valTable`

Severity: Medium Risk

Context: keeper.go#L235-L250

Description: This might be an impossible situation (still need to confirm this) but `valsync::Keeper::insertValidatorSet()` will insert a validator with an invalid power (negative) or invalid public key, then return an error without removing the validator. This could theoretically happen if `maybeStoreValidatorUpdates` were given a bad update entry.

Proof of Concept: For example, I made this test to show that validators with negative powers will still exist in the table after returning an error.

```go
func TestInsertValidatorWithNegativePower(t *testing.T) {
    keeper, sdkCtx := setupKeeper(t, defaultExpectation())

    var set []*Validator
    for pubkey := 1; pubkey <= 3; pubkey++ {
        var pk [32]byte
        pk[0] = byte(pubkey)
        priv, err := crypto.ToECDSA(pk[:])
        require.NoError(t, err)

        set = append(set, &Validator{
            PubKey: crypto.CompressPubkey(&priv.PublicKey),
            Power:  -1,
        })
    }

    sdkCtx = sdkCtx.WithBlockHeight(0)
    _, err := keeper.insertValidatorSet(sdkCtx, clone(set), true)
    require.Error(t, err, "negative power")
    setIter, err := keeper.valsetTable.List(sdkCtx, ValidatorSetPrimaryKey{})
    require.NoError(t, err)
    for setIter.Next() {
        valset, err := setIter.Value()
        require.NoError(t, err)
        valIter, err := keeper.valTable.List(sdkCtx, ValidatorValsetIdIndexKey{}.WithValsetId(valset.GetId()))
        require.NoError(t, err)
        for valIter.Next() {
            val, err := valIter.Value()
            require.NoError(t, err)
            require.Positive(t, val.GetPower())
        }
        valIter.Close()
    }
    setIter.Close()
}
```

```
Error Trace:    /Users/user/omni/halo/valsync/keeper/query_internal_test.go:172
Error:          "-1" is not positive%!(EXTRA int64=0)
Test:           TestInsertValidatorWithNegativePower
```

Recommendation: Perform validator checks before inserting the validator into the table.

Omni: Fixed in 75c04cf9 by adding a new `Validator::Validate()` function & tests.

Spearbit: Fix verified.

5.2 Low Risk

5.2.1 `Attestation::Verify()` contains insufficient signature verification

Severity: Low Risk

Context: tx.go#L212-L216

Description: `Attestation.Verify()` does not verify an attestation `Signature` is cryptographically associated with their `ValidatorAddress` or that the signature is made over the attestation root. This allows for any valid signature to bypass this verification method without having a relationship to the data or the validator set. This does not currently pose a significant risk to Omni since the the signatures are checked during the vote aggregation phase in `AggVote.Verify()`. However, future uses of this function could introduce a signature verification bypass issue if developers make the assumption that the function is safe.

Recommendation: Completely verify the signature comes from the correct public key and is signed over the correct vote similar to how it is done in `AggVote.Verify()`.

Omni: Acknowledged. Will fix later.

Spearbit: Acknowledged.

5.2.2 Multiple unsafe type conversions

Severity: Low Risk

Context: tx.go#L124-L125

Description: Omni currently uses multiple unsafe type conversions that can cause the error

```
panic: runtime error: cannot convert slice with length X
```

- `sigTuple.ToXChain()`
- `Attestation.AttestationRoot()`
- `Vote.ToXChain()`
- `SigFromProto()`
- `BlockHeaderFromProto()`

These issues can occur when casting slices into byte arrays with fixed lengths. Currently, there does not appear to be a way to trigger these `panic`s from attacker controlled buffers but this could change in the future.

Recommendation: Safe methods should be created for converting these data types to prevent the possibility of these possibilities of casting related `panic`s.

Omni: Fixed in 07b575ff as recommended.

Spearbit: Fix verified.

5.2.3 `p.instance` accessed outside of mutex protected region

Severity: Low Risk

Context: proxy.go#L169

Description: `(p *proxy).getInstance()` returns a reference `p.instance` without a lock. This getter called in `getInstance()` and the pointer will be immediately dereferenced to call `.String()` without a lock. This allows for accesses outside of the protected region and introduces the possibility of a race condition.

