Berachain Beaconkit Security Review

Auditors

Shotes, Lead Security Researcher

Guido Vranken, Lead Security Researcher

Hack3r0m, Security Researcher

0xDeadbeef, Associate Security Researcher

Report prepared by: Lucas Goiriz

February 1, 2025

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

Berachain is an EVM-identical L1 turning liquidity into security powered by Proof Of Liquidity.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of beacon-kit 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 25 days in total, Berachain engaged with Spearbit to review the beacon-kit protocol. In this period of time a total of **22** issues were found.

Summary

Project Name	Berachain
Repository	beacon-kit
Commit	cafd14...efc1
Type of Project	Infrastructure, Node
Audit Timeline	Nov 11th to Dec 9th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	2	2	0
Medium Risk	6	4	2
Low Risk	4	3	1
Gas Optimizations	0	0	0
Informational	10	4	6
Total	**22**	**13**	**9**

5 Findings

5.1 High Risk

5.1.1 Invalid signatures from `Deposit` event can halt chain

Severity: High Risk

Context: mod/state-transition/pkg/core/state_processor_staking.go#L176-L186

Description: During state transition, the state transition function will error out when any `Deposit` fails signature verification. Anytime a deposit with an invalid signature is submitted, this will result in an honest proposer becoming unable to propose a block without erroring out:

```
if err = dep.VerifySignature(
    d.New(
        version.FromUint32[common.Version](
            sp.cs.ActiveForkVersionForEpoch(epoch),
        ), genesisValidatorsRoot,
    ),
    sp.cs.DomainTypeDeposit(),
    sp.signer.VerifySignature,
); err != nil {
    return err // AUDIT: returns error instead of ignoring
}
```

Here is the chain of events in this scenario:

1. A `Deposit` transaction with an invalid signature gets submitted to the execution layer.
2. After that block is finalized, the `Deposits` are extracted from the execution layer.
3. These `Deposits` are only validated in byte array length.
4. These `Deposits` get saved in the "*deposit store*".
5. A proposer will craft the new `BeaconBlock` to propose. The deposits in this `BeaconBlock` are pulled directly from the "deposit store" that contains finalized deposits.
6. The proposer will run the state transition function to ensure it is about to propose a valid `BeaconBlock`. This will fail due to the invalid signature in the `Deposit`.
7. Each next proposer will be unable to propose a block due to the same reason.

This is `High` severity and not `Critical` because this can be resolved with a client update without revising any chain history.

Recommendation: When verifying the signature of a deposit during the state transition, skip over invalid deposits instead of returning an error. This is the behavior that corresponds to Ethereum Consensus Specs in the `apply_deposit()` function.

Here is an example of the recommended change:

```
// Verify that the message was signed correctly.
var d ForkDataT
if err = dep.VerifySignature(
    d.New(
        version.FromUint32[common.Version](
            sp.cs.ActiveForkVersionForEpoch(epoch),
        ), genesisValidatorsRoot,
    ),
    sp.cs.DomainTypeDeposit(),
    sp.signer.VerifySignature,
); err != nil {
    // Ignore deposits that fail the signature check.
    return nil
}

// Add the validator to the registry.
return sp.addValidatorToRegistry(st, dep)
```

Berachain: Fixed in commit 2fdcb1d.

Spearbit: Fix verified.

5.1.2 `mod/state-transition/pkg/core/state/ExpectedWithdrawals` returns error if non-0x01 withdrawal credential is present

Severity: High Risk

Context: mod/state-transition/pkg/core/state/statedb.go#L264-L268

Description: `ExpectedWithdrawals` is not in line with the reference implementation but this can only be a problem if and only if there are validators with a non-0x01 prefixed withdrawal credential.

`ExpectedWithdrawals` iterates over all validators and returns error if it cannot parse the execution address:

```
for range bound {
        // ...
    withdrawalAddress, err = validator.
        GetWithdrawalCredentials().ToExecutionAddress()
    if err != nil {
        return nil, err
    }
```

`ToExecutionAddress` itself returns error if the withdrawal credentials don't have the 0x01 prefix (withdrawal_credentials.go#L55C1-L57C3):

```
if wc[0] != EthSecp256k1CredentialPrefix {
    return common.ExecutionAddress{}, ErrInvalidWithdrawalCredentials
}
```

In the reference implementation, no such check exists. (see the consensus specs on expected withdrawals:

```
if is_fully_withdrawable_validator(validator, balance, epoch):
    withdrawals.append(Withdrawal(
        index=withdrawal_index,
        validator_index=validator_index,
        address=ExecutionAddress(validator.withdrawal_credentials[12:]),
        amount=balance,
    ))
    withdrawal_index += WithdrawalIndex(1)
elif is_partially_withdrawable_validator(validator, balance):
    withdrawals.append(Withdrawal(
        index=withdrawal_index,
        validator_index=validator_index,
        address=ExecutionAddress(validator.withdrawal_credentials[12:]),
        amount=balance - MAX_EFFECTIVE_BALANCE,
    ))
    withdrawal_index += WithdrawalIndex(1)
```

Instead, the withdrawal address is accessed if and only if `is_fully_withdrawable_validator()` or `is_partially_withdrawable_validator()` is true.

These methods have their own 0x01 prefix check (by calling `has_eth1_withdrawal_credential`):

- beacon-chain.md#is_fully_withdrawable_validator:

```
def is_fully_withdrawable_validator(validator: Validator, balance: Gwei, epoch: Epoch) -> bool:
    """
    Check if ``validator`` is fully withdrawable.
    """
    return (
        has_eth1_withdrawal_credential(validator)
        and validator.withdrawable_epoch <= epoch
        and balance > 0
    )
```

- beacon-chain.md#is_partially_withdrawable_validator:

```
def is_partially_withdrawable_validator(validator: Validator, balance: Gwei) -> bool:
    """
    Check if ``validator`` is partially withdrawable.
    """
    has_max_effective_balance = validator.effective_balance == MAX_EFFECTIVE_BALANCE
    has_excess_balance = balance > MAX_EFFECTIVE_BALANCE
    return has_eth1_withdrawal_credential(validator) and has_max_effective_balance and
    ↪  has_excess_balance
```

So, if a validator would exist with a non-0x01 prefixed withdrawal credential, `ExpectedWithdrawals` would return error, whereas the reference implementation would continue to iterate over the remaining validators (but not add it to the `withdrawals` return value).

