Berachain Beaconkit Security Review

Auditors

Dtheo, Lead Security Researcher

Shotes, Lead Security Researcher

Guido Vranken, Lead Security Researcher

Sujith Somraaj, Security Researcher

Report prepared by: Lucas Goiriz

February 1, 2025

Contents

1 About Spearbit **3**

2 Introduction **3**

3 Risk classification **3**
3.1 Impact . . . 3
3.2 Likelihood . . . 3
3.3 Action required for severity levels . . . 3

4 Executive Summary **4**

5 Findings **5**
5.1 Critical Risk . . . 5
5.1.1 Validator deposits not verified against deposit contract . . . 5
5.1.2 Unvalidated `ExecutionPayload.Timestamp` can halt chain . . . 6
5.2 High Risk . . . 6
5.2.1 A malicious transaction object can trigger a Node DoS . . . 6
5.3 Medium Risk . . . 7
5.3.1 `BlobSidecars` data availability race condition . . . 7
5.3.2 Maximum number of withdrawals processed on every block . . . 8
5.3.3 `deposit` does not update account balance for existing validators . . . 9
5.3.4 `max_tx_bytes` default 1MB can be exceeded in `PrepareProposal()` . . . 10
5.3.5 Unsigned `BeaconBlockHeader` in `Sidecar` in left unvalidated . . . 10
5.3.6 Unvalidated `ProposerIndex` in `BeaconBlock` . . . 11
5.3.7 `rpc.Client` race condition in `header http.Header` field . . . 12
5.3.8 `deposit.Service` race condition in `failedBlocks` field . . . 12
5.4 Low Risk . . . 14
5.4.1 `ExecutionPayloadHeader` SSZ roundtrip failure . . . 14
5.4.2 Non-optimistic builder builds optimistically . . . 15
5.4.3 Optimistic build happening on every node at every round . . . 15
5.4.4 `ValidateBasic` function in `config.go` is redundant / incomplete . . . 16
5.4.5 `SSZMarshal` panics on nil pointers . . . 16
5.4.6 `argsToLabels()` slice index panic . . . 18
5.4.7 `NewValidatorFromDeposit()` divide-by-0 panic . . . 19
5.4.8 Gwei amount truncation in `ConvertAmount()` . . . 20
5.4.9 `gabriel-vasile/mimetype@v1.4.4` dependency contains dangerous test case . . . 20
5.4.10 Validator `deposit effectiveBalanceIncrement` inconsistent usage . . . 20
5.4.11 `Timestamp` setting is inconsistent . . . 22
5.4.12 `U64.NextPowerOfTwo()` custom panic . . . 22
5.4.13 Beacon-kit will panic on malformed jwt secret . . . 22
5.4.14 `Withdrawals.EncodeIndex()` index out of range panic . . . 23
5.4.15 `BlobSidecars.Get()` index out of range panic . . . 23
5.4.16 `hex.String` type inconsistent invariants . . . 23
5.4.17 `node-api` listens on `0.0.0.0:3500` by default . . . 24
5.4.18 `math.NewU256FromBigInt` succeeds with negative integers . . . 24
5.4.19 `ExecutionPayload` JSON roundtrip failure . . . 25
5.4.20 `ExecutionPayloadHeader`, `ExecutionPayload` JSON deserialization panic . . . 26
5.4.21 Insufficient input validation in `common.NewRootFromHex` . . . 28
5.4.22 `version.DenebPlus` usage inconsistency . . . 29
5.4.23 `ExecutionPayload` implements `MarshalJSON` on pointer receiver . . . 29
5.4.24 Inappropriate input validation in `UnmarshalText` for `ExecutionAddress` . . . 30
5.4.25 Non-determinism in `github.com/goccy/go-json` JSON parser . . . 31
5.4.26 `math.U256 UnmarshalJSON` does not reject oversized decimal input . . . 32
5.4.27 Race condition in `broker.Broker` results in panic . . . 32
5.5 Informational . . . 34

5.5.1 Go encoding/json.Unmarshal data corruption of string fields . . . 34
5.5.2 Inaccurate error message in state_processor_staking.go . . . 35
5.5.3 Notes on RangeDB DeleteRange . . . 35
5.5.4 RangeDB does not guarantee producing oversized filenames . . . 37
5.5.5 ProcessSlots() returns silently on invalid slot . . . 37
5.5.6 Unused file helpers.go . . . 38
5.5.7 Remove hardcoded bArtio chainId from addValidatorToRegistry function . . . 38
5.5.8 Possible nil check for payload in validateStatelessPayload and validateStatefulPayload functions . . . 38
5.5.9 Remove hardcoded bArtio-specific config in state_processor_genesis.go . . . 39
5.5.10 recover() pattern can isolate panic's and increase node robustness . . . 39
5.5.11 EventDispatcher usage does not guarantee matching response . . . 39
5.5.12 Missing validator for validator_status . . . 40
5.5.13 node-api input validation inconsistent with usage . . . 40
5.5.14 Inconsistent logging and error checks on ExecutionPayload retrieval . . . 41
5.5.15 SendForceHeadFCU() does not check if PayloadBuilder is disabled . . . 42
5.5.16 Two SSZ libraries used for same types . . . 42
5.5.17 Improve metrics coverage in engine implementation . . . 42
5.5.18 Potential integer truncation/UB in math.GweiFromWei . . . 43
5.5.19 Potential undefined behavior in call to math/big.Int.Uint64() in EngineClient.verifyChainIDAndConnection() . . . 44
5.5.20 math.U256 JSON roundtrip failure . . . 45
5.5.21 ticker.Stop() used without defer . . . 46
5.5.22 Redundant check for version.DenebPlus . . . 46
5.5.23 Duplicate jwt signing functions . . . 46
5.5.24 Inconsistent math.Gwei usage in BeaconState . . . 47
5.5.25 Use EthSecp256k1CredentialPrefix constant for clarity . . . 47
5.5.26 Unnecessary extra allocation in BeaconBlock.NewFromSSZ . . . 47
5.5.27 ProposerIndex has wrong type alias . . . 48
5.5.28 PrevPowerOfTwo and NextPowerOfTwo functions could be optimized . . . 48
5.5.29 Typographical Issues . . . 49
5.5.30 Resolve all TODOs for production readiness . . . 50
5.5.31 Mislabeled constants bytesPer64Bits and bytesPer256Bits . . . 50

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 10 days in total, Berachain engaged with Spearbit to review the beacon-kit protocol. In this period of time a total of **69** issues were found.

Summary

Project Name	Berachain
Repository	beacon-kit
Commit	76943e...5f46
Type of Project	Infrastructure, Node
Audit Timeline	Sep 9th to Oct 21st

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	2	2	0
High Risk	1	1	0
Medium Risk	8	8	0
Low Risk	27	15	12
Gas Optimizations	0	0	0
Informational	31	20	11
Total	**69**	**46**	**23**

5 Findings

5.1 Critical Risk

5.1.1 Validator deposits not verified against deposit contract

Severity: Critical Risk

Context: state-transition/pkg/core/state_processor_staking.go#L103

Description: During the `state-transition` period, the function `applyDeposit` at state-transition/pkg/core/state_processor_staking.go#L103 exists to validate a deposit and then apply the deposit to an updated validator set. However, it does not verify that the deposit came from the deposit contract. In fact, there is no verification that the deposits in the `BeaconBlock` correlate with anything from the `ExecutionPayload` or the node state.

Here is the full chain of where the validator updates come from:

- `ProcessBeaconBlock()` calls `executeStateTransition()`: beacon/blockchain/process.go#L66.
- `executeStateTransition()` calls `Transition()`: beacon/blockchain/process.go#L108.
- `Transition()` grabs the validatorUpdates from `ProcessSlots()`: state-transition/pkg/core/state_processor.go#L172.
- `ProcessSlots()` grabs updates from `processEpoch()`: state-transition/pkg/core/state_processor.go#L214.
- `processEpoch()` returns `processSyncCommitteeUpdates`: state-transition/pkg/core/state_processor.go#L344.
- Finally, `processSyncCommitteeUpdates()` grabs the new validators from `st.GetValidatorsByEffectiveBalance()`: state-transition/pkg/core/state_processor_committee.go#L34.
- The validators inside of `st.GetValidatorsByEffectiveBalance()` were committed to this `BeaconState` during the previous slot's `FinalizeBlock()` in the `processOperations()` → `processDeposits()` → `processDeposit()` → `applyDeposit()` → `createValidator()` → `addValidatorToRegistry()` at state-transition/pkg/core/state_processor_staking.go#L200.

Note that the validators in the `BeaconState` (`st` variable) are never validated against the `Deposit` contract event logs. There is one location where deposits are fetched directly from the `ExecutionPayload` in `fetchAndStoreDeposits()` at execution/pkg/deposit/sync.go#L75. However, these fetched deposits seem to only ever be used to build a new proposal at beacon/validator/block_builder.go#L286-L289 and are never used to validate.

If the block passes the proposal, then once it is finalized, it will execute the same code in the `state-transition` and grab the validator udpates at beacon/blockchain/process.go#L66. This means that a proposer could craft arbitrary deposits in the `BeaconBlock` without changing anything in the `ExecutionPayload`, increasing validator balances or creating new validators for free.

There is currently an EIP, EIP-6110, that changes validator deposits on Ethereum to be included in the `ExecutionPayload` by utilizing emitted `Deposit` events. The current deposit implementation on Beacon-Kit is very similar to this EIP but does not implement the execution layer side (because Beacon-Kit is not an EL client).

The current Beacon-Kit deposit contract applies deposits by emitting the `Deposit` event. After applying the `ExecutionPayload` during the state transition, the deposits in the `BeaconBlock` need to be validated against the events emitted in the `ExecutionPayload`. EIP-6110 suggests performing this validation by including the `Deposit` event log receipts in the actual `ExecutionPayload` as a new field `deposit_receipts`, which is validated by the EL client to correspond with `Deposit` events. However, Beacon-Kit is not at liberty to make such changes as it does not include an implementation of the EL.

Recommendation: Beacon-Kit currently already includes a mechanism for grabbing deposits from the EL via RPC using `ReadDeposits()` at execution/pkg/deposit/contract.go#L77. The `ReadDeposits()` query happens after a block has been finalized since the EL will have received the `forkchoiceUpdate`. These `Deposit` events are stored in the `DepositStore` at beacon/validator/block_builder.go#L286-L289 and are currently only retrieved when preparing a new proposal. These same `Deposit` events can be retrieved from the `DepositStore` during `ProcessProposal()` and compared against.

Thus, during Beacon-Kit's implementation of `ProcessProposal()` at height `N`, the deposits in the `BeaconBlock` can be validated against the `Deposit` events retrieved at height `N-1` from `ReadDeposits()`.

Berachain: Addressed in PR 2115 and PR 2296.

Spearbit: Fix verified.

5.1.2 Unvalidated `ExecutionPayload.Timestamp` can halt chain

Severity: Critical Risk

Context: state-transition/pkg/core/state_processor_payload.go#L36

Description: The `Timestamp` field in `ExecutionPayload` at consensus-types/pkg/types/payload.go#L60 is never validated during a cometBFT `ProcessProposal()` request.

A malicious proposer could propose a `BeaconBlock` containing an `ExecutionPayload.Timestamp` in the future (perhaps `MaxUint64`). The execution client will only check that the timestamp is after the timestamp in the parent header, as seen in the implementation in go-ethereum at consensus/beacon/consensus.go#L243-L245. Since beacon-kit does not validate the timestamp, the proposed block would pass consensus and set the timestamp to `MaxUint64`. When proposing new blocks, it would be impossible to propose a block with a timestamp greater than `MaxUint64`, and the chain will halt.

In the general case too, timestamps must be a somewhat reliable indicator of the actual time the `ExecutionPayload` gets executed, due to events, blocks, and other third-party applications that rely on accurate timestamps from the chain.

Recommendation: The `ExecutionPayload.Timestamp` should be validated with the timestamp in the cometBFT `ProcessProposalRequest` in consensus/pkg/cometbft/service/abci.go#L268. The `ExecutionPayload.Timestamp` should be greater than the previous block and less than the cometBFT time plus the minimum slot time. This will require some minor refactoring, as the `ProcessProposalRequest` and `ExecutionPayload.Timestamp` are currently not accessible in the same location.

Berachain: Fixed in PRs 2095 and 2096.

Spearbit: Fix verified.

5.2 High Risk

5.2.1 A malicious transaction object can trigger a Node DoS

Severity: High Risk

Context: github.com/itsdevbear/ssz/hasher.go#L132

Description: Berachain uses forked versions of various ssz libraries including `https://github.com/itsdevbear/ssz`, which is forked from `https://github.com/karalabe/ssz`. This library implements `HashStaticBytes()`:

```
func HashStaticBytes[T commonBytesLengths](h *Hasher, blob *T) {
    // The code below should have used `blob[:]`, alas Go's generics compiler
    // is missing that (i.e. a bug): https://github.com/golang/go/issues/51740
    h.hashBytes(unsafe.Slice(&(*blob)[0], len(*blob)))
}
```

This function takes in any type whose underlying type is that of `commonBytesLengths`. Which is defined in the upstream repository as any type whose underlying type is a byte array of the following sizes here:

```
type commonBytesLengths interface {
    // fork | nonce | address | verkle-stem | hash | pubkey | committee | signature | bloom
    ~[4]byte | ~[8]byte | ~[20]byte | ~[31]byte | ~[32]byte | ~[48]byte | ~[64]byte | ~[96]byte |
↪   ~[256]byte
}
```

An issue was introduced when a change to this interface was made in the berachain-used fork that added the variable byte array size `~[]byte`:

```
type commonBytesLengths interface {
    // fork | address | verkle-stem | hash | pubkey | committee | signature | bloom | blob & tx
    ~[4]byte | ~[20]byte | ~[31]byte | ~[32]byte | ~[48]byte | ~[64]byte | ~[96]byte | ~[256]byte |
↪   ~[131072]byte | ~[]byte
}
```

This addition allows for any function that accepts this interface type to accept an array of any size, including a 0-length array, introducing a `panic: runtime error: index out of range [0] with length 0` when the 0 indexing of the blob object is done. This is exposed in the dependencies of transaction processing in Berachain due to the way `Transactions` are deserialized when they are hashed. This allows for an attacker to create a malicious transaction that can crash the beacon-kit process depending on how its execution client sanitizes these transactions.