Recommendation: Create a thread safe `Stop()` routine to stop the proxy instance.

Omni: Fixed in 0097ea08f as recommended.

Spearbit: Fix verified.

5.2.4 In `getConsXBlock()`, accessing `xblock.Msgs[0]` could panic

Severity: Low Risk

Context: fetch.go#L436-L443

Description: If the length of `xblock.Msgs` is zero, accessing the first element will cause a panic. After chatting with the team about this, we have deemed it an impossible situation. Omni consensus `xblocks` cannot be empty, only true EVM `xblocks` can be empty. The Omni consensus portal module only create `xblocks` if it has messages, or rather, when some other module wants to create a message, only then is a `xblock` created.

Recommendation: Add another condition to ensure there is at least one message:

```
+ } else if len(xblock.Msgs) == 0 {
+     return xchain.Block{}, errors.New("no messages [BUG]")
  } else if xblock.Msgs[0].DestChainID != 0 {
```

Omni: Fixed in 327a4d7e as recommended.

Spearbit: Fix verified.

5.2.5 `PrepareVotes()` suppresses error from `ValidateVoteExtensions()`

Severity: Low Risk

Context: cpayload.go#L32-L37

Description: If `ValidateVoteExtensions()` errors, the error is not returned; nil is returned. This does not appear to be intentional.

Recommendation: If this is intentional, add a comment stating such. Otherwise change the return statement to:

```
- return nil, nil
+ return nil, err
```

Omni: Fixed in 84dd860f by returning a wrapped error.

Spearbit: Fix verified.

5.2.6 `AttestationRoot()` does not check if input is nil

Severity: Low Risk

Context: tx.go#L183-L185, tx.go#L235-L237

Description: This applies to `AggVote::AttestationRoot()` and `Attestation::AttestationRoot()`. This is not a problem now, but we are making this a finding because they are exposed functions and it could be used improperly in the future. We access fields without ensuring `a` is not nil. This would cause a panic if `a` were nil. Currently, every path to this function ensures the input is not nil.

Recommendation: Out of caution, check if `a` is nil and return an error if it is.

Omni: Fixed in af90dcd1 as recommended.

Spearbit: Fix verified.

5.2.7 Mutable payload identifier is assigned outside of write lock

Severity: Low Risk

Context: abci.go#L58-L71, keeper.go#L206-L211

Description: In `PrepareProposal()`, the `getOptimisticPayload()` method returns a pointer to the payload identifier. When we finally get the payload, we use the pointer to assign the identifier. The mutable payload has a lock and we are writing to it without holding a write lock on it. Out of caution, we should avoid this pattern.

Recommendation: 1. Maybe use `setOptimisticPayload()` to assign the payload ID. 2. Change `ID` in `mutablePayload` to a non-pointer type; this is prone to errors.

Omni: Fixed in 60904774 by changing `ID` to a non-pointer type.

Spearbit: Fix verified.

5.2.8 `Attestation::Verify()` does not check that headers are for the same chain ID

Severity: Low Risk

Context: tx.go#L27-L33

Description: In `Attestation::Verify`, after verifying `BlockHeader` and `AttestHeader`, it would be worthwhile to check that the headers are for the same chain ID. This quick sanity check will prevent possible issues. This is marked as a low because we believe this is check occurs later.

Recommendation: Add the following check to `Attestation::Verify`:

```
if a.AttestHeader.SourceChainId != a.BlockHeader.ChainId {
    return errors.New("mismatching chain id", "block", a.BlockHeader.ChainId, "att",
↪   a.AttestHeader.SourceChainId)
}
```

Omni: Fixed in fb5d785b by adding the recommended check.

Spearbit: Fix verified.

5.2.9 `AttestHeader::Verify()` accepts invalid confirmation level

Severity: Low Risk

Context: tx.go#L71-L73

Description: Here `h.GetConfLevel()` is truncated to a byte. This function could accept an attestation with an invalid confirmation level. For example, `513 & 0xFF == 1`. It is unclear to us how this could be abused.