Recommendation: Do not prematurely call `ToExecutionAddress` but access the withdrawal credential whenever it is needed, just like in the reference implementation:

```
diff --git a/mod/state-transition/pkg/core/state/statedb.go
↪  b/mod/state-transition/pkg/core/state/statedb.go
index 830ad7e..fb3c4a6 100644
--- a/mod/state-transition/pkg/core/state/statedb.go
+++ b/mod/state-transition/pkg/core/state/statedb.go
@@ -236,18 +236,19 @@ func (s *StateDB[
            return nil, err
        }
```

```
-       withdrawalAddress, err = validator.
-           GetWithdrawalCredentials().ToExecutionAddress()
-       if err != nil {
-           return nil, err
-       }
-
        // Set the amount of the withdrawal depending on the balance of the
        // validator.
        var withdrawal WithdrawalT

        //nolint:gocritic // ok.
        if validator.IsFullyWithdrawable(balance, epoch) {
+           withdrawalAddress, err := validator.
+               GetWithdrawalCredentials().ToExecutionAddress()
+           if err != nil {
+               /* Shouldn't be possible */
+               return nil, err
+           }
+
            withdrawals = append(withdrawals, withdrawal.New(
                math.U64(withdrawalIndex),
                validatorIndex,
@@ -260,6 +261,13 @@ func (s *StateDB[
        } else if validator.IsPartiallyWithdrawable(
            balance, math.Gwei(s.cs.MaxEffectiveBalance()),
        ) {
+           withdrawalAddress, err := validator.
+               GetWithdrawalCredentials().ToExecutionAddress()
+           if err != nil {
+               /* Shouldn't be possible */
+               return nil, err
+           }
+
            withdrawals = append(withdrawals, withdrawal.New(
                math.U64(withdrawalIndex),
                validatorIndex,
@@ -270,6 +278,12 @@ func (s *StateDB[
            // Increment the withdrawal index to process the next withdrawal.
            withdrawalIndex++
        } else if s.cs.DepositEth1ChainID() == spec.BartioChainID {
+           withdrawalAddress, err := validator.
+               GetWithdrawalCredentials().ToExecutionAddress()
+           if err != nil {
+               ??????
+           }
+
            // Backward compatibility with Bartio
            // TODO: Drop this when we drop other Bartio special cases.
            withdrawal = withdrawal.New(
```

Berachain: Fixed in PR 2231.

Spearbit: Fix verified.

5.2 Medium Risk

5.2.1 Lack of validator penalties enables risk-free economic censorship and liveness attacks

Severity: Medium Risk

Context: mod/state-transition/pkg/core/state_processor.go

Description: Following is the current economic security / consensus safety model:

- Genesis validator set is picked by social trust.
- There is a cap on total no. of active validators participating in consensus at any given point in time.
- To become a validator, a deposit of arguably high stake needs to be made on execution layer such that it is greater than minimum balance of current validator set.
- There are no penalties, slashing, reputation score in current state. Also, any cometbft evidences are not considered.
- Rewards are accrued and distributed outside of the consensus layer.

Several points to note:

- Deposit barrier can easily be bypassed by pooling the required amount from multiple parties.
- There is no way to differentiate between an honest validator and validator who is acting maliciously, so even the offchain rewards distribution would be on basis of current active validator set for a given epoch or some derivative of that.

There are several issues with this approach:

- Allows voting on multiple proposal without any risk.
- Validator can propose block for any additional rewards (via fee recepient on `block.coinbase`) when its turn (which is deterministic due to cometbft weighted round-robin) and refrain from participating in any other activities.
- There is no inactivity leak so even any honest validator who is offline or has lost access to keys would not be kicked out of network unless someone else deposits more stake.
- There are potential spam and grieving vectors on all direct entry points to network such as filling up deposit queue, sending spam proposals, withdrawing and creating new validator from same stake, etc. which are zero-cost & risk-free activites.

Recommendation:

- Implement slashing logic.
- Implement inactivity leak protection.
- Rewards and penalties needs to be baked into consensus layer, much more closer to cometbft.
- Integrate with `abci.Evidence`.
- If more concrete and viable solution needs to be researched, document the process of guarded launch for time being.

Berachain: Acknowledged. Slashing and inactivity leaks are planned upgrades. Rewards will be handled via PoL.

Spearbit: Acknowledged.

5.2.2 `validatorUpdates` ignores the effects of `ProcessBlock` leading to wrong voting power for validators temporarily

Severity: Medium Risk

Context: state_processor.go#L187-L195

Description: Here is the current order of state processing in `Transition` function:

- Transition
  - ProcessSlots ⇒ (`validatorUpdates` and `IncreaseBalance/DecreaseBalance` happens correctly).
    - `processSlot` (for each slot until target).
      - If epoch boundary: `processEpoch`.
  - `ProcessBlock` ⇒ (`IncreaseBalance/DecreaseBalance` happens correctly, `validatorUpdates` are ignored and not sent to cometbft till next epoch).
    - `processBlockHeader`.
    - `processExecutionPayload`.
    - `processWithdrawals` ⇒ (can decrease balance).
    - `processRandaoReveal`.
    - `processOperations` ⇒ (can increase balance).

This breaks the various invariants and specs, major ones being voting power does not reflect the underlying stake and can be temporarily undermined or inflated

Recommendation: `validatorUpdates` should be sent to cometbft after `ProcessBlock`

Berachain: Acknowledged. This has been deprecated by a fix in PR 2226.

Spearbit: Verified.

5.2.3 Broker's `Publish, Broadcast & Shutdown` are unrealiable for delivery guarantees and prone to various race panics

Severity: Medium Risk

Context: mod/async/pkg/broker/broker.go

Description: Publish/Broadcast and shutdown (due to `<- ctx.done()` in start) can happen concurrently, this results in non-deterministic and unsound behavior resulting in following scenarios:

- Return `ctx.Canceled` or `ctx.DeadlineExceeded` error.
- Msg delivered to some subscribers but not to others.
- `panic: closing already closed channel` when ctx cancels/timeout from multiple places.
- `panic: send on closed channel` when broadcast is trying to iterate and shutdown has already closed some.

Here, only returning error is desirable and rest are unwanted side-effects can cause severe damage. These effects are observed when node operations deviate from the happy path.

Recommendation:

- Protect map by mutex or any sync primitive.
- Ensure idempotent behavior for subscribers and publishers.

Berachain: Fixed in PR 2225.

Spearbit: Fix verified.

5.2.4 Race condition results in mismatch between `BeaconBlock.StateRoot` and `BlodSidecar.BeaconBlockHeader.StateRoot`

Severity: Medium Risk

Context: beacon/validator/block_builder.go#L108-L124

Description: While building the `BeaconBlock` and `BlobSidecars` for proposal, there is the following two goroutines:

```
g.Go(func() error {
    sidecars, err = s.blobFactory.BuildSidecars(
        blk, envelope.GetBlobsBundle(),
    )
    return err
})


// Compute the state root for the block.
g.Go(func() error {
    return s.computeAndSetStateRoot(
    ctx,
    slotData.GetProposerAddress(),
    slotData.GetConsensusTime(),
    st,
    blk,
    )
})
```

The first goroutine calls `BuildSidecars` which calls `blk.GetHeader()`. This reads the `StateRoot` of the `BeaconBlock` and places it into the `BlobSidecar`.

The second goroutine calculates and sets the `StateRoot` in the `BeaconBlock`.

They both happen at the same time - This could result in a mismatched state root between the `BeaconBlock.StateRoot` and each `BlobSidecar.BeaconBlockHeader.StateRoot`.

Recommendation: There are two options:

1. The simplest option would be to serialize these two calls. First calculate the state root and then build the sidecars.
2. If wanting to keep the benefits of parallelization, we can set the `BlobSidecar.BeaconBlockHeader.StateRoot` to a placeholder value during the parallel goroutines. After both goroutines are complete, then each `BlobSidecar.BeaconBlockHeader.StateRoot` value can get updated to the new `BeaconBlock.StateRoot`.

Berachain: Fixed in commit 54047740.

Spearbit: Fix verified.