This is triggerable from an untrusted buffer when transactions are hashed in /mod/engine-primitives/pkg/engine-primitives/transactions.go and /mod/engine-primitives/pkg/engine-primitives/transactions_bartio.go and was discovered when fuzzing berachain's transaction processing flows.

Recommendation: Do not modify the `commonBytesLengths` interface definition to include variable length byte arrays. If a variable length array is needed then define another byte length type.

Berachain: Fixed in commit 8a041731.

Spearbit: Fix verified.

5.3 Medium Risk

5.3.1 `BlobSidecars` data availability race condition

Severity: Medium Risk

Context: beacon/blockchain/process.go#L74-L78

Description: During the `FinalizeBlock()` call from CometBFT, Beacon-Kit will begin processing on two different event handlers at abci.go#L345-L356, `async.FinalBeaconBlockReceived` and `async.FinalSidecarsReceived`. These call `handleBeaconBlockFinalization()` and `handleFinalSidecarsReceived()` respectively. These functions run in parallel with each other. This presents the following race:

- In `handleFinalSidecarsReceived()`, the sidecars will be written to the `AvailabilityStore` at da/pkg/blob/processor.go#L129.
- In `handleBeaconBlockFinalization()`, the sidecars are checked to see if they have been included in the `AvailabilityStore` by calling `IsDataAvailable()` at beacon/blockchain/process.go#L74-L78.

The `AvailabilityStore` is being written to and read at the same time. Thus it can be non-deterministic whether or not the `IsDataAvailable()` check will pass during `FinalizeBlock()`. If the data is not available when the check happens, then the validator will return error for `FinalizeBlock()`.

Note: This `IsDataAvailable()` check is absolutely necessary due to the fact that the block proposal is the only source of distributing `BlobSidecars`. If this check did not exist, a malicious proposer could purposefully not include a `BlobSidecar` that correlates with a `KZGCommitment` in the `BeaconBlock`. This would result in the chain continuing on without ever having the blob data made available. Though in my opinion, this check should happen in `ProcessProposal()`, so that an invalid proposal may be punished properly in the future.

Recommendation: The check that beacon/blockchain/process.go#L74-L78 should await the return of `handleFinalSidecarsReceived()`. To allow for some partial parallel processing (processing the sidecars at the same time as processing the state transition), this check could await a dispatcher event for a new event like `async.BlobDataAvailable`.

Berachain: Fixed in commit 711ce6fd.

Spearbit: Fix verified.

5.3.2 Maximum number of withdrawals processed on every block

Severity: Medium Risk

Context: state-transition/pkg/core/state/statedb.go#L257-L262

Description: The `BeaconState.ExectedWithdrawals()` function at state-transition/pkg/core/state/statedb.go#L192 is used to grab a list of all of the validators that are withdrawable, as defined in the Ethereum consensus specs:

```
def get_expected_withdrawals(state: BeaconState) -> Sequence[Withdrawal]:
    # ...
    for _ in range(bound):
        # ...
        if is_fully_withdrawable_validator(validator, balance, epoch):
            withdrawals.append(...)
            withdrawal_index += WithdrawalIndex(1)
        elif is_partially_withdrawable_validator(validator, balance):
            withdrawals.append(...)
            withdrawal_index += WithdrawalIndex(1)
        # ...
    return withdrawals
```

There is one major difference in the beacon-kit implementation at state-transition/pkg/core/state/statedb.go#L190-L282:

```
func ExpectedWithdrawals() ([]WithdrawalT, error) {
    // ...
    for range bound {
        // ...
        // Set the amount of the withdrawal depending on the balance of the
        // validator.
        if validator.IsFullyWithdrawable(balance, epoch) {
            amount = balance
        } else if validator.IsPartiallyWithdrawable(
            balance, math.Gwei(s.cs.MaxEffectiveBalance()),
        ) {
            amount = balance - math.Gwei(s.cs.MaxEffectiveBalance())
        }
        // ...
        withdrawal = withdrawal.New(...)
        withdrawals = append(withdrawals, withdrawal)
        // ...
}
```

The beacon-kit implementation will append the withdrawal even if it is neither `IsFullyWithdrawable()` nor `IsPartiallyWithdrawable()`. It will create a new withdrawal with `amount = 0`.

This means that in every single `BeaconBlock`, the number of withdrawals will always be equal to `MaxValidatorsPerWithdrawalsSweep()`. This incurs a somewhat significant cost in computation as well as the amount of data included in every `BeaconBlock`.

Recommendation: Only append a new withdrawal when the validator passes one of the two withdrawal checks (`IsFullyWithdrawable()` or `IsPartiallyWithdrawable()`):

```
if validator.IsFullyWithdrawable(balance, epoch) {
    withdrawals = append(withdrawals, withdrawal.New(
        math.U64(withdrawalIndex),
        validatorIndex,
        withdrawalAddress,
        balance,
    ))
    withdrawalIndex++
} else if validator.IsPartiallyWithdrawable(
    balance, math.Gwei(s.cs.MaxEffectiveBalance()),
) {
    withdrawals = append(withdrawals, withdrawal.New(
        math.U64(withdrawalIndex),
        validatorIndex,
        withdrawalAddress,
        balance-math.Gwei(s.cs.MaxEffectiveBalance()),
    ))
    withdrawalIndex++
}
```

Berachain: Addressed in PR 2110.

Spearbit: Fix verified.

5.3.3 `deposit` does not update account balance for existing validators

Severity: Medium Risk

Context: state-transition/pkg/core/state_processor_staking.go#L117-L119

Description: The validator store in Beacon-Kit keeps track of two different "*balance*" values for each validator:

1. `EffectiveBalance`: The effective balance of the validator, stored in the `KVStore` at storage/pkg/beacondb/kvstore.go#L91 via the `Validator` struct at consensus-types/pkg/types/validator.go#L52. This value has a maximum of `chainspec.MaxEffectiveBalance()`. This is updated by updating the `EffectiveBalance` field in the validator struct and then pushing the modified validator to the store via `st.UpdateValidatorAtIndex(idx, val)`.
2. `Balance`: The actual full balance of the validator, stored in a separate balance sheet in the `KVStore` at storage/pkg/beacondb/kvstore.go#L95. This is updated by writing the new balance value to the store via `st.IncreaseBalance(idx, dep.GetAmount())`.

Note that these values are tracked pretty much fully-independently.

When applying a validator deposit at state-transition/pkg/core/state_processor_staking.go#L117-L119, only the `EffectiveBalance` update is applied:

```
val.SetEffectiveBalance(min(val.GetEffectiveBalance()+dep.GetAmount(),
    math.Gwei(sp.cs.MaxEffectiveBalance())))
return st.UpdateValidatorAtIndex(idx, val)
```

There is no call to `st.IncreaseBalance()` to add the deposit amount to the validator balance. This results in a full loss of the funds from the `deposit` that updated the balance of the existing validator. This also results in a mismatch between the validator balance and the `EffectiveBalance`, meaning the `EffectiveBalance` could be larger than the actual balance.

Recommendation: Update the balance of the existing validator with a call to `st.IncreaseBalance()`.

Berachain: Addressed in PR 2111.

Spearbit: Fix verified.

5.3.4 `max_tx_bytes` default 1MB can be exceeded in `PrepareProposal()`

Severity: Medium Risk

Context: consensus/pkg/cometbft/service/abci.go#L218-L262, testing/networks/80084/config.toml#L368

Description: In CometBFT ABCI++, the proposed transactions from `PrepareProposal()` in total byte size cannot exceed `req.max_tx_bytes`. In one of the configs testing/networks/80084/config.toml#L368 (80084 is for the `bArtio` testnet), the `max_tx_bytes` seems to be set to `1048576` (1MB) which is the default for `CometBFT`.

However, both the `BeaconBlock` and the `BlobSidecars` can exceed this limit individually.

- Maximum `BeaconBlock` size - constrained by the `ExecutionPayload` gas limit (15 million average, 30 million max). An `ExecutionPayload` filled with 0's can be up to a maximum block of 7,500,000 bytes. So with headers and stuff, it's around `7.5 MB`.
- Maximum `Sidecars` size - 128KB per blob and `MAX_BLOBS_PER_BLOCK` is 6, so `(128KB * 6) + negligible header size = 768 KB`.

If the size of the `BeaconBlock` and `Sidecars` together exceed `req.max_tx_bytes`, then the proposal will not be accepted. In the current state, all honest validators using the current codebase will never prune transactions and thus if a proposed `BeaconBlock` plus `BlobSidecars` exceeds `req.max_tx_bytes` (1MB) in size the proposal will be rejected. Presumably, other honest proposers will try to craft the same proposed block or similar, potentially halting the chain entirely with all of the proposals being rejected.

This 1MB limit isn't super hard to hit. If a proposal has `MAX_BLOBS_PER_BLOCK` (6), then that is automatically ~768KB of 1MB used. Then the `BeaconBlock` would only have to exceed 280KB to hit this limit. This could easily happen.

Recommendation: Set the `max_tx_bytes` configuration in CometBFT high enough such that it is impossible to ever encounter a tuple of (`BeaconBlock`, `BlobSidecars`) that exceeds the byte limit:

```
max_tx_bytes = 104857600 // 100 MB is impossible to hit
```

This is allowable for two major reasons:

1. The CometBFT mempool is unused (see testing/networks/80084/config.toml#L365).
2. The execution client will enforce the max block gas limit which will constrain the payload to a finite size.

Berachain: Initially addressed in PR 2117. Further fixes addressed in PR 2362.

Spearbit: Fixes verified.

5.3.5 Unsigned `BeaconBlockHeader` in `Sidecar` in left unvalidated

Severity: Medium Risk

Context: consensus/pkg/cometbft/service/middleware/abci.go#L136

Description: The `BlobSidecar` struct at da/pkg/types/sidecar.go#L47 contains the `BeaconBlockHeader` for which the blob is being included. This is different than the expected `SignedBeaconBlockHeader` in the Ethereum consensus specs:

```
  type BlobSidecar struct {
      Index uint64
      Blob eip4844.Blob
      KzgCommitment eip4844.KZGCommitment
      KzgProof eip4844.KZGProof
-     SignedBeaconBlockHeader <unimplemented type>
+     BeaconBlockHeader *types.BeaconBlockHeader
      InclusionProof []common.Root
  }
```

This issue was originally brought up in `https://github.com/berachain/beacon-kit/issues/841` and I believe it remains an issue. Here are some thoughts on the problem:

- On Ethereum L1, `Sidecars` are gossiped separately from `BeaconBlocks`. This means the `Sidecars` *must* contain the `signed_beacon_block_header` so that it can correlate the `proposer` of the `BeaconBlock` with the `proposer` of the `Sidecar`.
- In Beacon-Kit, `BeaconBlocks` and `Sidecars` are gossiped together by cometBFT consensus. They are validated together, voted on together, and processed together. This means that Beacon-Kit has the privilege of being able to correlate directly a `BeaconBlock` with its `Sidecars`.
- That being said, Beacon-Kit does not check that the `BeaconBlockHeader` in the `Sidecars` correlates with the `BeaconBlock` that is received in the same "Proposal". This means that the proposer can set the `ProposerIndex` of each `Sidecar` to whatever they wanted.

Recommendation: The `ProposerIndex` of the `BeaconBlock` is validated against the local state. This first validation can be done as explained in the recommendation for issue titled "Unvalidated `ProposerIndex` in `BeaconBlock`".

The second part of the verification is to establish that the `BeaconBlockHeaders` in the `BlobSidecars` correlate with the headers of the proposed `BeaconBlock` (and thus verifying the `ProposerIndex` to be the same).

At first glance, this solution may seem insufficient. However, the validation's sufficiency hinges on the fact that `BlobSidecars` are included in the consensus mechanism and are voted on in direct correspondence with the `BeaconBlock` as opposed to asynchronously gossiped and validated separately as in Ethereum L1.

Berachain: Addressed in PR 2108, based on PR 2113, as well as in PR 2245.

Spearbit: Fix verified.

5.3.6 Unvalidated `ProposerIndex` in `BeaconBlock`

Severity: Medium Risk

Context: state-transition/pkg/core/state_processor.go#L421

Description: During the `ProcessProposal()` period in cometBFT, validators currently do not explicitly check to ensure that the `ProposerIndex` in the proposed `BeaconBlock` is the correct index for the proposer chosen by `cometBFT`.