Recommendation: Out of caution we should add a check to ensure the value fits in a single byte first. Something like:

```
conf := xchain.ConfLevel(h.GetConfLevel())
if uint32(conf) != h.GetConfLevel() || !conf.Valid() {
```

Omni: Fixed in fb5d785b by ensuring confirmation level is max 1 byte.

Spearbit: Fix verified.

5.3 Gas Optimization

5.3.1 Event sorting could be more efficient

Severity: Gas Optimization

Context: evmmsgs.go#L39-L43

Description: This section of code could be slightly better. The current implementation will concatenate all topics together then compare the whole byte slices. It would be better to use `slices.CompareFunc` instead because (1) you avoid two memory allocations and (2) you compare individual elements meaning you can bail on the first difference, which is faster.

Recommendation: Replace this chunk of code with the following:

```
if cmp := slices.CompareFunc(events[i].Topics, events[j].Topics, bytes.Compare); cmp != 0 {
    return cmp < 0
}
```

Omni: Acknowledged. Will fix in the next upgrade.

Spearbit: Acknowledged.

5.3.2 Call to `headerMap()` can be made before lock

Severity: Gas Optimization

Context: voter.go#L407-L417, voter.go#L436-L446

Description: The `headerMap()` method does not need to be executed with a lock. We can move it above the lock.

Recommendation: Move the `headerMap()` call to above the lock.

Omni: Fixed in 898e4bb0 as recommended.

Spearbit: Fix verified.

5.3.3 Read/write locks can be converted read-only locks

Severity: Gas Optimization

Context: keeper.go#L206-L211, voter.go#L124-L131, voter.go#L134-L139, voter.go#L391-L396, voter.go#L399-L404, voter.go#L486-L493

Description:

- halo/attest/voter/voter.go#L124-L131 -- `minWindow` only reads `v.minsByChain`.
- halo/attest/voter/voter.go#L134-L139 -- `isValidator` only reads `v.isVal`.
- halo/attest/voter/voter.go#L391-L396 -- `AvailableCount` only reads `v.available`.
- halo/attest/voter/voter.go#L399-L404 -- `GetAvailable` only reads `v.available`.
- halo/attest/voter/voter.go#L486-L493 -- `latestByChain` only reads `v.latest`.
- octane/evmengine/keeper/keeper.go#L206-L211 -- `getOptimisticPayload` only reads `mutablePayload`.

Recommendation: In these instances, use `RLock()` and `RUnlock()` instead.

Spearbit: Acknowledged.

5.4 Informational

5.4.1 `PrepareProposal()` does not prevent overfilling a block with bytes

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: In CometBFT, the `PrepareProposal()` function sets a limit on the maximum number of bytes that are allowed to fit in a proposed block octane/evmengine/keeper/abci.go#L41. However, Halo is not allowed to remove any transactions from the proposed block if this limit is exceeded. If a Halo block were to exceed this limit, the chain would simply halt.

In practice, this can never happen. The maximum amount of gas in an EVM block is 30,000,000. The largest block possible when constrained by gas is a block filled with 0's. This results in a maximum block of 7,500,000 bytes, or 15,000,000 bytes when hex encoded in the EngineAPI.

Since `cmttypes.MaxBlockSizeBytes*9/10` evaluates to roughly 94 MB, it is impossible for a 15 MB maximum block size to ever exceed this.

Recommendation: Add a comment indicating that the `MaxBlockSizeBytes` limit is impossible to hit due to the EVM gas limit.

Omni: Acknowledged. We will add a comment and ensure we do not encode the block as hex but as binary to reduce our consensus block size.

Spearbit: Acknowledged.

5.4.2 Private keys stored in memory for duration of process

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The private key for a validator in Halo is loaded into memory at the start of the program and remains there for the duration. The current best practices allow support for external key signers and potentially even hardware signers.

Recommendation: Whenever the validator set is un-whitelisted, it would be beneficial to support common web3 key-signers that allow more discreet usages of the private key.