5.2.5 Broker broadcasts sequentially can potentially timeout upstream block building / verification

Severity: Medium Risk

Context: mod/async/pkg/broker/broker.go

Description:

```
func (b *Broker[T]) broadcast(msg T) {
    for client := range b.subscriptions {
        // send msg to client (or discard msg after timeout)
    // we could consider using a go routine for each client to allow
    // for concurrent notification attempts, while respecting the timeout
    // for each client individually
    select {
    case client <- msg:
    case <-time.After(b.timeout):
    }
    }
}
```

and `b.timeout` is set to 1 second

It blocks for each subscriber before sending message to next subscriber:

- If subscriber doesn't accept for whatever reason (channel buffer limit reached, etc...).
- If timed out by broadcast context.
- If timed out by upstream context.

The consequences are following:

- Potentially out of sync services leading to non-deterministic & faulty behavior.
- Partially committed states.

However, This behavior is likely not to be triggered under normal circumstances. But can occur during:

- If CL and EL are disconnected and reconnected.
- If network is recovering from failure to propose blocks.
- If more events and/or subscribers are added, risk scales non-linearly.
- When performing initial sync to reach head.

Recommendation: Turn it into a goroutine:

```
func (b *Broker[T]) broadcast(msg T) {
    if msg.Context().Err() != nil {
        panic(msg.Context().Err())
    }

    for client := range b.subscriptions {
        go func(client chan T) {
        select {
            case client <- msg:
        case <-time.After(b.timeout):
        }
    }(client)
    }
}
```

Berachain: Fixed in PR 2225.

Spearbit: Fix verified.

5.2.6 Race condition in `da/pkg/store/store.Persist()` due to bug in `RangeDB`

Severity: Medium Risk

Context: mod/storage/pkg/filedb/range_db.go

Description: This was already hypothesized in `becon-kit` issue 3 but here's a more concrete example, using `store.Persist()` which uses explicit concurrency. The root cause of the bug is in RangeDB, not Store.

Put this in `beacon-kit/mod/da/pkg/store/store_test.go` (note: this test removes the directory `/tmp/store_-test`):

```
package store_test

import (
    "os"
    "testing"
    "github.com/berachain/beacon-kit/mod/da/pkg/store"
    "github.com/berachain/beacon-kit/mod/consensus-types/pkg/types"
    "github.com/berachain/beacon-kit/mod/storage/pkg/filedb"
    "github.com/berachain/beacon-kit/mod/config/pkg/spec"
    datypes "github.com/berachain/beacon-kit/mod/da/pkg/types"
    "cosmossdk.io/log"
)

func TestRace(t *testing.T) {
    err := os.RemoveAll("/tmp/store_test")
    if err != nil {
        panic(err)
    }

    logger := log.NewNopLogger()
    s := store.New[*types.BeaconBlockBody](
        filedb.NewRangeDB(
            filedb.NewDB(filedb.WithRootDirectory("/tmp/store_test"),
                filedb.WithFileExtension("ssz"),
                filedb.WithDirectoryPermissions(0700),
                filedb.WithLogger(logger),
            ),
        ),
        logger.With("service", "da-store"),
        spec.TestnetChainSpec(),
    )
    sc := make([]*datypes.BlobSidecar, 20)
    for i := range sc {
        sc[i] = &datypes.BlobSidecar{
            Index: uint64(i),
            BeaconBlockHeader: &types.BeaconBlockHeader{},
        }
    }
    sidecars := datypes.BlobSidecars{
        Sidecars: sc,
    }

    err = s.Persist(0, &sidecars)
    err = s.Persist(1, &sidecars)
    err = s.Prune(0, 1)
    err = s.Persist(0, &sidecars)
}
```

Run:

```
go test -race -run TestRace
```

Output:

```
==================
WARNING: DATA RACE
Read at 0x00c000f8a5f8 by goroutine 563:
  github.com/berachain/beacon-kit/mod/storage/pkg/filedb.(*RangeDB).Set()
      /home/jhg/berachain-race-condition-test/beacon-kit/mod/storage/pkg/filedb/range_db.go:75 +0x6d
  github.com/berachain/beacon-kit/mod/da/pkg/store.(*Store[go.shape.*uint8]).Persist.func1()
      /home/jhg/berachain-race-condition-test/beacon-kit/mod/da/pkg/store/store.go:109 +0x114
  github.com/sourcegraph/conc/iter.Mapper[go.shape.*uint8,go.shape.interface { Error() string
  ↪  }].Map.func1()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/iter/map.go:29
      ↪  +0x53
  github.com/sourcegraph/conc/iter.Iterator[go.shape.*uint8].ForEachIdx.func1()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/iter/iter.go:76
      ↪  +0x86
  github.com/sourcegraph/conc/panics.(*Catcher).Try()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/panics/panics.go:23
      ↪  +0x77
  github.com/sourcegraph/conc.(*WaitGroup).Go.func1()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/waitgroup.go:32
      ↪  +0x8f

Previous write at 0x00c000f8a5f8 by goroutine 565:
  github.com/berachain/beacon-kit/mod/storage/pkg/filedb.(*RangeDB).Set()
      /home/jhg/berachain-race-condition-test/beacon-kit/mod/storage/pkg/filedb/range_db.go:76 +0x87
  github.com/berachain/beacon-kit/mod/da/pkg/store.(*Store[go.shape.*uint8]).Persist.func1()
      /home/jhg/berachain-race-condition-test/beacon-kit/mod/da/pkg/store/store.go:109 +0x114
  github.com/sourcegraph/conc/iter.Mapper[go.shape.*uint8,go.shape.interface { Error() string
  ↪  }].Map.func1()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/iter/map.go:29
      ↪  +0x53
  github.com/sourcegraph/conc/iter.Iterator[go.shape.*uint8].ForEachIdx.func1()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/iter/iter.go:76
      ↪  +0x86
  github.com/sourcegraph/conc/panics.(*Catcher).Try()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/panics/panics.go:23
      ↪  +0x77
  github.com/sourcegraph/conc.(*WaitGroup).Go.func1()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/waitgroup.go:32
      ↪  +0x8f

Goroutine 563 (running) created at:
  github.com/sourcegraph/conc.(*WaitGroup).Go()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/waitgroup.go:30
      ↪  +0xe4
  github.com/sourcegraph/conc/iter.Iterator[go.shape.*uint8].ForEachIdx()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/iter/iter.go:82
      ↪  +0x23e
```

```
  github.com/sourcegraph/conc/iter.Mapper[go.shape.*uint8,go.shape.interface { Error() string }].Map()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/iter/map.go:28
      ↪  +0x190
  github.com/sourcegraph/conc/iter.Map[go.shape.*uint8,go.shape.interface { Error() string }]()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/iter/map.go:20
      ↪  +0x2a5
  github.com/berachain/beacon-kit/mod/da/pkg/store.(*Store[go.shape.*uint8]).Persist()
      /home/jhg/berachain-race-condition-test/beacon-kit/mod/da/pkg/store/store.go:98 +0x2b5
  github.com/berachain/beacon-kit/mod/da/pkg/store_test.TestRace()
      /home/jhg/berachain-race-condition-test/beacon-kit/mod/da/pkg/store/store_test.go:46 +0x857
  testing.tRunner()
      /home/jhg/berachain-race-condition-test/go/src/testing/testing.go:1690 +0x226
  testing.(*T).Run.gowrap1()
      /home/jhg/berachain-race-condition-test/go/src/testing/testing.go:1743 +0x44

Goroutine 565 (running) created at:
  github.com/sourcegraph/conc.(*WaitGroup).Go()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/waitgroup.go:30
      ↪  +0xe4
  github.com/sourcegraph/conc/iter.Iterator[go.shape.*uint8].ForEachIdx()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/iter/iter.go:82
      ↪  +0x23e
  github.com/sourcegraph/conc/iter.Mapper[go.shape.*uint8,go.shape.interface { Error() string }].Map()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/iter/map.go:28
      ↪  +0x190
  github.com/sourcegraph/conc/iter.Map[go.shape.*uint8,go.shape.interface { Error() string }]()
      /home/jhg/berachain-race-condition-test/go/packages/pkg/mod/github.com/sourcegraph/conc@v0.3.1-0.
      ↪  20240121214520-5f936abd7ae8/iter/map.go:20
      ↪  +0x2a5
  github.com/berachain/beacon-kit/mod/da/pkg/store.(*Store[go.shape.*uint8]).Persist()
      /home/jhg/berachain-race-condition-test/beacon-kit/mod/da/pkg/store/store.go:98 +0x2b5
  github.com/berachain/beacon-kit/mod/da/pkg/store_test.TestRace()
      /home/jhg/berachain-race-condition-test/beacon-kit/mod/da/pkg/store/store_test.go:46 +0x857
  testing.tRunner()
      /home/jhg/berachain-race-condition-test/go/src/testing/testing.go:1690 +0x226
  testing.(*T).Run.gowrap1()
      /home/jhg/berachain-race-condition-test/go/src/testing/testing.go:1743 +0x44
==================
--- FAIL: TestRace (0.03s)
    testing.go:1399: race detected during execution of test
FAIL
exit status 1
FAIL    github.com/berachain/beacon-kit/mod/da/pkg/store    0.125s
```