Down the line, the `ProposerIndex` is almost fully verified implicitly during the `RANDAO` reveal signature check at state-transition/pkg/core/state_processor_randao.go#L73, due to the correlation of grabbing the validators public key via the `ProposerIndex`:

```
proposer, err := st.ValidatorByIndex(blk.GetProposerIndex())
// ...
signingRoot := fd.ComputeRandaoSigningRoot(
  sp.cs.DomainTypeRandao(), epoch,
)
reveal := body.GetRandaoReveal()
if err = sp.signer.VerifySignature(
    proposer.GetPubkey(),
    signingRoot[:],
    reveal,
)
```

However, this remains vulnerable. Here is an example scenario in which this check can be bypassed:

- `ProposerA` proposes and distributes a block, but due to network errors, less than the quorum amount of validators receive the block.
- cometBFT will move on and ask the next proposer for a block. It chooses `ProposerB`.
- `ProposerB` is a malicious validator and had received the previously proposed block from `ProposerA` that didn't end up getting quorom.
- `ProposerB` now sets its `ProposerIndex` to the index of `ProposerA` and also uses the same `RandaoReveal` that it saw in `ProposerA`'s proposed block. Since the proposed BeaconBlock is at the same `slot`, the `fd.ComputerRandaoSigningRoot()` is the same and the copied `RandaoReveal` is valid.

- `ProposerB` now proposes its new block, pretending to be `ProposerA` from the BeaconBlock perspective.

This could result in the penalization of `ProposerA` if there's any slashing conditions, as well as bypassing the `IsSlashed()` check at state-transition/pkg/core/state_processor.go#L421.

Recommendation: Explicitly correlate and validate the `ProposerIndex` with the proposer's public key from the cometBFT `ProcessProposalRequest`.

Berachain: Addressed in PR 2102.

Spearbit: Fix verified.

5.3.7 `rpc.Client` race condition in `header http.Header` field

Severity: Medium Risk

Context: execution/pkg/client/ethclient/rpc/client.go#L50

Description: The `rpc.Client.header` field is an `http.Header` from the go library. This is a wrapper over a map, and it is not concurrency-safe.

In the current usage, there is a race condition in the following spots:

- `rpc.Start()` is an infinite loop that will update the header with the `rpc.header.Set("Authorization", "Bearer "+token)`.
- `rpc.CallRaw()` will read the `rpc.header` value in a separate go routine.

This could result in undefined behavior that could cause invalid data or panic.

Recommendation: Protect access to the `header http.Header` field at execution/pkg/client/ethclient/rpc/client.go#L50 with a `sync.RWMutex`.

Berachain: Fixed in commit bea7c27c.

Spearbit: Fix verified.

5.3.8 `deposit.Service` race condition in `failedBlocks` field

Severity: Medium Risk

Context: execution/pkg/deposit/sync.go#L95

Description: The `s.failedBlocks` map is used concurrently in the `depositCatchupFetcher()` and in the `depositFetcher()`.

The manner in which this map is used concurrently is seemingly safe - a single map key index is never accessed concurrently, only separate key indices. This is guaranteed by the strictly increasing nature of the key value `blockNums` that comes from the finalized block event stream.

However, the go map implementation can move, reallocate, or resize underlying data during write operations and key deletion operations, and it is not safe to read/write concurrently without risk of undefined behavior.

Here is a mock program that closely imitates the deposit fetcher and results in `fatal error: concurrent map writes`:

```
package main

import (
    "context"
    "time"
)

type Service struct {
    failedBlocks map[uint64]struct{}
}
```

```go
func (s *Service) depositFetcher(ctx context.Context, blockNum uint64) {
    s.fetchAndStoreDeposits(blockNum)
}

func (s *Service) eventLoop(ctx context.Context, subFinalizedBlockNums chan uint64) {
    for {
        select {
        case <-ctx.Done():
            return
        case blockNum := <-subFinalizedBlockNums:
            s.depositFetcher(ctx, blockNum)
        }
    }

}

func (s *Service) fetchAndStoreDeposits(blockNum uint64) {
    // Fail every 5 times for testing
    if blockNum%5 == 0 {
        s.failedBlocks[blockNum] = struct{}{}
        return
    }
    delete(s.failedBlocks, blockNum)

}

func (s *Service) depositCatchupFetcher(ctx context.Context) {
    ticker := time.NewTicker(5 * time.Second)
    defer ticker.Stop()
    for {
        select {
        case <-ctx.Done():
            return
        case <-ticker.C:
            if len(s.failedBlocks) == 0 {
                continue
            }
            for blockNum := range s.failedBlocks {
                s.fetchAndStoreDeposits(blockNum)
            }
        }
    }
}

func main() {
    ctx := context.Background()
    depositService := &Service{
        make(map[uint64]struct{}),
    }
    mockFinalizedBlockNums := make(chan uint64)
    go depositService.eventLoop(ctx, mockFinalizedBlockNums)
    go depositService.depositCatchupFetcher(ctx)
    for i := 0; i < 1000000000; i++ {
        mockFinalizedBlockNums <- uint64(i)
    }
}
```

Recommendation: Use a `sync.RWMutex` to protect `s.failedBlocks` or make `s.failedBlocks` into a `sync.Map`.

Berachain: Fixed in commit cde88287.

Spearbit: Fix verified.

5.4 Low Risk

5.4.1 `ExecutionPayloadHeader` SSZ roundtrip failure

Severity: Low Risk

Context: consensus-types/pkg/types/payload_header.go#L36

Description: The following logic

```
package main

import (
    consensustypes "github.com/berachain/beacon-kit/mod/consensus-types/pkg/types"
    "github.com/berachain/beacon-kit/mod/primitives/pkg/encoding/json"
    "fmt"
)

func main() {
    serialized_json := `{"parentHash":"0x0000000000000000000000000000000000000000000000000000000000000000
↪  00","feeRecipient":"0x0000005B0000000000000000000000000000000","stateRoot":"0x0000000000000000000000
↪  0000000000000000000000000000000000000000","receiptsRoot":"0x0000000000000000000000000000000000000000
↪  00000000000000000000000000","logsBloom":"0x0000000000000000000000000000000000000000000000000000000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000","prevRandao":"0x000000000000000000
↪  0000000000000000000000000000000000000000","blockNumber":"0x0","gasLimit":"0x0","gasUsed":"0x0
↪  ","timestamp":"0x0","extraData":"0x0000000000000000000000000000000000000000000000000000000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000","baseFeePerGas":"199","bloc
↪  kHash":"0x00000000000000000000000000000000000000003bfffffff70000000000","transactionsRoot":"0x0
↪  0000000000000000000000000000000000000000000000000000000000000000","withdrawalsRoot":"0x0000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000","blobGasUsed":"0x0","excecsBlobGas":"0x0"}`
    var v1 consensustypes.ExecutionPayloadHeader

    if err := json.Unmarshal([]byte(serialized_json), &v1); err != nil {
        fmt.Println("json.Unmarshal failed", err)
        return
    }

    serialized_ssz, err := v1.MarshalSSZ()
    if err != nil {
        fmt.Println("MarshalSSZ failed", err)
        return
    }

    v2 := new(consensustypes.ExecutionPayloadHeader)
    err = v2.UnmarshalSSZ(serialized_ssz)
    if err != nil {
        fmt.Println("UnmarshalSSZ failed", err)
        return
    }
}
```

Prints:

> UnmarshalSSZ failed ssz: maximum item size exceeded: decoded 67, max 32

Recommendation: Consider checking the other serialization findings as this issue might be related:

- `ExecutionPayload` JSON roundtrip failure
- `ExecutionPayloadHeader`, `ExecutionPayload` JSON deserialization panic

- `ExecutionPayload` implements `MarshalJSON` on pointer receiver

Currently I am time-constrained (last day of the audit), the fuzzer found this overnight, for now I'll leave this here as-is without analysis but will revisit this issue if I have time left.

Berachain: Acknowledged. This will be considered in the future.

Spearbit: Acknowledged.

5.4.2 Non-optimistic builder builds optimistically

Severity: Low Risk

Context: beacon/blockchain/execution_engine.go#L47-L51

Description: After finalizing a block, Beacon-Kit has the following check when sending the new `engine_forkchoiceUpdatedV3` to the EL client:

```
if !s.shouldBuildOptimisticPayloads() && s.localBuilder.Enabled() {
    s.sendNextFCUWithAttributes(ctx, st, blk, lph)
} else {
    s.sendNextFCUWithoutAttributes(ctx, blk, lph)
}
```

If the node is not an optimistic builder, `s.shouldBuildOptimisticPayloads()` returns `false`. Then `sendNextFCUWithAttributes()` is called which sends an `engine_forkchoiceUpdatedV3` containing `PayloadAttributes`. This triggers a build on the EL client for the next slot.

This means that every single node, even if it has optimistic builds disabled, is requesting optimistic builds on every finalized block.

Recommendation: Remove the `sendNextFCUWithAttributes()` function entirely. In `sendPostBlockFCU()`, Beacon-Kit should only send an `engine_forkchoiceUpdatedV3` request without attributes. This is because an optimistic client has already requested an optimistic build in `handleOptimisticPayloadBuild()`, and the non-optimistic client does not need to request an optimistic build in the first place.

This function should only be using `engine_forkchoiceUpdatedV3` to notify the EL client of the new head.

Berachain: Fixed in PR 2240.

Spearbit: Fix verified.

5.4.3 Optimistic build happening on every node at every round

Severity: Low Risk

Context: beacon/blockchain/receive.go#L140

Description: From my understanding, the `s.optimisticPayloadBuilds` in Beacon-Kit determines if we should send an `engine_forkchoiceUpdatedV2` containing `PayloadAttributes` to have the EL client start building a new ExecutionPayload for the next slot. This is useful for a validator that is going to require a built ExecutionPayload when it becomes the proposer in the next round. That being said, every single validator in beacon-kit is optimistically building a payload every single slot, even if they are not going to be the next proposer. This is a *ton* of computation that doesn't need to happen.

Recommendation: In `cometBFT`, the next validator that is going to be the proposer is determined by a deterministic process that I believe can be calculated ahead of time by just knowing the current validator set. This `shouldBuildOptimisticPayloads()` function should determine if the local validator is going to be the proposer in the next slot. Otherwise it should not build the optimistic payload.

Berachain: Acknowledged. A fix for this is being considered.

Spearbit: Acknowledged.

5.4.4 `ValidateBasic` function in `config.go` is redundant / incomplete

Severity: Low Risk

Context: mod/config/pkg/config/config.go#L119

Description: Per inline documentation, the `ValidateBasic` function in `config.go` should return an error if the min-gas-prices field is empty. However, no field is called `min-gas-price` in the input passed to the function.

Moreover, the existing logic validates the snapshot interval against the pruning type, which is also commented out, leaving the function redundant.

Recommendation: Consider implementing the checks accurately if intended to validate the snapshot interval based on the pruning type (or) remove the function to improve performance.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.4.5 `SSZMarshal` panics on nil pointers

Severity: Low Risk

Context: mod/cli/pkg/commands/genesis/types/validators.ssz.go#L12, mod/da/pkg/types/sidecar.go#L131, mod/da/pkg/types/sidecars.go#L110