Omni: Acknowledged. This is currently on the roadmap.

Spearbit: Acknowledged.

5.4.3 In `searchOffsetInHistory()`, latency function is not executed

Severity: Informational

Context: abci.go#L366

Description: In `searchOffsetInHistory()`, the function returned by `latency(endpoint)` is not executed. This will prevent the latency from being logged.

Recommendation: Make the following change:

```
- defer latency(endpoint)
+ defer latency(endpoint)()
```

Omni: Fixed in 509ad8ea as recommended.

Spearbit: Fix verified.

5.4.4 `ExtendVote()` should log filtered votes

Severity: Informational

Context: keeper.go#L678-L692

Description: In `ExtendVote()`, after filtering votes, we log *unfiltered* votes when we should log *filtered* votes.

Recommendation: Make the following change:

```
- for _, vote := range votes {
+ for _, vote := range filtered {
```

Omni: Fixed in 7d944686 as recommended.

Spearbit: Fix verified.

5.4.5 Length check in final return statement can be replaced with `true`

Severity: Informational

Context: abci.go#L551-L560

Description: In the final return statement for `attsFromAtHeight()`, it should be impossible for `len(resp.Attestations)` to be zero. We can replace this with `true` for slightly better readability. There is a chance that this was supposed to check `len(atts)` instead, but we think that is unlikely.

Recommendation: Replace the final return statement with the following:

```
- return atts, len(resp.Attestations) != 0, nil
+ return atts, true, nil
```

Or if it was supposed to check `atts` length, use the following:

```
- return atts, len(resp.Attestations) != 0, nil
+ return atts, len(atts) != 0, nil
```

Omni: Fixed in fb5d785b by returning a hardcoded true value.

Spearbit: Fix verified.

5.4.6 `uintSub()` can be replaced with `umath.SubtractOrZero()`

Severity: Informational

Context: keeper.go#L1075-L1083, umath.go#L13-L18

Description: There is another version of this method with (in our opinion) a better name. We suggest using it instead.

Recommendation: Delete `uintSub` and replace calls to it with `umath.SubtractOrZero()`.

Omni: Fixed in 99edcf2e as recommended.

Spearbit: Fix verified.

5.4.7 Unused statistics in `insertValidatorSet()` can be deleted

Severity: Informational

Context: keeper.go#L231-L272

Description: The code and variables associated with `totalUpdated`, `totalLen`, and `totalRemoved` are unused. This appears to have been moved to `setStats` and should be removed from `insertValidatorSet()`.

Recommendation: Remove statistics code/variables from `insertValidatorSet()`.

Omni: Fixed in c5aa4fb1 as recommended.

Spearbit: Fix verified.

5.4.8 Signature checks can be replaced with `SigTuple::Verify()`

Severity: Informational

Context: tx.go#L15-L41

Description: For consistency, we can replace the signature checks in `Vote::Verify()` with `SigTuple::Verify()`.

Recommendation: Make the following changes:

```
  func (v *Vote) Verify() error {
      if v == nil {
          return errors.New("nil attestation")
      }
-     if v.Signature == nil {
-         return errors.New("nil signature")
-     }

      if len(v.MsgRoot) != len(common.Hash{}) {
          return errors.New("invalid message root length")
      }

      if err := v.BlockHeader.Verify(); err != nil {
          return errors.Wrap(err, "verify block header")
      }

      if err := v.AttestHeader.Verify(); err != nil {
          return errors.Wrap(err, "verify attestation header")
      }

-     if len(v.Signature.Signature) != len(xchain.Signature65{}) {
-         return errors.New("invalid signature length")
-     }
-
-     if len(v.Signature.ValidatorAddress) != len(common.Address{}) {
-         return errors.New("invalid validator address length")
-     }
+     if err := v.Signature.Verify(); err != nil {
+         return errors.Wrap(err, "verify signature")
+     }
```

Omni: Fixed in 0a1578ac as recommended.

Spearbit: Fix verified.