Recommendation: Make DB writes sequential.

Berachain: Fixed in PR 2258.

Spearbit: Fix verified.

5.3 Low Risk

5.3.1 Invalid range inputs for deposit store pruner

Severity: Low Risk

Context: mod/node-core/pkg/components/deposit_store.go

Description: While running kurtosis cluster and testnet, some of errors of deposit store pruner were observed

```
^^[[90m2024-11-26T09:12:01Z ^^[[31mERRR^^[[0m  error pruning index  service=deposit-store-pruner^^[[0m
↪  ^^[[31merror=range start greater than end^^[[0m
```

The root cause is unknown but indicates the issue with either arguments being passed or data not available.

Recommendation: Add integration/e2e to identify root cause and ensure proper validation.

Berachain: Fixed in commit bd74b71.

Spearbit: Fix verified.

5.3.2 Race condition in `dispatch.Subscribe` can crash the node during startup / restart

Severity: Low Risk

Context: mod/async/pkg/dispatcher/dispatcher.go#L71

`Subscribe` on broker will access map `subscriptions`.

```
b.subscriptions[client] = struct{}{}
```

It is not concurrency safe, If more than one subscriber tries to `.Subscribe` then it will panic. This can happen during init (`StartAll`) where all services subscribe at once:

```
package dispatcher

import (
    "context"
    "sync"
    "testing"

    "github.com/berachain/beacon-kit/mod/async/pkg/broker"
    "github.com/berachain/beacon-kit/mod/log/pkg/noop"
    "github.com/berachain/beacon-kit/mod/primitives/pkg/async"
)

func TestDispatcherRaceCondition(t *testing.T)  {
       logger := noop.NewLogger[any]()

    ctx, cancel := context.WithCancel(context.Background())
    defer cancel()

    dispatcher, err := New(logger, WithEvent[async.BaseEvent](async.GenesisDataReceived))
    if err != nil {
        t.Fatalf("Failed to create dispatcher: %v", err)
    }

    dispatcher.Start(ctx)


    numPublishers := 50
    numSubscribers := 100
    // numUnsubscribers := 25
    numMessagesPerPublisher := 100
```

```
    var wg sync.WaitGroup

    wg.Add(numSubscribers)

    for i := 0; i < numSubscribers; i++ {
        go func(id int) {
            defer wg.Done()
            ch := make(chan async.BaseEvent, numPublishers*numMessagesPerPublisher)
            _ = dispatcher.Subscribe(async.GenesisDataReceived, ch)
        }(i)
    }

    wg.Wait()
}
```

Output: `fatal error: concurrent map writes`.

Berachain: Fixed in PR 2225.

Spearbit: Fix verified.

5.3.3 Possible DoS via premature state access in beacon block processing

Severity: Low Risk

Context: mod/beacon/blockchain/receive.go

Description:

- `txn` is received from cometbft to ABCI via `ProcessProposal(tx [][]byte)`.
- `txn[beaconIndex=0]` is beacon block.
- beacon block is unmarshalled into json from protobuf.
- to verify validity and correctness of block following sequence, a copy of chain state is accessed made from the DB (which is a disk-op and also memory consuming) and it is checked if beacon block is nil/empty or malformed.

This is directly accessible over gossip and increase CPU, RAM and disk usage unnecessarily and may cause temporary DoS. This can be serious since cosmos peer reputation score or evidence are not used.

At same time, this cannot halt chain / impact validators because they are generally guarded by sentry nodes.

Recommendation: Move the nil checks and other static checks over state access.

Berachain: Acknowledged. This will be considered at a later date.

Spearbit: Acknowledged.

5.3.4 `cometbft` version is using BLS implementation which allows keys outside of allowed subgroup

Severity: Low Risk

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

Description:

```go
// VerifySignature verifies a signature against a message and a public key.
func (f BLSSigner) VerifySignature(
    pubKey crypto.BLSPubkey,
    msg []byte,
    signature crypto.BLSSignature,
) error {
    if ok := bls12381.PubKey(pubKey[:]).
        VerifySignature(msg, signature[:]); !ok {
        return ErrInvalidSignature
    }
    return nil
}
```

`beacon-kit` uses BLS keys extensively for signing messages in cometbft validator set and verifying messages signed by other validators.

The cometbft version used in signer package is `github.com/cometbft/cometbft v1.0.0-rc1.0.20240806094948-2c4293ef36c4` . This version accepts public keys which are outside of the permitted subgroup due to loose validation checks.

Due to this, It can force computations and operations on larger subgroup which are expensive and can consume more resources on node. However, computations are still mathematically valid and deterministic, there is still unknown risk due to unexpected cryptographic properties.

Recommendation: The BLS module was fixed in `cometbft` in PR 4104. Upgrade `cometbft` version to a more recent commit in all packages.

Berachain: Fixed in PR 2221.

Spearbit: Fix verified.

5.4 Informational

5.4.1 Duplicate validation of `BeaconBlockHeader.HashTreeRoot()`

Severity: Informational

Context: da/pkg/blob/verifier.go#L96-L98

Description: Each `BlobSidecar`'s `BeaconBlockHeader.HashTreeRoot` is validated against the corresponding `BeaconBlock`'s `HashTreeRoot` at da/pkg/blob/verifier.go#L75-L79. This entirely encompasses the following check that ensures that each `BlobSidecar` has the same `HashTreeRoot` at da/pkg/blob/verifier.go#L96-L98.

Recommendation: Remove the duplicate check at da/pkg/blob/verifier.go#L96-L98.

Berachain: Fixed in PR 2245.