Description: The SSZ encoder will panic on nil pointers:

```
package main

import (
    "github.com/berachain/beacon-kit/mod/primitives/pkg/encoding/json"
    clitypes "github.com/berachain/beacon-kit/mod/cli/pkg/commands/genesis/types"
    datypes "github.com/berachain/beacon-kit/mod/da/pkg/types"
)

func main() {
    /*
    panic: runtime error: invalid memory address or nil pointer dereference
    [signal SIGSEGV: segmentation violation code=0x1 addr=0x0 pc=0xb1947c]

    goroutine 1 [running]:
    github.com/berachain/beacon-kit/mod/consensus-types/pkg/types.(*Validator).DefineSSZ(0x0,
↪   0xc000013ea8)
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/consensus-types/pkg/types/validator.go:133
↪   +0x1c
    github.com/karalabe/ssz.EncodeToBytes({0xc00029b100, 0x79, 0x79}, {0xf2cb00, 0x0})
        /home/jhg/poc-beacon-kit-json-ssz/go/packages/pkg/mod/github.com/itsdevbear/ssz@v0.0.0-20240729
↪   201410-1a53beff08cb/ssz.go:115
↪   +0x410
    github.com/berachain/beacon-kit/mod/consensus-types/pkg/types.(*Validator).MarshalSSZ(...)
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/consensus-types/pkg/types/validator.go:151
    github.com/berachain/beacon-kit/mod/consensus-types/pkg/types.(*Validator).MarshalSSZTo(0x0,
↪   {0xc00029ac80, 0x4, 0x7d})
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/consensus-types/pkg/types/validator.go:166
↪   +0x59
    github.com/berachain/beacon-kit/mod/cli/pkg/commands/genesis/types.(*ValidatorsMarshaling).MarshalS
↪   SZTo(0xc000013d88, {0xc00029ac80?, 0xc000013d88?,
↪   0xc0003d64a8?})
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/cli/pkg/commands/genesis/types/validators.ssz.
↪   go:30
↪   +0x1dd
    github.com/ferranbt/fastssz.MarshalSSZ({0xf30cf0, 0xc000013d88})
```

```
        /home/jhg/poc-beacon-kit-json-ssz/go/packages/pkg/mod/github.com/itsdevbear/fastssz@v0.0.0-2024
↪  0731164358-a354a31813e6/encode.go:13
↪  +0x56
    github.com/berachain/beacon-kit/mod/cli/pkg/commands/genesis/types.(*ValidatorsMarshaling).MarshalS
↪  SZ(...)

↪  /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/cli/pkg/commands/genesis/types/validators.ssz.go:13
    main.main()
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/poc/x.go:14 +0x9e
    */
    {
        obj := clitypes.ValidatorsMarshaling{}
        err := json.Unmarshal([]byte(`{"validators": [null]}`), &obj)
        if err != nil { return }
        obj.MarshalSSZ()
    }

    /*
    panic: runtime error: invalid memory address or nil pointer dereference
    [signal SIGSEGV: segmentation violation code=0x1 addr=0x0 pc=0xb1585c]

    goroutine 1 [running]:
    github.com/berachain/beacon-kit/mod/consensus-types/pkg/types.(*BeaconBlockHeader).DefineSSZ(0x0,
↪  0xc00053d830)
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/consensus-types/pkg/types/header.go:106 +0x1c
    github.com/karalabe/ssz.EncodeStaticObject(...)
        /home/jhg/poc-beacon-kit-json-ssz/go/packages/pkg/mod/github.com/itsdevbear/ssz@v0.0.0-20240729
↪  201410-1a53beff08cb/encoder.go:276
    github.com/karalabe/ssz.DefineStaticObject[...](0x0?, 0x0?)
        /home/jhg/poc-beacon-kit-json-ssz/go/packages/pkg/mod/github.com/itsdevbear/ssz@v0.0.0-20240729
↪  201410-1a53beff08cb/codec.go:205
↪  +0x31
    github.com/berachain/beacon-kit/mod/da/pkg/types.(*BlobSidecar).DefineSSZ(0xc000800000,
↪  0xc00053d830)
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/da/pkg/types/sidecar.go:115 +0xaa
    github.com/karalabe/ssz.EncodeToBytes({0xc000822000, 0x201d8, 0x201d8}, {0xf25e20, 0xc000800000})
        /home/jhg/poc-beacon-kit-json-ssz/go/packages/pkg/mod/github.com/itsdevbear/ssz@v0.0.0-20240729
↪  201410-1a53beff08cb/ssz.go:115
↪  +0x410
    github.com/berachain/beacon-kit/mod/da/pkg/types.(*BlobSidecar).MarshalSSZ(...)
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/da/pkg/types/sidecar.go:133
    main.main()
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/poc/x.go:22 +0x97
    */
    {
        obj := datypes.BlobSidecar{}
        err := json.Unmarshal([]byte("{}"), &obj)
        if err != nil { return }
        obj.MarshalSSZ()
    }

    /*
    panic: runtime error: invalid memory address or nil pointer dereference
    [signal SIGSEGV: segmentation violation code=0x1 addr=0x0 pc=0xb1585c]

    goroutine 1 [running]:
    github.com/berachain/beacon-kit/mod/consensus-types/pkg/types.(*BeaconBlockHeader).DefineSSZ(0x0,
↪  0xc00052d908)
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/consensus-types/pkg/types/header.go:106 +0x1c
    github.com/karalabe/ssz.EncodeStaticObject(...)
        /home/jhg/poc-beacon-kit-json-ssz/go/packages/pkg/mod/github.com/itsdevbear/ssz@v0.0.0-20240729
↪  201410-1a53beff08cb/encoder.go:276
```

```
    github.com/karalabe/ssz.DefineStaticObject[...](0x47921a?, 0xb151bd?)
        /home/jhg/poc-beacon-kit-json-ssz/go/packages/pkg/mod/github.com/itsdevbear/ssz@v0.0.0-20240729
↪   201410-1a53beff08cb/codec.go:205
↪   +0x31
    github.com/berachain/beacon-kit/mod/da/pkg/types.(*BlobSidecar).DefineSSZ(0xc000800000,
↪   0xc00052d908)
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/da/pkg/types/sidecar.go:115 +0xaa
    github.com/karalabe/ssz.EncodeSliceOfStaticObjectsContent[...](...)
        /home/jhg/poc-beacon-kit-json-ssz/go/packages/pkg/mod/github.com/itsdevbear/ssz@v0.0.0-20240729
↪   201410-1a53beff08cb/encoder.go:603
    github.com/karalabe/ssz.DefineSliceOfStaticObjectsContent[...](0x0?, 0x0?, 0x143e100?)
        /home/jhg/poc-beacon-kit-json-ssz/go/packages/pkg/mod/github.com/itsdevbear/ssz@v0.0.0-20240729
↪   201410-1a53beff08cb/codec.go:438
↪   +0x6b
    github.com/berachain/beacon-kit/mod/da/pkg/types.(*BlobSidecars).DefineSSZ(0xc00052d758,
↪   0xc00052d908)
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/da/pkg/types/sidecars.go:98 +0x49
    github.com/karalabe/ssz.EncodeToBytes({0xc000822000, 0x201dc, 0x201dc}, {0xf25e20, 0xc00052d758})
        /home/jhg/poc-beacon-kit-json-ssz/go/packages/pkg/mod/github.com/itsdevbear/ssz@v0.0.0-20240729
↪   201410-1a53beff08cb/ssz.go:118
↪   +0x2fd
    github.com/berachain/beacon-kit/mod/da/pkg/types.(*BlobSidecars).MarshalSSZTo(...)
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/da/pkg/types/sidecars.go:118
    github.com/berachain/beacon-kit/mod/da/pkg/types.(*BlobSidecars).MarshalSSZ(0xc00052d758)
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/da/pkg/types/sidecars.go:112 +0x4e
    main.main()
        /home/jhg/poc-beacon-kit-json-ssz/beacon-kit/mod/poc/x.go:28 +0x98
    */
    {
        obj := datypes.BlobSidecars{}
        err := json.Unmarshal([]byte(`{"Sidecars":[{}]}`), &obj)
        if err != nil { return }
        obj.MarshalSSZ()
    }
}
```

Recommendation: It might be feasible to change the ssz library to not call `DefineSSZ` on nil pointers (return error if it is nil). It would eliminate a whole class of potential vulnerabilities.

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

Spearbit: Acknowledged.

5.4.6 `argsToLabels()` slice index panic

Severity: Low Risk

Context: mod/node-core/pkg/components/metrics/sink.go#L76-L77

Description: `argsToLabels()` takes a "*variadic parameter*" input, allowing a varying number of string arguments to be provided to the function. While the function will accept any number of inputs, it can only gracefully handle even numbers of arguments:

```
func argsToLabels(args ...string) []metrics.Label {
    labels := make([]metrics.Label, len(args)/2)
    for i := 0; i < len(args); i += 2 {
        labels[i/2] = metrics.Label{
        Name:  args[i],
        Value: args[i+1],
        }
    }
    return labels
}
```

If the function is called with an odd number of inputs then the finally iteration of the `for` loop above will `panic: runtime error: index out of range [x] with length y`. This does not appear to be vulnerable in beacon-kit's current form but it is possible that a vulnerable scenario has been overlooked or may be introduced in the future. There are currently 41 reachable call stacks for this function when analyzing the call hierarchy 2 functions above.

Recommendation: It is recommended to either return an error or append an empty string element to `args` to prevent a `panic` if this function is ever called with an odd number of inputs. It might also be good to consider its behavior when calling with zero arguments as it will currently return an empty map.

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

Spearbit: Acknowledged.

5.4.7 `NewValidatorFromDeposit()` divide-by-0 panic

Severity: Low Risk

Context: mod/consensus-types/pkg/types/validator.go#L87-L90

Description: `NewValidatorFromDeposit()` calculates `EffectiveBalance` as the minimum of `maxEffectiveBalance` and `amount-amount%effectiveBalanceIncrement`.

```
EffectiveBalance: min(
    amount-amount%effectiveBalanceIncrement,
    maxEffectiveBalance,
),
```

If `effectiveBalanceIncrement` is ever set to zero then beacon-kit will `panic: runtime error: integer divide by zero`. This currently only appears to be reachable via `GetGenesisValidatorRootCmd()` and does not appear to be something that an attacker could trigger.

Recommendation: Check that `effectiveBalanceIncrement` is not zero before using it as the divisor, alternatively use `recover()` to gracefully catch all panics in the `Commands()` workflow.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.4.8 Gwei amount truncation in `ConvertAmount()`

Severity: Low Risk

Context: mod/cli/pkg/utils/parser/validator.go#L74

Description: If the value passed to `ConvertAmount` represents an integer that cannot be represented as a uint64 (`< 0` or `>= 2^64`) then it will be truncated without raising an error.

Recommendation:

```
diff --git a/mod/cli/pkg/utils/parser/validator.go b/mod/cli/pkg/utils/parser/validator.go
index 604eb71d5..db32f01e0 100644
--- a/mod/cli/pkg/utils/parser/validator.go
+++ b/mod/cli/pkg/utils/parser/validator.go
@@ -71,6 +71,9 @@ func ConvertAmount(amount string) (math.Gwei, error) {
        if !ok {
                return 0, ErrInvalidAmount
        }
+       if !amountBigInt.IsUint64() {
+               return 0, ErrInvalidAmount
+       }
        return math.Gwei(amountBigInt.Uint64()), nil
 }
```

Berachain: Addressed in PR 2104.

Spearbit: Fixed.

5.4.9 `gabriel-vasile/mimetype@v1.4.4` dependency contains dangerous test case

Severity: Low Risk

Context: beacond/go.mod#L124

Description: Multiple packages contain the dependency `github.com/gabriel-vasile/mimetype@v1.4.4` which has a test case that contains a tar bomb that can fill up a developers disk when running tests (see `mimetype` issue 575). The issue appears to be made by accident and does not appear to be malicious but should be removed regardless.

Recommendation: Update dependencies to use the latest version of `github.com/gabriel-vasile/mimetype` where the issue has been fixed.

Berachain: Addressed in PR 2105.

Spearbit: Fixed.

5.4.10 Validator `deposit` `effectiveBalanceIncrement` inconsistent usage

Severity: Low Risk

Context: state-transition/pkg/core/state_processor_staking.go#L117-L119, state-transition/pkg/core/state_processor_staking.go#L209

Description: In `applyDeposit()`, if we receive a deposit for a pre-existing validator, the update is as such:

```
val.SetEffectiveBalance(min(val.GetEffectiveBalance()+dep.GetAmount(),
    math.Gwei(sp.cs.MaxEffectiveBalance())))
return st.UpdateValidatorAtIndex(idx, val)
```

This ensures that `EffectiveBalance <= MaxEffectiveBalance`. Even if the deposit submitted a `dep.GetAmount()` that exceeds the `MaxEffectiveBalance`, it takes the minimum between the two. This can indeed happen because the deposit contract does not keep track of the max effective balance.

When creating a new validator, `val.New()` near the beginning of this function ends up calling `NewValidatorFromDeposit()` which will enforce the `effectiveBalanceIncrement` on the balance update:

```
EffectiveBalance: min(
    amount-amount%effectiveBalanceIncrement,
    maxEffectiveBalance,
),
```

This effective balance update is inconsistent with the pre-existing validators. New validator deposits enforce the `effectiveBalanceIncrement` while the pre-existing validator updates do not.

Recommendation: Create a helper function for the `Validator` type that applies the `EffectiveBalance` update and enforces the `effectiveBalanceIncrement`:

```
func AddEffectiveBalance(
    amount math.Gwei, effectiveBalanceIncrement math.Gwei, maxEffectiveBalance math.Gwei,
) math.Gwei {
    return min(
         amount-amount%effectiveBalanceIncrement,
        maxEffectiveBalance,
    )
}
```

This function can be used both in `Validator.NewValidatorFromDeposit()` and in `applyDeposit()`:

```
func NewValidatorFromDeposit(
    pubkey crypto.BLSPubkey,
    withdrawalCredentials WithdrawalCredentials,
    amount math.Gwei,
    effectiveBalanceIncrement math.Gwei,
    maxEffectiveBalance math.Gwei,
) *Validator {
    return &Validator{
        Pubkey:                     pubkey,
        WithdrawalCredentials:      withdrawalCredentials,
        EffectiveBalance:           AddEffectiveBalance(amount, effectiveBalanceIncrement,
↪   maxEffectiveBalance),
        Slashed:                    false,
        ActivationEligibilityEpoch: math.Epoch(constants.FarFutureEpoch),
        ActivationEpoch:            math.Epoch(constants.FarFutureEpoch),
        ExitEpoch:                  math.Epoch(constants.FarFutureEpoch),
        WithdrawableEpoch:          math.Epoch(constants.FarFutureEpoch),
    }
}
```

and

```
func applyDeposit() {
    // ...
    val.SetEffectiveBalance(
        types.AddEffectiveBalance(
            dep.GetAmount(), math.Gwei(sp.cs.EffectiveBalanceIncrement()),
↪   math.Gwei(sp.cs.MaxEffectiveBalance()),
        ),
    )
    // ...
}
```

And use this function for setting the balance.

Berachain: Addressed in PR 2103.

Spearbit: Fix verified.

5.4.11 `Timestamp` setting is inconsistent

Severity: Low Risk

Context: beacon/blockchain/execution_engine.go#L134-L136, beacon/blockchain/payload.go#L126-L130, beacon/blockchain/payload.go#L195-L199, beacon/validator/block_builder.go#L237-L241

Description: The timestamp that gets used for requesting the next execution payload is calculated independently in four different locations (the "*optimistic*" ones are building the `ExecutionPayload` for the *next* slot and not the current one):

- Optimistic: beacon/blockchain/execution_engine.go#L134-L136.
- Optimistic: beacon/blockchain/payload.go#L195-L199.
- beacon/blockchain/payload.go#L126-L130.
- beacon/validator/block_builder.go#L237-L241.

The mechanism for setting these timestamps can be inconsistent because they are based on the node's local clock.

Recommendation: Craft the `ExecutionPayload.Timestamp`, based on the cometBFT consensus clock. This will guarantee monotonicity in the timestamp and is more easily verified and enforceable.

Berachain: Fixed in PR 2094 and others.

Spearbit: Fix verified.

5.4.12 `U64.NextPowerOfTwo()` custom panic

Severity: Low Risk

Context: mod/primitives/pkg/math/pow/pow.go#L46-L48

Description: `U64.NextPowerOfTwo()` delivers a custom panic when overflowing. This function appears to be reachable from multiple calls stacks but it is unclear if it is possible to trigger this panic from an attacker controlled buffer.