5.4.9 Unnecessary nil check can be removed

Severity: Informational

Context: tx.go#L30-L40

Description: The nil check for `l.Address` is unnecessary. The length check below is sufficient because `len(nil)` is zero.

Recommendation:

```
- if l.Address == nil {
-    return errors.New("nil address")
- }
-
  if len(l.Topics) == 0 {
      return errors.New("empty topics")
  }

  if len(l.Address) != len(common.Address{}) {
-      return errors.New("invalid address length")
+      return errors.New("invalid address")
  }
```

Omni: Fixed in 24952374 by removing the unnecessary nil check.

Spearbit: Fix verified.

5.4.10 Validator power warning can happen with less than 1/3 power

Severity: Informational

Context: keeper.go#L259-L269

Description: This could alarm if a validator has less than 1/3 of the total power. This is because the fractional part of `totalPower/3` is truncated. If `totalPower` is 10 and `power` is 3, `power >= totalPower/3` will evaluate to `3 >= 3` (true).

Recommendation: Update the check to handle this edge case.

Omni: Fixed in 22dac87f by checking if power is strictly greater than 1/3.

Spearbit: Fix verified.

5.4.11 Protobuf message identifiers should start with one

Severity: Informational

Context: portal.proto#L10-L15, registry.proto#L10-L14, valsync.proto#L13-L17, valsync.proto#L27-L32

Description:

- halo/portal/keeper/portal.proto#L10-L15 -- the query index id should be 1.
- halo/registry/keeper/registry.proto#L10-L14 -- the option id should be 1.
- halo/valsync/keeper/valsync.proto#L13-L17 -- the query index id should be 1.
- halo/valsync/keeper/valsync.proto#L27-L32 -- the query index id should be 1.

Recommendation: Fix these identifiers.

Omni: Acknowledged. This would require a network upgrade and migration.

Spearbit: Acknowledged.

5.4.12 Attestation structure field order does not match protobuf definition

Severity: Informational

Context: tx.proto#L28-L34, types.go#L201-L208

Description: The order of fields in this type does not match the order in the protobuf type.

Recommendation: Move `ValidatorSetID` to the bottom of the structure:

```
  type Attestation struct {
      AttestHeader               // AttestHeader uniquely identifies the attestation.
      BlockHeader                // BlockHeader identifies the cross-chain Block
-     ValidatorSetID uint64      // Validator set that approved this attestation.
      MsgRoot        common.Hash // Merkle root of all messages in the cross-chain Block
      Signatures     []SigTuple  // Validator signatures and public keys
+     ValidatorSetID uint64      // Validator set that approved this attestation.
  }
```

Omni: Fixed in 45ad070f as recommended.

Spearbit: Fix verified.

5.4.13 Misleading comments

Severity: Informational

Context: evmupgrade.go#L59, keeper.go#L70-L74, portal.proto#L9-L19, tx.proto#L55, types.go#L31-L32

Description:

- halo/evmupgrade/evmupgrade.go#L59 -- "omni stake contract" should be "omni upgrade contract".
- halo/portal/keeper/portal.proto#L9-L19 -- `Block` does not contain an offset field, and comment about `BlockHeight`/`AttestOffset` being auto-incremented are wrong.
- halo/registry/keeper/keeper.go#L70-L74 -- `getOrCreateEpoch` should be `getOrCreateNetwork`.
- halo/attest/types/tx.proto#L55 -- should use singular `signature` and `public key`.

Recommendation: Fix these comments.

Omni: Acknowledged. Will fix later.

Spearbit: Acknowledged.