Spearbit: Fix verified.

5.4.2 Unvalidated `BlobSidecar.Index` field allows integer overflow

Severity: Informational

Context: da/pkg/types/sidecar.go#L89

Description: The `BlobSidecar.Index` field is used during the inclusion proof computation in `HasValidInclusionProof()`. This field is added with another value (`kzgOffset+b.Index` at da/pkg/types/sidecar.go#L89). However, the value of `b.Index` is never checked to be a valid index. This could result in supplying any `uint64` value as the index, resulting in overflow during `HasValidInclusionProof()`.

This currently has no impact, as an overflow of this field should result in an invalid inclusion proof, correctly returning an error. However, in principle, overflowing arithmetic should be prevented ahead of time.

Recommendation: In `ProcessProposal()`, ensure that the `BlobSidecar.Index` field has a value less than `spec.MaxBlobsPerBlock`.

Berachain: Fixed in PR 2289.

Spearbit: Fix verified.

5.4.3 Sidecar omission not checked in `ProcessProposal`

Severity: Informational

Context: da/pkg/da/service.go#L195-L198

Description: During `ProcessProposal()`, there is no check to verify that `KzgCommitments` included in the `BeaconBlock` have a corresponding `BlobSidecar`. This means that a proposer could omit the corresponding `BlobSidecars` for a `BeaconBlock` and still pass the checks in `ProcessProposal()`.

This will get caught in `FinalizeBlock()` when it checks to ensure that the data is available. However, this should get caught during the proposal instead of during block finalization.

Recommendation: During `ProcessProposal()`, ensure that all `KzgCommitments` in the `BeaconBlock` have a corresponding `BlobSidecar`.

Berachain: Fixed in PR 2291.

Spearbit: Fix verified.

5.4.4 Possible chain halt if vote extensions are used due to vulnerable CometBFT version

Severity: Informational

Context: noops.go#L64-L76, go.mod#L12

Description: Cometbft version `v1.0.0-rc1.0.20240805092115-3b2c5d9e1843` that is used is vulnerable to a high severity issue (`https://github.com/cometbft/cometbft/security/advisories/GHSA-p7mv-53f2-4cwj`).

The vulnerability allows a malicious actor that's part of the network to send a message including a vote that will halt the chain.

However - impact only occurs if vote extensions are enabled. it seems that currently, berachain did not implement calls to vote logic but rather kept it in placeholders in `noops.go`:

```
func (Service[_]) ExtendVote(
    context.Context,
    *abci.ExtendVoteRequest,
) (*abci.ExtendVoteResponse, error) {
    return &abci.ExtendVoteResponse{}, nil
}

func (Service[_]) VerifyVoteExtension(
    context.Context,
    *abci.VerifyVoteExtensionRequest,
) (*abci.VerifyVoteExtensionResponse, error) {
    return &abci.VerifyVoteExtensionResponse{}, nil
}
```

Therefore no direct impact in current implementation. If `ExtendVote` and `VerifyVoteExtension` will be implemented in the future, there is a risk of vulnerability becoming relevant.

Recommendation: Update to patched version as mentioned in the GHSA-p7mv-53f2-4cwj advisory.

Berachain: Acknowledged. There currently is no intention of using `VoteExtensions`, but will keep this in mind for the future.

Spearbit: Acknowledged.

5.4.5 SSZ roundtrip failures on size-constrained slice types

Severity: Informational

Context: da/types/sidecars.go#L97-L98, consensus-types/types/body.go#L127-L134, consensus-types/types/payload.go#L118

Description/Recommendation: These are slice types for which SSZ serialization is implemented:

- `BlobSidecars` in `da`.
- `BeaconBlockBody.BlobKzgCommitments` in `consensus-types`.
- `BeaconBlockBody.Deposits` in `consensus-types`.
- `ExecutionPayload.Withdrawals` in `consensus-types`.

In their `DefineSSZ()` methods, the slice size is constrained to 6, 16, 16 and 16 elements respectively. Serializing slices with a number of elements in excess of these constants is possible, but subsequent serialization will fail.

```
package main

import (
    datypes "github.com/berachain/beacon-kit/da/types"
    constypes "github.com/berachain/beacon-kit/consensus-types/types"
    "github.com/berachain/beacon-kit/engine-primitives/engine-primitives"
    "github.com/berachain/beacon-kit/primitives/eip4844"
    "fmt"
)

func test_BlobSidecars() {
    var in, out datypes.BlobSidecars
    for i := 0; i < 10; i++ {
        in.Sidecars = append(in.Sidecars, &datypes.BlobSidecar{})
    }
    /* Succeeds */
    serialized_ssz, err := in.MarshalSSZ()
    if err != nil {
        fmt.Println("MarshalSSZ failed for BlobSidecars", err)
```

```
        return
    }
    /* Fails */
    err = out.UnmarshalSSZ(serialized_ssz)
    if err != nil {
        fmt.Println("UnmarshalSSZ failed for BlobSidecars", err)
        return
    }
}

func test_BeaconBlockBody_BlobKzgCommitments() {
    var in, out constypes.BeaconBlockBody
    for i := 0; i < 20; i++ {
        in.BlobKzgCommitments = append(in.BlobKzgCommitments, eip4844.KZGCommitment{})
    }
    /* Succeeds */
    serialized_ssz, err := in.MarshalSSZ()
    if err != nil {
        fmt.Println("MarshalSSZ failed for BeaconBlockBody.BlobKzgCommitments", err)
        return
    }
    /* Fails */
    err = out.UnmarshalSSZ(serialized_ssz)
    if err != nil {
        fmt.Println("UnmarshalSSZ failed for BeaconBlockBody.BlobKzgCommitments", err)
        return
    }
}

func test_BeaconBlockBody_Deposits() {
    var in, out constypes.BeaconBlockBody
    for i := 0; i < 20; i++ {
        in.Deposits = append(in.Deposits, &constypes.Deposit{})
    }
    /* Succeeds */
    serialized_ssz, err := in.MarshalSSZ()
    if err != nil {
        fmt.Println("MarshalSSZ failed for BeaconBlockBody.Deposits", err)
        return
    }
    /* Fails */
    err = out.UnmarshalSSZ(serialized_ssz)
    if err != nil {
        fmt.Println("UnmarshalSSZ failed for BeaconBlockBody.Deposits", err)
        return
    }
}

func test_ExecutionPayload_Withdrawals() {
    var in, out constypes.ExecutionPayload
    for i := 0; i < 20; i++ {
        in.Withdrawals = append(in.Withdrawals, &engineprimitives.Withdrawal{})
    }
    /* Succeeds */
    serialized_ssz, err := in.MarshalSSZ()
    if err != nil {
        fmt.Println("MarshalSSZ failed for ExecutionPayload.Withdrawals", err)
        return
    }
    /* Fails */
    err = out.UnmarshalSSZ(serialized_ssz)
    if err != nil {
```

```
        fmt.Println("UnmarshalSSZ failed for ExecutionPayload.Withdrawals", err)
        return
    }
}

func main() {
    test_BlobSidecars()
    test_BeaconBlockBody_BlobKzgCommitments()
    test_BeaconBlockBody_Deposits()
    test_ExecutionPayload_Withdrawals()
}
```

Output:

```
UnmarshalSSZ failed for BlobSidecars ssz: maximum item count exceeded: decoded 10, max 6
UnmarshalSSZ failed for BeaconBlockBody.BlobKzgCommitments ssz: maximum item count exceeded: decoded
↪  20, max 16
UnmarshalSSZ failed for BeaconBlockBody.Deposits ssz: maximum item count exceeded: decoded 20, max 16
UnmarshalSSZ failed for ExecutionPayload.Withdrawals ssz: maximum item count exceeded: decoded 20, max
↪  16
```

Other struct members elsewhere in the code might also be susceptible, such as in `BeaconState`:

- state.go#L176: `BlockRoots`.
- state.go#L177: `StateRoots`.
- state.go#L189: `RandaoMixes`.