Recommendation: Change to an error returning function or wrap affected call stacks in `recover()` to prevent panics like these from threatening node uptime.

Berachain: Acknowledged. Unreachable in the current code.

Spearbit: Acknowledged.

5.4.13 Beacon-kit will panic on malformed jwt secret

Severity: Low Risk

Context: mod/node-core/pkg/components/jwt_secret.go#L41-L4

Description: Beacon-kit's jwt secret parsing flow bubbles up errors to the top level `DefaultComponents()` calling function which will panic on any error. This prevents a descriptive error message from being delivered to the user and instead returns a panic call stack dump.

Recommendation: Catch the error in `ProvideJWTSecret()` and provide a clear error to the user indicating that a correctly formated jwt secret is required.

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

Spearbit: Acknowledged.

5.4.14 `Withdrawals.EncodeIndex()` index out of range panic

Severity: Low Risk

Context: mod/engine-primitives/pkg/engine-primitives/withdrawals.go#L84-L85

Description: `Withdrawals.EncodeIndex()` is vulnerable to `index out of range` panics if the method is called with an `i` value larger than the length of the withdrawals buffer. This function does not appear to be used at the moment but may be included in the future.

```
func (w Withdrawals) EncodeIndex(i int, _w *bytes.Buffer) {
    // #nosec:G703 // its okay.
    _ = w[i].EncodeRLP(_w)
}
```

Recommendation: Either implement a length check or include a description of the assumption that `i` should not exceed `len(w)` in the functions comments to prevent the introduction of a panic in future updates that may use this code.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.4.15 `BlobSidecars.Get()` index out of range panic

Severity: Low Risk

Context: mod/da/pkg/types/sidecars.go#L49-L51

Description: A `panic: runtime error: index out of range [x] with length y` can be triggered in `BlobSidecars.Get()` if it is called with a negative `int` or with an index that is higher than the number of `bs.Sidecars`. While this function appears to be unused in the current version of the codebase it might be added in the future when data availability sampling is enabled.

Recommendation: It is recommend to check the that the requested index is non-negative and is within the length of `bs.Sidecars` before indexing into the the slice.

Berachain: Fixed in PR 2313.

Spearbit: Fix verified.

5.4.16 `hex.String` type inconsistent invariants

Severity: Low Risk

Context: primitives/pkg/encoding/hex/string.go

Description: We've been finding some issues around the `hex.String` object and its type conversion methods (from/to `u64`, `u256`, `big.Int`, `bytes`).

Some observations:

- `hex.String` fundamentally is a `string` which is a text type rather than an integer type, which is what the object aims to implement. This is fundamentally a type mismatch and several different validation functions must accommodate for this.
- `hex.String` is a stateful object that is expected to maintain integrity across any number of calls to its methods. This can make it harder to reason about nth-order effects if any changes are made.
- Some functions reimplement functionality already mostly present in the Go standard library (`big.Int` has `SetString(s, 16)`, `IsUint64()`, `Uint64()`, `Bytes()`, etc...).

Recommendation: We feel this part of the code would benefit from being transformed either to:

1. A set of stateless type-to-type utility functions which use standard library functions where possible.

2. (If a stateful object is needed) use `big.Int` as the underlying primitive type, which is a superset of the other numerical types that `hex.String` interfaces with. Standard `big.Int` methods can be used where suitable.

Berachain: Fixed in commit 43902188.

Spearbit: Fix verified.

5.4.17 `node-api` listens on `0.0.0.0:3500` by default

Severity: Low Risk

Context: node-api/server/config.go#L24

Description: The `node-api` is an implementation of the Beacon API from Ethereum. This API's purpose is a means of communication between the beacon node and local clients (such as the validator client). It is not meant to be an external API and is not protected against malicious users. Some endpoints can be used to DOS and others may disclose information about the beacon node.

The `defaultAddress` for the `node-api` is set to `0.0.0.0:3500` at node-api/server/config.go#L24. This means it is listening on all interfaces on the machine and is exposed. The `node-api` is then, by default, accessible to anyone that can reach the port `3500` over the network.

Recommendation: Change the default to `127.0.0.1:3500`. If attempting to reach the `node-api` server from a different host, it is recommended to use SSH tunneling to securely allow remote access to the `localhost` server.

Berachain: Fixed in PR 2092.

Spearbit: Fix verified.

5.4.18 `math.NewU256FromBigInt` succeeds with negative integers

Severity: Low Risk

Context: mod/primitives/pkg/math/u256.go#L37-L40

Description: Like oversized integers (> 256 bits), negative integers ought to be rejected by `math.NewU256FromBigInt(b)` as neither can be expressed in the `U256` type:

```
package main

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

func main() {
    b, _ := new(big.Int).SetString("-123", 10)
    u := math.NewU256FromBigInt(b)
    /* Prints: 115792089237316195423570985008687907853269984665640564039457584007913129639813 */
    fmt.Println(u.String())
}
```

This was reported before (by others) to `holiman/uint256` and it looks like it won't be addressed: https://github.com/holiman/uint256/issues/115

Recommendation:

```
diff --git a/mod/primitives/pkg/math/u256.go b/mod/primitives/pkg/math/u256.go
index 4f10344c8..acf7b142e 100644
--- a/mod/primitives/pkg/math/u256.go
+++ b/mod/primitives/pkg/math/u256.go
@@ -36,6 +36,9 @@ func NewU256(v uint64) *U256 {

 // NewU256FromBigInt creates a new U256 from a big.Int.
 func NewU256FromBigInt(b *big.Int) *U256 {
+       if b.Sign() < 0 {
+           panic("Attempting to create U256 from negative big.Int")
+       }
        return uint256.MustFromBig(b)
 }
```

`NewU256FromBigInt` calls `uint256.MustFromBig` which will panic if the integer is oversized (> 256 bits). Hence, the suggested patch will also cause `NewU256FromBigInt` to panic if the integer is negative.

Consider whether this is a good approach and whether it might not be better to have `NewU256FromBigInt` return an error if the input is invalid. In that case `uint256.FromBig(b)` can be used for the conversion, which does not panic (and returns an `(*uint256.Int, bool)` tuple), unlike `uint256.MustFromBig(b)`.

In the fix, consider not including the integer in the error string because 1) decimal string conversion is a relatively expensive process and 2) the resulting string could be very large and this could stress the logging facilities.

Berachain: Fixed initial issue in PR 2079. The integer error string fix will be considered at a later date.

Spearbit: Initial fix verified.

5.4.19 `ExecutionPayload` JSON roundtrip failure

Severity: Low Risk

Context: mod/consensus-types/pkg/types/payload.go#L275-L444

Description: If the `BaseFeePerGas` field of `ExecutionPayload` is `nil`, `json.Marshal` on the `ExecutionPayload` object will succeed, but subsequent `json.Unmarshal` will fail with the error:

> missing required field 'baseFeePerGas' for ExecutionPayload

A second finding is that the `ExtraData` and `Transactions` fields may differ before and after a roundtrip. If they are `nil` before, they are empty after. This might not be worth addressing; I will consult with the team on this issue.

```
package main

import (
    "github.com/berachain/beacon-kit/mod/consensus-types/pkg/types"
    "github.com/berachain/beacon-kit/mod/primitives/pkg/encoding/json"
    "github.com/berachain/beacon-kit/mod/primitives/pkg/math"
    "reflect"
    "fmt"
)

func poc_empty_struct_JSON_roundtrip(set_basefeepergas bool) {
    var v1, v2 types.ExecutionPayload

    if set_basefeepergas {
        v1.BaseFeePerGas = &math.U256{}
    }

    /* -> json */
    serialized_json, err := json.Marshal(v1)
    if err != nil { return }
```

```
    /* -> struct */
    if err := json.Unmarshal(serialized_json, &v2); err != nil {
        /* path taken if set_basefeepergas == false */

        fmt.Println(string(serialized_json))
        /* err: "missing required field 'baseFeePerGas' for ExecutionPayload" */
        fmt.Println(err)
        panic("json.Unmarshal failed")
    }

    if !reflect.DeepEqual(v1.ExtraData, v2.ExtraData) {
        /* path taken if set_basefeepergas == true */

        fmt.Println("v1/v2 ExtraData DeepEqual mismatch")
        fmt.Println("v1.ExtraData is nil: ", v1.ExtraData == nil)
        fmt.Println("v2.ExtraData is nil: ", v2.ExtraData == nil)
        /* Prints:
           v1.ExtraData is nil:  true
           v2.ExtraData is nil:  false
        */

    }

    if !reflect.DeepEqual(v1.Transactions, v2.Transactions) {
        /* path taken if set_basefeepergas == true */

        fmt.Println("v1/v2 Transactions DeepEqual mismatch")
        fmt.Println("v1.Transactions is nil: ", v1.Transactions == nil)
        fmt.Println("v2.Transactions is nil: ", v2.Transactions == nil)
        /* Prints:
            v1.Transactions is nil:  true
            v2.Transactions is nil:  false
        */
    }
}

func main() {
    poc_empty_struct_JSON_roundtrip(false)
    poc_empty_struct_JSON_roundtrip(true)
}
```

Recommendation: Return error from `json.Marshal` if `BaseFeePerGas` is `nil`.

Berachain: Acknowleged. This fix will be considered at a later date.

Spearbit: Acknowledged.

5.4.20 `ExecutionPayloadHeader`, `ExecutionPayload` JSON deserialization panic

Severity: Low Risk

Context: mod/primitives/pkg/common/consensus.go#L119 and mod/primitives/pkg/common/execution.go#L126

Description: There is an implicit assumption in `Root.UnmarshalJSON` and `ExecutionAddress.UnmarshalJSON` that the input is at least 2 bytes, because if it is only 1 bytes, the expression `input[1 : len(input)-1]` resolves to `input[1 : 0]`, which is invalid because the end index is lower than the start index, leading to a panic.

```
func (r *Root) UnmarshalJSON(input []byte) error {
    return r.UnmarshalText(input[1 : len(input)-1])
}
```

```
func (a *ExecutionAddress) UnmarshalJSON(input []byte) error {
    return a.UnmarshalText(input[1 : len(input)-1])
}
```

The reproducer below demonstrates that this can occur when deserializing into `types.ExecutionPayloadHeader`.

```
package main

import (
    "github.com/berachain/beacon-kit/mod/consensus-types/pkg/types"
    "github.com/berachain/beacon-kit/mod/primitives/pkg/encoding/json"
)

func main() {
    var v1 types.ExecutionPayloadHeader
    s := "{\"transactionsroot\":1}"
    json.Unmarshal([]byte(s), &v1)
}
```

```
panic: runtime error: slice bounds out of range [1:0]

goroutine 1 [running]:
github.com/berachain/beacon-kit/mod/primitives/pkg/common.(*Root).UnmarshalJSON(0xcc47c0?,
↪  {0xc0003bd124?, 0xc000257480?, 0x417a00?})
    /home/jhg/berachain/poc4/beacon-kit/mod/primitives/pkg/common/consensus.go:114 +0x47
encoding/json.(*decodeState).literalStore(0xc00034c3f0, {0xc0003bd124, 0x1, 0x4}, {0xcc47c0?,
↪  0xc00034c3c8?, 0x1?}, 0x0)
    /home/jhg/berachain/poc4/go/src/encoding/json/decode.go:854 +0x21de
encoding/json.(*decodeState).value(0xc00034c3f0, {0xcc47c0?, 0xc00034c3c8?, 0x10?})
    /home/jhg/berachain/poc4/go/src/encoding/json/decode.go:388 +0x115
encoding/json.(*decodeState).object(0xc00034c3f0, {0xc1a3a0?, 0xc00034c360?, 0x46fedd?})
    /home/jhg/berachain/poc4/go/src/encoding/json/decode.go:755 +0xd11
encoding/json.(*decodeState).value(0xc00034c3f0, {0xc1a3a0?, 0xc00034c360?, 0x147a400?})
    /home/jhg/berachain/poc4/go/src/encoding/json/decode.go:374 +0x3e
encoding/json.(*decodeState).unmarshal(0xc00034c3f0, {0xc1a3a0?, 0xc00034c360?})
    /home/jhg/berachain/poc4/go/src/encoding/json/decode.go:181 +0x11e
encoding/json.Unmarshal({0xc0003bd110, 0x16, 0x18}, {0xc1a3a0, 0xc00034c360})
    /home/jhg/berachain/poc4/go/src/encoding/json/decode.go:108 +0xf9
github.com/berachain/beacon-kit/mod/consensus-types/pkg/types.(*ExecutionPayloadHeader).UnmarshalJSON(0
↪  xc000362c88, {0xc0003bd110, 0x16,
↪  0x18})
    /home/jhg/berachain/poc4/beacon-kit/mod/consensus-types/pkg/types/payload_header.go:337 +0x63
encoding/json.(*decodeState).object(0xc00034c2d0, {0xd3bb60?, 0xc000362c88?, 0x46fedd?})
    /home/jhg/berachain/poc4/go/src/encoding/json/decode.go:604 +0x6cf
encoding/json.(*decodeState).value(0xc00034c2d0, {0xd3bb60?, 0xc000362c88?, 0x147a400?})
    /home/jhg/berachain/poc4/go/src/encoding/json/decode.go:374 +0x3e
encoding/json.(*decodeState).unmarshal(0xc00034c2d0, {0xd3bb60?, 0xc000362c88?})
    /home/jhg/berachain/poc4/go/src/encoding/json/decode.go:181 +0x11e
encoding/json.Unmarshal({0xc0003bd110, 0x16, 0x18}, {0xd3bb60, 0xc000362c88})
    /home/jhg/berachain/poc4/go/src/encoding/json/decode.go:108 +0xf9
main.main()
    /home/jhg/berachain/poc4/beacon-kit/mod/consensus-types/pkg/p3/p.go:11 +0x4e
```

Reproducer that deserializes into `ExecutionPayload` and panics in `ExecutionAddress.UnmarshalJSON`.

```
package main

import (
    "github.com/berachain/beacon-kit/mod/consensus-types/pkg/types"
    "github.com/berachain/beacon-kit/mod/primitives/pkg/encoding/json"
)

func main() {
    var v1 types.ExecutionPayload
    s := "{\"feeRecipient\":1}"
    json.Unmarshal([]byte(s), &v1)
}
```

This issue is not related to the use of `github.com/goccy/go-json` as it also happens with `encoding/json`.