5.4.14 Typographical issues

Severity: Informational

Context: abci.go#L136, config.go#L37, connect.go#L24, connect.go#L41, evmslashing.go#L47, evmslashing.go#L58, evmupgrade.go#L48, fetch.go#L109, keeper.go#L26, keeper.go#L34, keeper.go#L45, keeper.go#L54, keeper.go#L644, keeper.go#L667, keeper.go#L72, logeventproc.go#L128, query.go#L90, tx.proto#L22, tx.proto#L68, voter.go#L476, voter.go#L497, voter.go#L532, voter.go#L633, xblock.go#L109-L112

Description:

- halo/attest/keeper/keeper.go#L644 -- `limited exceeded` should be `limit exceeded`.
- halo/attest/keeper/keeper.go#L667 -- `no` should be `not`.
- halo/attest/types/tx.proto#L68 -- `belangs` should be `belongs`.
- halo/attest/voter/voter.go#L476 -- `availableAndProposed` should be `availableAndProposedUnsafe`.
- halo/attest/voter/voter.go#L497 -- `atts` should be `votes`.
- halo/attest/voter/voter.go#L532 -- `atts` should be `votes`.
- halo/attest/voter/voter.go#L633 -- `atts` should be `votes`.
- halo/registry/keeper/keeper.go#L26 -- `portalRegAdress` should be `portalRegAddress`.

- halo/registry/keeper/keeper.go#L54 -- `protalReg` should be `portalReg`.
- halo/registry/keeper/logeventproc.go#L128 -- `shads` should be `shards`.
- halo/valsync/keeper/keeper.go#L41 -- `emilPortal` should be `emitPortal`.
- halo/valsync/keeper/keeper.go#L45 -- `portalRegAdress` should be `portalRegAddress`.
- halo/valsync/keeper/keeper.go#L72 -- `protalReg` should be `portalReg`.
- halo/valsync/keeper/query.go#L90 -- `vatset` should be `valset`.
- lib/cchain/provider/abci.go#L136 -- `actived Height` should be `activated Height`.
- lib/cchain/provider/xblock.go#L111 -- `pack validators` should be `pack networks`.
- lib/xchain/connect/connect.go#L24 -- `provider` should be `provides`.
- lib/xchain/connect/connect.go#L41 -- `confired` should be `configured`.
- octane/evmengine/types/tx.proto#L22 -- `the  next` should be `the next`.
- halo/evmupgrade/evmupgrade.go#L48 -- `new staking` should be `new upgrade`.
- halo/evmslashing/evmslashing.go#L47 -- `new staking` should be `new slashing`.
- halo/evmslashing/evmslashing.go#L58 -- `stake contract` should be `unjail contract`.
- lib/xchain/provider/fetch.go#L109 -- `OutXMgsOffset` should be `OutXMsgOffset`.
- halo/config/config.go#L37 -- `Cosmsos` should be `Cosmos`.

Recommendation: Fix these typos.

Omni: Acknowledged. Will fix later.

Spearbit: Acknowledged.

5.4.15 `*valAddrCache.SetAll()` mutex use can be optimized

Severity: Informational

Context: valaddr.go#L33

Description: `valAddrCache` contains a `RWMutex` to guard concurrent use of the `ethAddrs` map:

```
type valAddrCache struct {
    sync.RWMutex
    ethAddrs map[[crypto.AddressSize]byte]common.Address
}
```

`valAddrCache.SetAll()` is a method used to create the map and requires a write lock to do so. Currently the lock is held while the new map is being created which is not optimal. Since the new map is created in a separate map object and then copied at the end of the function, the write lock can be moved down to the end of the function where `c.ethAddrs` is assigned.

Recommendation:

```
  func (c *valAddrCache) SetAll(vals []*vtypes.Validator) error {
-     c.Lock()
-     defer c.Unlock()

      var ethAddrs = make(map[[crypto.AddressSize]byte]common.Address, len(vals))
      for _, val := range vals {
          cmtAddr, err := val.CometAddress()
          if err != nil {
              return err
          } else if len(cmtAddr) != crypto.AddressSize {
              return errors.New("invalid comet address length [BUG]", "len", len(cmtAddr))
          }

          ethAddr, err := val.EthereumAddress()
          if err != nil {
              return err
          }

          ethAddrs[[crypto.AddressSize]byte(cmtAddr)] = ethAddr
      }

+     c.Lock()
      c.ethAddrs = ethAddrs
+     c.Unlock()

      return nil
  }
```

Omni: Fixed in fb5d785b by moving the lock to the recommended location.

Spearbit: Fix verified.