Berachain: Acknowledged. This will be considered at a later date.

Spearbit: Acknowledged.

5.4.6 Potential overflow and zero division in `mod/state-transition/pkg/core.processSlash()`

Severity: Informational

Context: mod/state-transition/pkg/core/state_processor_slashing.go#L128-L131

Description: In `processSlash`:

```
// Calculate the penalty.
increment := sp.cs.EffectiveBalanceIncrement()
balDivIncrement := val.GetEffectiveBalance().Unwrap() / increment
penaltyNumerator := balDivIncrement * adjustedTotalSlashingBalance
penalty := penaltyNumerator / totalBalance * increment
```

If we ignore assumptions about the values of the variables involved, there is the potential of multiplication overflow, and division by zero (if `increment` or `totalBalance` are 0).

Rather than rely on assumptions (whose validity can change as the code base progresses), I propose to bail on invalid divisors and use overflow-safe math, to guarantee correct logic at the expense of minimal performance overhead.

Recommendation:

```
diff --git a/mod/state-transition/pkg/core/state_processor_slashing.go
↪  b/mod/state-transition/pkg/core/state_processor_slashing.go
index 581852a..1669ea9 100644
--- a/mod/state-transition/pkg/core/state_processor_slashing.go
+++ b/mod/state-transition/pkg/core/state_processor_slashing.go
@@ -22,6 +22,9 @@ package core

 import (
```

```
    "github.com/berachain/beacon-kit/mod/primitives/pkg/math"
+   "fmt"
+   "math/big"
+   "errors"
 )

 // processSlashingsReset as defined in the Ethereum 2.0 specification.
@@ -113,6 +116,40 @@ func (sp *StateProcessor[
    return nil
 }

+// Calculate the slashing penalty.
+func calculatePenalty(
+   effectiveBalance uint64,
+   increment uint64,
+   adjustedTotalSlashingBalance uint64,
+   totalBalance uint64) (uint64, error) {
+   if increment == 0 || totalBalance == 0 {
+       return 0, errors.New("increment and totalBalance must be non-zero")
+   }
+
+   // Convert all inputs to big.Int
+   bigVal := new(big.Int).SetUint64(effectiveBalance)
+   bigIncrement := new(big.Int).SetUint64(increment)
+   bigAdjTotalSlashing := new(big.Int).SetUint64(adjustedTotalSlashingBalance)
+   bigTotalBalance := new(big.Int).SetUint64(totalBalance)
+
+   // balDivIncrement := effectiveBalance / increment
+   balDivIncrement := new(big.Int).Div(bigVal, bigIncrement)
+
+   // penaltyNumerator := balDivIncrement * adjustedTotalSlashingBalance
+   penaltyNumerator := new(big.Int).Mul(balDivIncrement, bigAdjTotalSlashing)
+
+   // penalty := penaltyNumerator / totalBalance * increment
+   penalty := new(big.Int).Div(penaltyNumerator, bigTotalBalance)
+   penalty.Mul(penalty, bigIncrement)
+
+   // Check if result fits in uint64
+   if !penalty.IsUint64() {
+       return 0, fmt.Errorf("penalty calculation overflow: %s", penalty.String())
+   }
+
+   return penalty.Uint64(), nil
+}
+
 // processSlash handles the logic for slashing a validator.
 //
 //nolint:unused // will be used later
@@ -124,11 +161,14 @@ func (sp *StateProcessor[
    adjustedTotalSlashingBalance uint64,
    totalBalance uint64,
 ) error {
-   // Calculate the penalty.
-   increment := sp.cs.EffectiveBalanceIncrement()
-   balDivIncrement := val.GetEffectiveBalance().Unwrap() / increment
-   penaltyNumerator := balDivIncrement * adjustedTotalSlashingBalance
-   penalty := penaltyNumerator / totalBalance * increment
+   penalty, err := calculatePenalty(
+       val.GetEffectiveBalance().Unwrap(),
+       sp.cs.EffectiveBalanceIncrement(),
+       adjustedTotalSlashingBalance,
+       totalBalance)
```

```
+   if err != nil {
+       return err
+   }

    // Get the val index and decrease the balance of the validator.
    idx, err := st.ValidatorIndexByPubkey(val.GetPubkey())
```

Berachain: This section of code has been removed.

Spearbit: Fixed.

5.4.7 Suggestion: Check for overflow in `state-transition/pkg/core/state.IncreaseBalance()`

Severity: Informational

Context: mod/state-transition/pkg/core/state/statedb.go#L145

Description/Recommendation: The expression `balance+delta` is susceptible to a silent overflow:

```
// IncreaseBalance increases the balance of a validator.
func (s *StateDB[
    _, _, _, _, _, _, _, _, _, _,
]) IncreaseBalance(
    idx math.ValidatorIndex,
    delta math.Gwei,
) error {
    balance, err := s.GetBalance(idx)
    if err != nil {
        return err
    }
    return s.SetBalance(idx, balance+delta)
}
```

We suggest changing it to something like:

```
// IncreaseBalance increases the balance of a validator.
// Returns error if the addition would cause uint64 overflow.
func (s *StateDB[
    _, _, _, _, _, _, _, _, _, _,
]) IncreaseBalance(
    idx math.ValidatorIndex,
    delta math.Gwei,
) error {
    balance, err := s.GetBalance(idx)
    if err != nil {
        return err
    }

    if math.MaxUint64-balance < delta {
        return fmt.Errorf("balance overflow")
    }

    return s.SetBalance(idx, balance+delta)
}
```

There shouldn't be any risk of bona fide balances exceeding 64 bits. Even in the hypothetical scenario where this could occur, this change wouldn't make a material difference, as neither code path (not storing the balance, or storing the overflown balance) is storing the proper balance.