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

Spearbit: Acknowledged.

5.4.21 Insufficient input validation in `common.NewRootFromHex`

Severity: Low Risk

Context: mod/primitives/pkg/common/consensus.go#L71-L77

Description: There are two issues with `common.NewRootFromHex`:

- Undersized input results in panic (`panic: runtime error: cannot convert slice with length N to array or pointer to array with length 32`).
- Oversized input results in silent truncation without an error being raised:

```
package main

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

func main() {
    /* panic: runtime error: cannot convert slice with length 1 to array or pointer to array
↪   with length 32 */
    common.NewRootFromHex("0xaa")

    r, err := common.NewRootFromHex("0x000102030405060708090a0b0c0d0e0f101112131415161718191a1b1
↪   c1d1e1f2021222324252627")
    if err == nil {
        /* Prints: 0x000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
           Everything past the 32th byte is silently truncated
        */
        fmt.Println(r)
    }
}
```

`NewRootFromHex` is used for JSON unmarshalling, so both these issues can manifest if the application were to receive malformed JSON input.

Recommendation: Before instantiating and returning the `Root` object, return with error if `val` is not 32 bytes.

Berachain: Addressed by PR 2051.

Spearbit: Fix verified.

5.4.22 `version.DenebPlus` usage inconsistency

Severity: Low Risk

Context: mod/consensus-types/pkg/types/block.go#L88-L89

Description: In general, there are a few different instances of using the `version.DenebPlus` fork version. However, there is some inconsistency with treating that fork - places lacking version checks, panicing in one spot while erroring in another, etc.

A specific instance of this - For the `BeaconBlock` struct type, there are two direct constructor methods - `NewWithVersion` and `NewFromSSZ`. These operate in a similar manner, but they treat error handling of invalid fork versions inconsistently. Specifically, `NewFromSSZ` `panic`s when the `forkVersion == version.DenebPlus`, while `NewWithVersion` returns an error with `return &BeaconBlock{}, ErrForkVersionNotSupported`.

Recommendation: Treat the `version.DenebPlus` fork version with consistency.

For the instance in the `NewFromSSZ` method, return an error instead of `panic`ing on external SSZ input.

Berachain: Fixed in commit e608243b.

Spearbit: Fix verified.

5.4.23 `ExecutionPayload` implements `MarshalJSON` on pointer receiver

Severity: Low Risk

Context: mod/consensus-types/pkg/types/payload.go#L276

Description: The `ExecutionPayload` struct implements `MarshalJSON` on a pointer receiver instead of on a value receiver. Due to how the `json` library handles these definitions, using `json.Marshal` on a value receiver will result in attempting to use the default marshalling behavior instead of the custom defined `MarshalJSON` method. This could result in unexpected bytes or invalid round trip conversions if the value receiver is ever used to marshal via `json`.

Here is an example of this mismatch behavior:

```
func main() {
    val := types.ExecutionPayload{}

    /* Marshal on raw val uses default json marshal */
    raw_serialized, err := json.Marshal(val)
    if err != nil {
        panic("Raw serialization fails")
    }

    /* Marshal on ptr val uses implemented MarshalJSON */
    ptr_serialized, err := json.Marshal(&val)
    if err != nil {
        panic("Ptr serialization fails")
    }

    if !stdbytes.Equal(raw_serialized, ptr_serialized) {
        panic("mismatch") // panics here
    }
}
```

Note: Since this is used in a manner in which there is a generic type `ExecutionPayloadT` that corresponds to the pointer value `*ExecutionPayload`, this situation is unlikely unless you were to dereference the generic type.

Recommendation: Change the function definition of `MarshalJSON` for `ExecutionPayload` to use a value receiver instead of a pointer receiver as such:

```
- func (p *ExecutionPayload) MarshalJSON() ([]byte, error) {
+ func (p ExecutionPayload) MarshalJSON() ([]byte, error) {
```

Berachain: Fixed in PR 2042.

Spearbit: Fix verified.

5.4.24 Inappropriate input validation in `UnmarshalText` for `ExecutionAddress`

Severity: Low Risk

Context: mod/primitives/pkg/common/execution.go#L116

Description: Ill-formatted and under- and oversized input to `UnmarshalText` for `ExecutionAddress` is incorrectly handled, leading to panics and silent data truncation. Reproducer:

```
package main

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

func main() {
    var v common.ExecutionAddress

    /* panic: runtime error: cannot convert slice with length 1 to array or pointer to array with
↪   length 20 */
    json.Unmarshal([]byte("\"0xab\""), &v)

    /* panic: hex string without 0x prefix */
    json.Unmarshal([]byte("\"abc\""), &v)

    err := json.Unmarshal([]byte("\"0x000102030405060708090a0b0c0d0e0f101112131415161718\""), &v)
    if err == nil {
        /* Silent truncation: prints: 0x0001020304050607080090a0b0c0d0e0f10111213 */
        fmt.Println(v)
    }
}
```

`UnmarshalText` can return `error` on invalid input which is preferred to panicking. Returning error on oversized input is preferred to truncation.

Recommendation: Apply this patch:

```
diff --git a/mod/primitives/pkg/common/execution.go b/mod/primitives/pkg/common/execution.go
index f31c782e9..d3c5d915c 100644
--- a/mod/primitives/pkg/common/execution.go
+++ b/mod/primitives/pkg/common/execution.go
@@ -113,8 +113,7 @@ func (a ExecutionAddress) MarshalText() ([]byte, error) {

 // UnmarshalText parses an address in hex syntax.
 func (a *ExecutionAddress) UnmarshalText(input []byte) error {
-       *a = NewExecutionAddressFromHex(string(input))
-       return nil
+       return hex.DecodeFixedText(input, a[:])
 }

 // MarshalJSON returns the JSON representation of a.
```

Berachain: Fixed by PR 2034 (added reproducers as UTs to show fix).

Spearbit: Fixed.

5.4.25 Non-determinism in `github.com/goccy/go-json` JSON parser

Severity: Low Risk

Context: mod/primitives/pkg/encoding/json/json.go

Description: Via `mod/primitives/pkg/encoding/json`, the `github.com/goccy/go-json` library is used for JSON IO. This library demonstrates non-determinism, which can be reproducer using the code below:

```
package main

import (
    "fmt"
    "math/rand"
    "github.com/berachain/beacon-kit/mod/engine-primitives/pkg/engine-primitives"
    "github.com/berachain/beacon-kit/mod/primitives/pkg/encoding/json"
)

func main() {
    original := []byte{0x7b, 0x22, 0x5c}
    rand.Seed(123)

    iter := 0
    for {
        iter += 1
        fmt.Println("Iteration ", iter)
        mutated := []byte{0x7b, 0x22, 0x5c}
        numToChange := rand.Intn(3) + 1

        for i := 0; i < numToChange; i++ {
            index := rand.Intn(3)

            newByte := byte(rand.Intn(95) + 32)

            mutated[index] = newByte
        }

        json.Unmarshal(mutated, &engineprimitives.PayloadStatusV1{})
        json.Unmarshal(original, &engineprimitives.PayloadStatusV1{})
    }
}
```

Usually this will panic after a few thousand iterations (tested using `go1.23.0 linux/amd64`). The panic itself is not the most worrying symptom; the non-determinism raises concerns about whether data integrity can be guaranteed in general.

The underlying cause may be the use of `unsafe` blocks in the `goccy/go-json` library. That would imply that random process memory is inadvertently integrated in the decoder state, which poses risks to confidentiality, integrity and availability.

The non-determinism can be observed as far back as June 2021. For a bug to go unnoticed for such a long time may imply that the library is under-tested or under-maintained.

Recommendation: Remove `github.com/goccy/go-json` as a dependency and use `encoding/json` instead.

Berachain: Being addressed in PR 2033.

Spearbit: Fix confirmed.

5.4.26 `math.U256 UnmarshalJSON` does not reject oversized decimal input

Severity: Low Risk

Context: primitives/pkg/math/u256.go#L55

Description: The `math.U256` json unmarshaller misinterprets rather than rejects invalid input. Specifically, an integer exceeding 256 bits presented as a decimal string is accepted. Oversized integers in hexadecimal representation are correctly rejected. Reproducer:

```
func main() {
    var v math.U256
    // This decimal value is 2^256+1
    if err := json.Unmarshal([]byte("115792089237316195423570985008687907853269984665640564039457584007
↪  913129639937"), &v); err != nil
↪  {
        return
    }
    fmt.Println("As string:", v.String()) // Prints 1
}
```

This is not a bug in beacon-kit, but in the underlying `https://github.com/holiman/uint256` dependency. It has been reported upstream.

Recommendation: Subscribe to holiman/uint256 issue 184 and await fix and new release. Then update to new release. Add reproducer above to tests.

Berachain: Fixed in PR 2416.

Spearbit: Fix verified.

5.4.27 Race condition in `broker.Broker` results in panic

Severity: Low Risk

Context: mod/async/pkg/broker/broker.go#L127-L133

Description: The `broker.Broker` struct includes the following map `subscriptions map[chan T]struct{}`. This maps acts as a list of all of the subscribed channels. You can add and remove channels from this map by using the `Subscribe` and `Unsubscribe` functions. However, reads and writes to this `subscriptions` map are not protected by any concurrency locking mechanism. During any given broadcast, a broker could `Unsubscribe`, resulting in a channel closing at the same time that it is being read from or written to. This results in a race condition in which a closed channel is written to, inducing a panic.

Here is a proof of concept that results in `panic: send on closed channel`:

```
package main

import (
    "context"
    "github.com/berachain/beacon-kit/mod/async/pkg/broker"
    "github.com/berachain/beacon-kit/mod/async/pkg/dispatcher"
    "github.com/berachain/beacon-kit/mod/primitives/pkg/async"
)

type event struct {
    ctx  context.Context
    id   async.EventID
    data uint64
}

func (e event) ID() async.EventID {
    return e.id
}
```

```go
func (e event) Context() context.Context {
    return e.ctx
}

func printer(ctx context.Context, ch chan event) {
    for {
        select {
        case <-ctx.Done():
            return
        case <-ch:
        }
    }
}

func main() {
    eventIDStr := "testevent"
    eventID := async.EventID(eventIDStr)
    ctx, cancel := context.WithCancel(context.Background())

    // Create the broker
    testeventBroker := broker.New[event](eventIDStr)

    // Create the dispatcher with nil logger and register broker
    dispatch, err := dispatcher.New(nil)
    if err != nil {
        panic("could not create dispatcher")
    }

    if err := dispatch.RegisterBrokers(testeventBroker); err != nil {
        panic("could not register broker")
    }

    if err := dispatch.Start(ctx); err != nil {
        panic("could not start dispatcher")
    }

    // Loop until we hit the race condition panic
    for i := 1; i < 1000; i++ {
        // Create channel and start reading from it
        ch := make(chan event)
        go printer(ctx, ch)

        // Subscribe the new channel
        if err := dispatch.Subscribe(eventID, ch); err != nil {
            panic("could not subscribe to event")
        }

        // Publish a new event
        if err := dispatch.Publish(event{
            ctx:  ctx,
            id:   eventID,
            data: uint64(i),
        }); err != nil {
            panic("could not publish event")
        }

        // Unsubscribe from the channel
        if err := dispatch.Unsubscribe(eventID, ch); err != nil {
            panic("could not unsubscribe from event")
        }
    }
```

```
    cancel()
}
```

Recommendation: Add a `sync.RWMutex` to the `type Broker[T async.BaseEvent] struct` that protects accesses to `subscriptions`.

Berachain: Fixed in PR 2225.

Spearbit: Fix verified.

5.5 Informational

5.5.1 Go `encoding/json.Unmarshal` data corruption of `string` fields

Severity: Informational

Context: golang/go/tree/master/src/encoding/json

Description: *Note: this is not a bug in beacon-kit. Arguably it is a design issue with the Go standard library found during this audit.*

When unmarshalling data using `encoding/json.Unmarshal`, any `string` fields in the target variable whose value is set to `null` in the serialized form, will silently retain their previous value (if any) without producing an error.

```
package main

import (
    "encoding/json"
    "fmt"
)

func main() {
    var s string

    err := json.Unmarshal([]byte(`"abc"`), &s)
    if err != nil {
        return
    }
    fmt.Printf("After decoding 'abc': %q\n", s)

    err = json.Unmarshal([]byte("null"), &s)
    if err != nil {
        return
    }
    fmt.Printf("After decoding 'null': %q\n", s)
}
```

This is an issue that could manifest in any code where a variable is reused for unmarshalling, e.g.:

```
func processInputs(inputs [][]byte) {
    type MyStruct struct {
        x int
        y string
    }
    var v MyStruct
    for _, input := range inputs {
        err := json.Unmarshal(input, &v)
        if err != nil {
            continue
        }
        /* Do something with values in tmp */
        if v.x == 123 {
            /* ... */
        }
    }
}
```

This would work fine until a string field is encoded as `null` in which case the value of the previous unmarshalling is retained, which amounts to data corruption, as `v` will now contain values that were not encoded in serialized form.

Recommendation: As far as I am aware the only way to deal with this would be to create a custom `String` time with a custom `UnmarshalJSON` that rejects `null` values.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.5.2 Inaccurate error message in `state_processor_staking.go`

Severity: Informational

Context: mod/state-transition/pkg/core/state_processor_staking.go#L250

Description: Per the mod/state-transition/pkg/core/errors.go, the error message `ErrNumWithdrawalsMismatch` is used to trigger an error when the number of withdrawals don't match a specific value.

However, the same error message is re-used while comparing a withdrawal to the local state. If the local state does not match the incoming withdrawal, then the `ErrNumWithdrawalsMismatch` error is triggered, leading to confusion while debugging.

Recommendation: Consider introducing an more accurate error message in the above mentioned condition where the withdrawal does not match the local state.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.5.3 Notes on `RangeDB DeleteRange`

Severity: Informational

Context: mod/storage/pkg/filedb/range_db.go#L91-L103

Description/Recommendation: Regarding `func (db *RangeDB) DeleteRange(from, to uint64) error`:

1. Consider returning error if `from > to`. That condition would indicate something is amiss, and while it wouldn't cause any deletions (because the subsequent `for` loop body would not be exercised), it could be better to signal this anomaly to the caller.

   On the other hand if users of `DeleteRange` implicitly expect leniency from the function (silently ignore invalid ranges, e.g. the way it is currently) then returning error could only cause problems higher up the call chain, like inappropriately invoking error handlers for what fundamentally is not an error.

A way to solve this could be to define a `Range` struct that can only be instantiated with valid arguments, e.g.:

```
type Range struct {
    Start uint64
End   uint64
}

var ErrInvalidRange = fmt.Errorf("invalid range: start cannot be greater than end")

func NewRange(start, end uint64) (Range, error) {
    if start > end {
        return Range{}, ErrInvalidRange
    }
    return Range{Start: start, End: end}, nil
}
```

This would create a more strongly defined contract between `DeleteRange` and its callers by disambiguating the meaning of `from` and `to`.

`DeleteRange` is currently used very sparingly across the code so introducing a new type for this could be considered over-engineering, but valid range construction is notoriously hard to get right (buffer overflows in C are often due to off-by-one and similar bugs) so this could be a good place to catch such a miscalculation early.

2. The `for` loop body will return with error upon the first failure to remove the path.

```
for ; from < to; from++ {
    path := strconv.FormatUint(from, 10) + "/"
    if err := f.fs.RemoveAll(path); err != nil {
        return err
    }
}
```

It should be considered whether a single deletion error must prevent deleting the other paths as well.

A single failure arguably denotes a critical integrity fault if the caller expects the path to exist, or if it exists but cannot be deleted due to file permissions, filesystem corruption, inability to connect to a network drive etc. Halting the program might be preferable to proceeding when the state is evidently corrupt or incapable of performing essential operations.

On the other hand if there's a good reason to expect `f.fs.RemoveAll` to fail for some entries, then not proceeding to delete the remaining entries would cause an inconsistency between program state and database state, and the code should be updated accordingly.

Berachain: Addressed in PR 2106. The strategy followed was avoiding the `Range` struct (although it's very nice) because there are instances where we would modify the range after initialization, which requires new checks.

Spearbit: Fixed.

5.5.4 `RangeDB` does not guarantee producing oversized filenames

Severity: Informational

Context: mod/storage/pkg/filedb/range_db.go#L122

Description: Currently only `KZGCommitment` (`[48]byte`) is used as a key for the database interface. On Linux/ext4 filenames are limited to 255 characters and paths to 4096 characters.

With `KZGCommitment` as a key, the theoretical maximum of a filename is:

```
# Python
len(str(2**64) + '/' + '0x' + ('00' * 48)) = 119
```

If keys with a dynamic size were to be introduced this could lead to file creation errors for (some) data leading to database integrity issues.

Recommendation: It might be better to enforce the use of fixed-size byte slices as keys by requiring the key type to be specified upon declaration via type arguments, e.g.:

```
type DB[N int] interface {
    Get(key [N]byte) ([]byte, error)
    Has(key [N]byte) (bool, error)
    Set(key [N]byte, value []byte) error
    Delete(key [N]byte) error
}
```

That will make it impossible for dynamically sized keys to exist. Keys can still be too large to express as an ext4 filename, but this can be caught with tests.

With regards to the path concatenation, consider using a canonical solution like `path/filepath.Join()` which is designed to work as intended across all operating systems, instead of manual string concatenation (which makes an assumption about the path separator, `/`).

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

Spearbit: Acknowledged.

5.5.5 `ProcessSlots()` returns silently on invalid `slot`

Severity: Informational

Context: state-transition/pkg/core/state_processor.go#L195-L198

Description: The `ProcessSlots()` function does not check to see if the requested slot is greater than the current state's slot. It will return silently without error. In the `state-transition` in `Transition()`, this is not too much of a problem since the `processBlockHeader()` ends up validating the slot, but this could cause issue in other places.

Recommendation: Return error in `ProcessSlots()` when `stateSlot >= slot`:

```
stateSlot, err := st.GetSlot()
if err != nil {
    return nil, err
}
if stateSlot >= slot {
    return nil, ErrBlockSlotTooLow
}
```

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.5.6 Unused file `helpers.go`

Severity: Informational

Context: consensus/pkg/cometbft/service/helpers.go

Description/Recommendation: This file is unused. Consider deleting it.

Berachain: Fixed in a commit.

Spearbit: Fix verified.

5.5.7 Remove hardcoded `bArtio chainId` from `addValidatorToRegistry` function

Severity: Informational

Context: mod/state-transition/pkg/core/state_processor_staking.go#L195

Description: The `addValidatorToRegistry` function adds new validators to the registry. This function has a hardcoded `bArtio chainId` and utilizes that to route the validator addition to respective functions.

Recommendation: Consider moving `bArtio chainId` to an environment/config file.

Berachain: Fixed in PR 2362.

Spearbit: Fix verified.

5.5.8 Possible `nil` check for payload in `validateStatelessPayload` and `validateStatefulPayload` functions

Severity: Informational

Context: mod/state-transition/pkg/core/state_processor_payload.go#L138, mod/state-transition/pkg/core/state_processor_payload.go#L99

Description: The `validateStatefulPayload` and `validateStatelessPayload` functions perform checks on the execution payload. The function lacks an explicit `nil` check for the payload returned from `body.GetExecutionPayload()`, which could guard the validations against unexpected behavior.

Recommendation: Consider adding a `nil` check to the payload returned by both of the above-mentioned functions.

```
  func (sp *StateProcessor[
      BeaconBlockT, _, _, _,
      _, _, _, _, _, _, _, _, _, _, _, _, _,
  ]) validateStatelessPayload(blk BeaconBlockT) error {
      body := blk.GetBody()
      payload := body.GetExecutionPayload()

+     if payload == nil return error;
```

Berachain: Acknowledged. The SSZ unmarshalling should guarantee that this data is non-nil.

Spearbit: Acknowledged.

5.5.9 Remove hardcoded `bArtio`-specific config in `state_processor_genesis.go`

Severity: Informational

Context: mod/state-transition/pkg/core/state_processor_genesis.go#L33

Description: The `state_processor_genesis.go` contains hardcoded values specific to the "Bartio" chain, including a validator root and chain ID. These values are used in a conditional statement within the `InitializePreminedBeaconStateFromEth1` function.

```
//nolint:lll // temporary.
const (
  bArtioValRoot = "0x9147586693b6e8faa837715c0f3071c2000045b54233901c2e7871b15872bc43"
  bArtioChainID = 80084
)
```

Recommendation: Consider introducing these chain-specific values to external configuration files that can be easily updated without changing the source code.

Berachain: Fixed in PR 2362.

Spearbit: Fix verified.

5.5.10 `recover()` pattern can isolate `panic`'s and increase node robustness

Severity: Informational

Context: Global scope

Description: There are a number of functions in beacon-kit which panic purposefully with `panic()` or have been identified to be prone to built-in Go panics (eg. slice indexing or nil pointer dereferences). These panics threaten node operation because they will kill the entire beacon-kit process when they happen.

Recommendation: Use the `recover()` function where applicable (eg. at the point of registration of rpc handlers) to prevent panics from threatening node liveness.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.5.11 `EventDispatcher` usage does not guarantee matching response

Severity: Informational

Context: `async.BeaconBlockReceived`

Description: Several `Service` services utilize the `EventDispatcher`'s publisher/subscriber model. These services will publish a message to a channel with a single subscriber, and then await a response over a separate channel that it is subscribed to.

For example, the `ProcessProposal()` publishes the `async.BeaconBlockReceived` event. It will then await the `async.BeaconBlockVerified` response through separate event subscriptions. A separate event loop goroutine that is subscribed to `async.BeaconBlockReceived` will validate the data and then send back the `async.BeaconBlockVerified` event response. In this manner, the `EventDispatcher` is used to asynchronously process events and then collect the results at the end.

There is a potential issue with this setup. Let's say that after `ProcessProposal()` publishes the `async.BeaconBlockReceived` event, some error occurs in `ProcessProposal()` and we stop awaiting the `async.BeaconBlockVerified` response. This does not prevent the event loop that is handling the event from finishing processing and sending back a response. Thus, there currently is a channel "flush" that happens at the beginning of `ProcessProposal()` to ensure that we are not receiving old "Verified" responses if we had previously errored out.

What if, however, the event loop that is handling the `BeaconBlockReceived` happens to take a while and send back its `BeaconBlockVerified` response AFTER this flush happens? Then the `ProcessProposal` receives a `BeaconBlockVerified` message for the wrong BeaconBlock.

This error case exists in other services when the `EventDispatcher` is used in this manner, though the timing requirements are extremely limiting.

Recommendation: When publishing an event via the `EventDispatcher` and then awaiting a response, verify that the response correlates with the initial event. This can be achieved by sending a unique ID with the published event and checking that it matches the ID of the response.

Berachain: Fixed in PR 2225.

Spearbit: Fix verified.

5.5.12 Missing validator for `validator_status`

Severity: Informational

Context: node-api/engines/echo/vaildator.go#L68

Description: The `ValidateValidatorStatus()` function is unused, leaving `validator_status` fields from API requests unvalidated.

Recommendation: Add `ValidateValidatorStatus` for `validator_status` fields in `ConstructValidator()` at node-api/engines/echo/vaildator.go#L61 (with sorted fields):

```
validators := map[string](func(fl validator.FieldLevel) bool){
    "block_id":         ValidateBlockID,
    "epoch":            ValidateUint64,
    "execution_id":     ValidateExecutionID,
    "slot":             ValidateUint64,
    "state_id":         ValidateStateID,
    "validator_id":     ValidateValidatorID,
    "validator_status": ValidateValidatorStatus,
}
```

Berachain: Fixed in PR 2090.

Spearbit: Fix verified.

5.5.13 `node-api` input validation inconsistent with usage

Severity: Informational

Context: node-api/engines/echo/vaildator.go#L80-L207

Description: The `node-api` contains a bunch of validation functions in node-api/engines/echo/vaildator.go#L80-L207 that get called during the `BindAndValidate()` routine when unpacking the JSON request input. These validated requests will then go on to their individual handlers to be used.

In some instances, the usage of the request value in the handler is different than what is validate in the validator function. Here is an example - `ValidateValidatorID()` validates the `validator_id` string field with the following:

```
validateRegex(fl.Field().String(), `^0x[0-9a-fA-F]{1,96}$`)
```

The `validator_id` is then used by `crypto.BLSPubkey.UnmarshalText()`. The regex would, for example, allow `0x001`. The unmarshal would reject that same input due to the string being un-even in length.

Recommendation: For each validation function, use strict type enforcement that corresponds with the usage of the request data.

- `ValidateValidatorID()`:

```
func ValidateValidatorID(fl validator.FieldLevel) bool {
var key crypto.BLSPubkey
err := key.UnmarshalText([]byte(fl.Field().String()))
if err == nil {
    return true
}
if ValidateUint64(fl) {
    return true
}
return false
}
```

- `ValidateRoot()`:

```
func ValidateRoot(value string) bool {
_, err := common.NewRootFromHex(value)
return err == nil
}
```

- `validateRegex()`: Deleted.

Berachain: Fixed in PR 2093.

Spearbit: Fix verified.

5.5.14 Inconsistent logging and error checks on `ExecutionPayload` retrieval

Severity: Informational

Context: payload/pkg/builder/payload.go#L143

Description: There are two ways to retrieve an `ExecutionPayload` from the engine client:

1. `RequestPayloadSync()` sends the request, awaits a response, and returns the `BuiltExecutionPayloadEnv` synchronously.
2. `RetrievePayload` grabs the `BuiltExecutionPayloadEnv` that was previously asynchronously requested.

These should perform the same checks and logging. However, `RequestPayloadSync()` directly returns the values returned from `pb.ee.GetPayload()` at payload/pkg/builder/payload.go#L143 while `RetrievePayload()` will check if the returned envelope is `nil`, log some information about retrieving the payload from the local builder, and also log errors when there is an unexpected `FeeRecipient`.

Recommendation: `RequestPayloadSync()` should call `pb.RetrievePayload()` instead of `pb.ee.GetPayload()` to ensure that the same logging and checks happen.

If re-grabbing the `payloadID` in `RetrievePayload()` introduces too much overhead, then `RequestPayloadSync()` should implement the same logging and error handling as `RetrievePayload`.

Berachain: Fixed in PR 2088.

Spearbit: Fix verified.