But this is a good opportunity to catch (a subset of) instances where `delta` (computed elsewhere, prior to calling `IncreaseBalance`) was itself prone to a computation error (such as a `uint64` underflow, resulting in a very large value) and to prevent it from propagating into the balance state.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.4.8 `da.pkg.blob.BuildBlockBodyProof` panics if `body.Length()` is 0

Severity: Informational

Context: mod/da/pkg/blob/factory.go#L148

Description: `BuildBlockBodyProof` calls `merkle.NewTreeWithMaxLeaves()` with the `maxLeaves` parameter set to `body.Length() - 1`.

```
// BuildBlockBodyProof builds a block body proof.
func (f *SidecarFactory[_, BeaconBlockBodyT, _]) BuildBlockBodyProof(
    body BeaconBlockBodyT,
) ([]common.Root, error) {
    startTime := time.Now()
    defer f.metrics.measureBuildBlockBodyProofDuration(startTime)
    tree, err := merkle.NewTreeWithMaxLeaves[common.Root](
        body.GetTopLevelRoots(),
        body.Length()-1,
    )
    if err != nil {
        return nil, err
    }

    return tree.MerkleProof(f.kzgPosition)
}
```

If `body.Length` is 0, `NewTreeWithMaxLeaves` will crash as shown in the isolated proof of concept below:

```
package main

import (
    "github.com/berachain/beacon-kit/mod/primitives/pkg/common"
    "github.com/berachain/beacon-kit/mod/primitives/pkg/merkle"
)

func Length() uint64 {
    return 0
}

func main() {
    merkle.NewTreeWithMaxLeaves[common.Root](
        []common.Root{},
        Length()-1)
}
```

```
panic: Next power of 2 is 1 << 64.

goroutine 1 [running]:
github.com/berachain/beacon-kit/mod/primitives/pkg/math/pow.NextPowerOfTwo[...](...)
    /home/jhg/beacon-kit-pocs/beacon-kit/mod/primitives/pkg/math/pow/pow.go:47
github.com/berachain/beacon-kit/mod/primitives/pkg/math.U64.NextPowerOfTwo(0xc0000061c0?)
    /home/jhg/beacon-kit-pocs/beacon-kit/mod/primitives/pkg/math/u64.go:116 +0x8d
github.com/berachain/beacon-kit/mod/primitives/pkg/merkle.NewTreeWithMaxLeaves[...]({0x5d9120, 0x0,
↪  0x0}, 0xc0000061c0)
    /home/jhg/beacon-kit-pocs/beacon-kit/mod/primitives/pkg/merkle/tree.go:68 +0x2a
main.main()
```

Currently informational severity; upgrade if this can be shown to actually happen.

Recommendation: It is better to return error if `body.Length()` is 0 prior to calling `merkle.NewTreeWithMaxLeaves` even if this is thought not to be possible in practice.

Berachain: The `body.Length()` is a hardcoded constant, so this should be unreachable.

Spearbit: Acknowledged.

5.4.9 Notes on `mod/storage/pkg/filedb`

Severity: Informational

Context: mod/storage/pkg/filedb

- **Fuzzing campaign:**

  We performed a fuzzing campaign on `mod/storage/pkg/filedb`, using an in-memory key-value map implementing the same methods as `RangeDB` as an oracle for differential fuzzing. Testing was done on both a real and a virtual (`afero.NewMemMapFs`) filesystem. No issues other than the following were found.

- **Malformed extensions can cause database failures:**

  This should hardly be a concern in the current iteration of the code, but this came out of a fuzzing campaign of `filedb` so I'm reporting it.

  WithFileExtension is used to specify a file extension for the database items. Things like dots and slashes in the extension (e.g. using `../../` as an extension) will cause various database failures. Non-ascii characters like NULL bytes might too.

  As long as static extensions are used (e.g. `ssz` this won't be possible but if the storage layer were to be used more broadly, with potentially dynamically generated extensions, then this could accidentally happen (though still not likely).

  Either way it would be good to enforce alphanumeric filenames with a reasonably small size (e.g. < 10 characters to prevent file size limits as noted in `https://github.com/spearbit-audits/review-berachain-beaconkit-0919/issues/91`). If these constraints are not met, panicking could be reasonable here (as the alternative is a botched database).

  ```
  https://github.com/berachain/beacon-kit/blob/8a041731f2bda14a51f85aa1d159caca6b486c44/mod/storag
  ↪ e/pkg/filedb/db_options.go#L50
  ```

- **Race conditions:**

  There is a risk of race conditions if multiple threads were to operate on the same database, due to concurrent access to both object variables like `firstNonNilIndex` and filesystem items.

- `RangeDB.DeleteRange` **and** `RangeDB.Prune` **potential DoS:**

  In range_db.go#L99-L104, range_db.go#L115-L122.

  These methods can run for a long time if a very long range is specified. I am still researching whether this can happen in the current iteration of the code. Using hex instead of decimal strings can somewhat alleviate compute load (see next item).

- **Using decimal strings for index paths is relatively slow:**

  In range_db.go#L100, range_db.go#L140

  `RangeDB` uses a decimal string to represent indices as directory names.

  Using hex instead of decimal is both faster and producer shorter filenames (which lowers the risk of hitting path size limits as noted in "`RangeDB` does not guarantee producing oversized filenames"), while containing the same amount of information.

  Benchmark:

```
package strconv_test

import (
    "fmt"
    "math/rand"
    "strconv"
    "testing"
)

const (
    numTestValues = 8
    rngSeed      = 42
)

var (
    testValues     []uint64
    decimalStrings []string
    hexStrings     []string
)

func init() {
    r := rand.New(rand.NewSource(rngSeed))
    testValues = make([]uint64, numTestValues)
    decimalStrings = make([]string, numTestValues)
    hexStrings = make([]string, numTestValues)

    for i := range testValues {
        testValues[i] = r.Uint64()
        decimalStrings[i] = strconv.FormatUint(testValues[i], 10)
        hexStrings[i] = strconv.FormatUint(testValues[i], 16)
    }
}

func BenchmarkUint64ToDecimal(b *testing.B) {
    for _, v := range testValues {
        b.Run(fmt.Sprintf("decimal-%d", v), func(b *testing.B) {
            for i := 0; i < b.N; i++ {
                strconv.FormatUint(v, 10)
            }
        })
    }
}

func BenchmarkUint64ToHex(b *testing.B) {
    for _, v := range testValues {
        b.Run(fmt.Sprintf("hex-%d", v), func(b *testing.B) {
            for i := 0; i < b.N; i++ {
                strconv.FormatUint(v, 16)
            }
        })
    }
}

func BenchmarkDecimalToUint64(b *testing.B) {
    for i, s := range decimalStrings {
        b.Run(fmt.Sprintf("decimal-%d", testValues[i]), func(b *testing.B) {
            for i := 0; i < b.N; i++ {
                _, err := strconv.ParseUint(s, 10, 64)
                if err != nil {
                    b.Fatal(err)
                }
```

```
            }
        })
    }
}

func BenchmarkHexToUint64(b *testing.B) {
    for i, s := range hexStrings {
        b.Run(fmt.Sprintf("hex-%d", testValues[i]), func(b *testing.B) {
            for i := 0; i < b.N; i++ {
                _, err := strconv.ParseUint(s, 16, 64)
                if err != nil {
                    b.Fatal(err)
                }
            }
        })
    }
}
```

```
go test -bench=.
```

```
goos: linux
goarch: amd64
cpu: AMD Ryzen 5 5600G with Radeon Graphics
BenchmarkUint64ToDecimal/decimal-12663951391086054483-12          31463632          36.27 ns/op
BenchmarkUint64ToDecimal/decimal-9832119173398632219-12           30155130          35.94 ns/op
BenchmarkUint64ToDecimal/decimal-5571782338101878760-12           32240172          35.63 ns/op
BenchmarkUint64ToDecimal/decimal-1926012586526624009-12           32045224          36.65 ns/op
BenchmarkUint64ToDecimal/decimal-9627525982598323465-12           32228310          35.80 ns/op
BenchmarkUint64ToDecimal/decimal-3534334367214237261-12           32432403          35.77 ns/op
BenchmarkUint64ToDecimal/decimal-7497468244883513247-12           31931006          35.73 ns/op
BenchmarkUint64ToDecimal/decimal-12769259138917390016-12          32071852          36.40 ns/op
BenchmarkUint64ToHex/hex-12663951391086054483-12                  37131082          31.46 ns/op
BenchmarkUint64ToHex/hex-9832119173398632219-12                   32137905          31.29 ns/op
BenchmarkUint64ToHex/hex-5571782338101878760-12                   36602337          31.37 ns/op
BenchmarkUint64ToHex/hex-1926012586526624009-12                   36165129          31.32 ns/op
BenchmarkUint64ToHex/hex-9627525982598323465-12                   37014406          31.36 ns/op
BenchmarkUint64ToHex/hex-3534334367214237261-12                   35491934          31.47 ns/op
BenchmarkUint64ToHex/hex-7497468244883513247-12                   34760338          31.38 ns/op
BenchmarkUint64ToHex/hex-12769259138917390016-12                  36433398          31.30 ns/op
BenchmarkDecimalToUint64/decimal-12663951391086054483-12          42846453          26.90 ns/op
BenchmarkDecimalToUint64/decimal-9832119173398632219-12           45942121          25.52 ns/op
BenchmarkDecimalToUint64/decimal-5571782338101878760-12           47503257          25.90 ns/op
BenchmarkDecimalToUint64/decimal-1926012586526624009-12           46896894          25.41 ns/op
BenchmarkDecimalToUint64/decimal-9627525982598323465-12           46923669          26.13 ns/op
BenchmarkDecimalToUint64/decimal-3534334367214237261-12           44727643          26.31 ns/op
BenchmarkDecimalToUint64/decimal-7497468244883513247-12           45227252          26.66 ns/op
BenchmarkDecimalToUint64/decimal-12769259138917390016-12          43038775          28.25 ns/op
BenchmarkHexToUint64/hex-12663951391086054483-12                  55685986          21.69 ns/op
BenchmarkHexToUint64/hex-9832119173398632219-12                   55826914          21.17 ns/op
BenchmarkHexToUint64/hex-5571782338101878760-12                   56775049          20.69 ns/op
BenchmarkHexToUint64/hex-1926012586526624009-12                   58460756          20.82 ns/op
BenchmarkHexToUint64/hex-9627525982598323465-12                   59924533          21.63 ns/op
BenchmarkHexToUint64/hex-3534334367214237261-12                   55628858          22.65 ns/op
BenchmarkHexToUint64/hex-7497468244883513247-12                   56971554          22.26 ns/op
BenchmarkHexToUint64/hex-12769259138917390016-12                  55283824          22.35 ns/op
```

```
goos: linux
goarch: amd64
cpu: AMD EPYC 7742 64-Core Processor
BenchmarkUint64ToDecimal/decimal-12663951391086054483-256          18463749          63.83
↪  ns/op
BenchmarkUint64ToDecimal/decimal-9832119173398632219-256           18080352          64.90
↪  ns/op
BenchmarkUint64ToDecimal/decimal-5571782338101878760-256           18041667          64.79
↪  ns/op
BenchmarkUint64ToDecimal/decimal-1926012586526624009-256           18433909          64.72
↪  ns/op
BenchmarkUint64ToDecimal/decimal-9627525982598323465-256           17965050          64.85
↪  ns/op
BenchmarkUint64ToDecimal/decimal-3534334367214237261-256           18268488          63.93
↪  ns/op
BenchmarkUint64ToDecimal/decimal-7497468244883513247-256           18552255          64.63
↪  ns/op
BenchmarkUint64ToDecimal/decimal-12769259138917390016-256          18472101          64.27
↪  ns/op
BenchmarkUint64ToHex/hex-12663951391086054483-256                  21078548          55.90
↪  ns/op
BenchmarkUint64ToHex/hex-9832119173398632219-256                   21120962          55.53
↪  ns/op
BenchmarkUint64ToHex/hex-5571782338101878760-256                   21253447          55.95
↪  ns/op
BenchmarkUint64ToHex/hex-1926012586526624009-256                   21277513          54.71
↪  ns/op
BenchmarkUint64ToHex/hex-9627525982598323465-256                   21430350          54.63
↪  ns/op
BenchmarkUint64ToHex/hex-3534334367214237261-256                   21952039          54.55
↪  ns/op
BenchmarkUint64ToHex/hex-7497468244883513247-256                   21404242          54.55
↪  ns/op
BenchmarkUint64ToHex/hex-12769259138917390016-256                  21543303          55.15
↪  ns/op
BenchmarkDecimalToUint64/decimal-12663951391086054483-256          33713239          35.02
↪  ns/op
BenchmarkDecimalToUint64/decimal-9832119173398632219-256           35647128          33.48
↪  ns/op
BenchmarkDecimalToUint64/decimal-5571782338101878760-256           35466343          33.57
↪  ns/op
BenchmarkDecimalToUint64/decimal-1926012586526624009-256           35352523          33.48
↪  ns/op
BenchmarkDecimalToUint64/decimal-9627525982598323465-256           35713856          34.22
↪  ns/op
BenchmarkDecimalToUint64/decimal-3534334367214237261-256           35713292          33.56
↪  ns/op
BenchmarkDecimalToUint64/decimal-7497468244883513247-256           35828391          33.62
↪  ns/op
BenchmarkDecimalToUint64/decimal-12769259138917390016-256          33735440          34.99
↪  ns/op
BenchmarkHexToUint64/hex-12663951391086054483-256                  35824147          34.08
↪  ns/op
BenchmarkHexToUint64/hex-9832119173398632219-256                   33891504          35.37
↪  ns/op
BenchmarkHexToUint64/hex-5571782338101878760-256                   37182722          36.75
↪  ns/op
BenchmarkHexToUint64/hex-1926012586526624009-256                   34398735          32.55
↪  ns/op
BenchmarkHexToUint64/hex-9627525982598323465-256                   38007918          34.69
↪  ns/op
BenchmarkHexToUint64/hex-3534334367214237261-256                   36652592          31.60
↪  ns/op
BenchmarkHexToUint64/hex-7497468244883513247-256                   34929393          33.15
↪  ns/op
BenchmarkHexToUint64/hex-12769259138917390016-256                  32042349          36.48
```

Berachain: Lots to unpack here. Acknowledged.

Spearbit: Acknowledged.

5.4.10 `GweiFromWei` modifies input wei amount to gwei

Severity: Informational

Context: mod/primitives/pkg/math/u64.go#L149

Description: `GweiFromWei` takes an amount denominated in wei and returns the amount denominated in gwei. The issue is that is also converts the input wei amount to gwei. This is both redundant and potentially confusing, as the caller may expect the input variable to retain its wei-denominated value.

The function is currently not used by any code.

- Reproducer:

```
package main

import (
    "github.com/berachain/beacon-kit/mod/primitives/pkg/math"
    "math/big"
    "fmt"
)

func main() {
    b, _ := new(big.Int).SetString("123456", 10)
    fmt.Println("b before", b.String())
    math.GweiFromWei(b)
    fmt.Println("b after", b.String())
}
```

Output:

```
b before 123456
b after 0
```

Recommendation: Suggested patch:

(This also shortens the `intToGwei` definition by setting its value only once instead of twice).

Berachain: Acknowledged. This will be considered at a later date.

Spearbit: Acknowledged.