5.5.15 `SendForceHeadFCU()` does not check if `PayloadBuilder` is disabled

Severity: Informational

Context: payload/pkg/builder/payload.go#L230

Description: The `PayloadBuilder.SendForceHeadFCU()` method does not check if the `PayloadBuilder` is disabled via `!pb.Enabled()` like the other `PayloadBuilder` methods.

Recommendation: Add the following check to the beginning of `SendForceHeadFCU()`:

```
if !pb.Enabled() {
    return nil, ErrPayloadBuilderDisabled
}
```

Berachain: Fixed in PR 2087.

Spearbit: Fix verified.

5.5.16 Two SSZ libraries used for same types

Severity: Informational

Context: engine-primitives/pkg/engine-primitives/withdrawal.go#L111

Description: There are few different SSZ marshalable types that implement the SSZ marshaler using both `github.com/ferranbt/fastssz` and `github.com/karalabe/ssz`. There are discrepancies between the internals of these libraries and functions between the libraries should not be used interoperably. For example, `ssz.HashTreeRoot()` does not return an error when dynamic slice fields exceed their allotted `maxSize`, because it is enforced by the `ssz` marshaler. However, `fastssz.HashTreeRootWith` does return error in that same situation.

Recommendation: In all locations where both the `ferranbt/fastssz` and `karalabe/ssz` libraries are implemented for a single type, ensure that each implementation is used independently of each other. In the current state of the code, this is not an issue. Moving forward, I would recommend refactoring to use a single one of these libraries.

Berachain: Acknowledged in meeting.

Spearbit: Acknowledged.

5.5.17 Improve metrics coverage in engine implementation

Severity: Informational

Context: execution/pkg/engine/engine.go#L294, execution/pkg/engine/engine.go#L99, execution/pkg/engine/engine.go#L203

Description: The current implementation of the Engine in `mod/execution/pkg/engine/engine.go` has good metrics coverage for the main functions `NotifyForkchoiceUpdate` and `VerifyAndNotifyNewPayload`. However, more metrics could be added to increase observability and monitoring.

1. Implement metrics for `Start`:
    - Record successful engine starts.
    - Track any panics during startup.
2. Add a specific metric for invalid block hash and versioned hashes:
    - Count occurrences of this specific error case.
3. Implement a metric for optimistic handling in `VerifyAndNotifyNewPayload`:
    - Track the frequency of optimistic case triggers.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.5.18 Potential integer truncation/UB in `math.GweiFromWei`

Severity: Informational

Context: mod/primitives/pkg/math/u64.go#L143

Description: `GweiFromWei` takes `i *big.Int)` and returns `i / (10**9)` as a `uint64`. From this it follows that for `i < 0` and `i >= (2**64) * (10**9)`, the result cannot be expressed as a `uint64`. Truncation will occur on the last line of the function in casting `i` to `uint64` through `i.Uint64()`:

```
func GweiFromWei(i *big.Int) Gwei {
    intToGwei := big.NewInt(0).SetUint64(GweiPerWei)
    i.Div(i, intToGwei)
    return Gwei(i.Uint64())
}
```

The Wei amount required for this to happen is unrealistically large (`18446744073709551616000000000`) or small (negative), but if it were to emerge through a serialization bug or other mishap, any logic dealing with Wei amounts will error out where this value it too large to satisfy some constraint.

However, the truncated Gwei value can be reasonably small (e.g. `0` or `1`) and more likely than not will pass any checks in code that processes Gwei-denominated amounts.

Overflow is not an issue in `ToWei` (which takes `uint64` and returns `U256`) because `(2**64-1) * (10**9) < (2**256)`.

UB (undefined behavior) is mentioned in the issue title as `https://pkg.go.dev/math/big#Int.Uint64` states:

> `Uint64` returns the `uint64` representation of x. If x cannot be represented in a `uint64`, the result is undefined.

Reproducer:

```
package main

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

func main() {
    {
        b, _ := new(big.Int).SetString("18446744073709551616000000000", 10)
        gwei := math.GweiFromWei(b)
        /* Prints 0 */
        fmt.Println(gwei)
    }
    {
        b, _ := new(big.Int).SetString("-1", 10)
        gwei := math.GweiFromWei(b)
        /* Prints 1 */
        fmt.Println(gwei)
    }
}
```

Recommendation:

```
func GweiFromWei(i *big.Int) (Gwei, error) {
        intToGwei := big.NewInt(0).SetUint64(GweiPerWei)
        i.Div(i, intToGwei)
        if !i.IsUint64() {
            return 0, errors.New("uint64 overflow")
        }
        return Gwei(i.Uint64()), nil
}
```

Berachain: Addressed by PR 2080.

Spearbit: Fixed.

5.5.19 Potential undefined behavior in call to `math/big.Int.Uint64()` in `EngineClient.verifyChainIDAndConnection()`

Severity: Informational

Context: mod/execution/pkg/client/client.go#L170

Description: In `EngineClient.verifyChainIDAndConnection()`, the `Uint64()` is called on `eth1ChainID`, which is a `math/big.Int`:

```
if chainID.Unwrap() != s.eth1ChainID.Uint64() {
    err = errors.Wrapf(
        ErrMismatchedEth1ChainID,
        "wanted chain ID %d, got %d",
        s.eth1ChainID,
        chainID,
    )
    return err
}
```

big#Int.Uint64 states:

> `Uint64` returns the `uint64` representation of x. If x cannot be represented in a uint64, the result is undefined.

There might not be a way to set `eth1ChainID` to an illegal `uint64` value, but out of an abundance of caution, consider verifying that it is properly castable (see patch below), as non-determinism or other side effects could arise out of the uncertain nature of the return value.

Recommendation:

```
diff --git a/mod/execution/pkg/client/client.go b/mod/execution/pkg/client/client.go
index e14d8cc55..1f8defa92 100644
--- a/mod/execution/pkg/client/client.go
+++ b/mod/execution/pkg/client/client.go
@@ -167,7 +167,7 @@ func (s *EngineClient[
                return err
        }

-       if chainID.Unwrap() != s.eth1ChainID.Uint64() {
+       if !s.eth1ChainID.IsUint64() || chainID.Unwrap() != s.eth1ChainID.Uint64() {
                err = errors.Wrapf(
                        ErrMismatchedEth1ChainID,
                        "wanted chain ID %d, got %d",
```

Berachain: Addressed by PR 2086.

Spearbit: Fixed.

5.5.20 `math.U256` JSON roundtrip failure

Severity: Informational

Context: mod/primitives/pkg/math/u256.go#L48-L50

Description:

```
package main

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

func main() {
    v1 := math.NewU256(123)
    fmt.Println("As string: ", v1.String())

    /* U256 -> bytes */
    serialized, err := json.Marshal(v1)
    if err != nil {
        panic("Serialization 1 fails")
    }
    fmt.Println("serialized 1: ", string(serialized))

    /* U256 -> bytes -> U256 */
    var v2 math.U256
    if err := json.Unmarshal(serialized, &v2); err != nil {
        panic("Deserialization 1 fails")
    }
    fmt.Println("As string after roundtrip: ", v2.String())

    /* U256 -> bytes -> U256 -> bytes */
    serialized2, err := json.Marshal(v2)
    if err != nil {
        panic("Serialization 2 fails")
    }
    fmt.Println("serialized 2: ", string(serialized2))

    /* U256 -> bytes -> U256 -> bytes -> U256 */
    var v3 math.U256
    if err := json.Unmarshal(serialized2, &v3); err != nil {
        panic("Deserialization 2 fails")
    }
}
```

```
As string:  123
serialized 1:  "123"
As string after roundtrip:  123
serialized 2:  [123,0,0,0]
panic: Deserialization 2 fails
```

See also discussion at https://github.com/spearbit-audits/review-berachain-beaconkit-0919/pull/1/files#r1769192908

Recommendation: Implement the `UnmarshalJSON` method on a value receiver.

Berachain: Acknowledged. Will consider this in the future.

Spearbit: Acknowledged.

5.5.21 `ticker.Stop()` used without `defer`

Severity: Informational

Context: execution/pkg/client/ethclient/rpc/client.go#L85, node-core/pkg/services/version/version.go#L77

Description: The `time.NewTicker` ticker is only stopped with `Stop()` if we are safely returning from the function. This means that if a `panic` or some other exit condition happens, the ticker will continue to run.

This only becomes a problem if we attempt to recover from a `panic` or other exit conditions. Then we will have a ticker that keeps running and is a memory/resource leak.

Recommendation: There are two spots this happens and the recommendation is slightly different for each:

- execution/pkg/client/ethclient/rpc/client.go#L85: Call `defer ticker.Stop()` immediately after creation.
- node-core/pkg/services/version/version.go#L77: Call `defer ticker.Stop()` at the top of the new inline goroutine.

Berachain: Fixed in commit 9715e5d0.

Spearbit: Fix verified.

5.5.22 Redundant check for `version.DenebPlus`

Severity: Informational

Context: execution/pkg/client/engine.go#L161

Description: There is a redundant check `forkVersion >= version.DenebPlus` that will never evaluate to `true` since `version.Deneb` is already checked and is less than `version.DenebPlus`.

Recommendation: Remove the redundant check `forkVersion >= version.DenebPlus`.

Berachain: Fixed in commit 18f5d37e

Spearbit: Fix verified.

5.5.23 Duplicate `jwt` signing functions

Severity: Informational

Context: primitives/pkg/net/jwt/sign.go#L31, primitives/pkg/net/jwt/jwt.go#L79

Description: The `sign.go` file contains a single function `BuildSignedJWT` which duplicates the `BuildSignedToken` function in the same package in `jwt.go`.

Recommendation: Remove the primitives/pkg/net/jwt/sign.go file.

Berachain: Fixed in commit afba230d.

Spearbit: Fix verified.

5.5.24 Inconsistent `math.Gwei` usage in `BeaconState`

Severity: Informational

Context: consensus-types/pkg/types/state.go#L72

Description: The `Slashings` and `Balances` fields in the `BeaconState` can be made to use `[]math.Gwei` instead of `[]uint64`. This would be made consistent with the `TotalSlashing` field.

Recommendation: Change the `Slashings` and `Balances` field types in-place from `[]uint64` to `[]math.Gwei`. You'll have to change the types at the locations they get used as well. Note that this is an in-place change and produces the same SSZ `HashTreeRoot`.

Berachain: Fixed in commit 0b80e548.

Spearbit: Fix verified.

5.5.25 Use `EthSecp256k1CredentialPrefix` constant for clarity

Severity: Informational

Context: consensus-types/pkg/types/withdrawal_credentials.go#L45

Description: The `NewCredentialsFromExecutionAddress` function should use the constant `EthSecp256k1CredentialPrefix` instead of `0x01`.

Recommendation:

```
- credentials[0] = 0x01
+ credentials[0] = EthSecp256k1CredentialPrefix
```

Berachain: Fixed in commit 98e38c91.

Spearbit: Fix verified.

5.5.26 Unnecessary extra allocation in `BeaconBlock.NewFromSSZ`

Severity: Informational

Context: mod/consensus-types/pkg/types/block.go#L87

Description: The `NewFromSSZ` function double allocates a `BeaconBlock` in the expected success case (`forkVersion == version.Deneb`). This is super minor and definitely a nitpick, but this double allocation could be removed and be made consistent with `BeaconBlock.NewWithVersion`.

Recommendation: Make the allocation scheme the same as `NewWithVersion`, where `block` is initially just a nil pointer. This will avoid the double allocation on each call.

Berachain: Fixed in commit 54c7c250.

Spearbit: Fix verified.

5.5.27 `ProposerIndex` has wrong type alias

Severity: Informational

Context: mod/consensus-types/pkg/types/block.go#L37

Description: The `ProposerIndex` field of the `BeaconBlock` struct uses the type `math.Slot`. However, the `BeaconBlock.NewWithVersion()` function expected `proposerIndex math.ValidatorIndex`. These are both type aliases for the same type, but they should be corrected regardless.

Recommendation: Change the `ProposerIndex` type from `math.Slot` to `math.ValidatorIndex`.

```
- ProposerIndex math.Slot `json:"proposer_index"`
+ ProposerIndex math.ValidatorIndex `json:"proposer_index"`
```

Berachain: Fixed in commit 4b7b4d98.

Spearbit: Fix verified.

5.5.28 `PrevPowerOfTwo` and `NextPowerOfTwo` functions could be optimized

Severity: Informational

Context: mod/primitives/pkg/math/pow/pow.go#L27, mod/primitives/pkg/math/pow/pow.go#L43

Description: The functions `PrevPowerOfTwo` and `NextPowerOfTwo` are used to calculate the previous and next power of two for a given `u64` input.

The functions return one if the value equals zero. However, to further optimize them, they can return 1 for values less than or equal to 1 in both functions. Currently, the two functions perform unnecessary bit operations for inputs equal to 1, slightly impacting performance.

Recommendation: Consider optimizing the two functions mentioned above as follows:

```
func PrevPowerOfTwo[U64T ~uint64](u U64T) U64T {
- if u == 0 {
+ if u <= 1 {
      return 1
  }
  // ...
}

func NextPowerOfTwo[U64T ~uint64](u U64T) U64T {
-  if u == 0 {
+  if u <= 1 {
     return 1
  }
  // ...
}
```

Berachain: Fixed in commit 2806185c.

Spearbit: Fix verified.

5.5.29 Typographical Issues

Severity: Informational

Context: mod/payload/pkg/attributes/factory.go#L61, mod/payload/pkg/cache/payload_id.go#L120, mod/consensus-types/pkg/types/block.go#L196, mod/primitives/pkg/constraints/basic.go#L39, mod/primitives/pkg/constraints/basic.go#L50, mod/primitives/pkg/encoding/hex/bytes.go#L85, mod/primitives/pkg/encoding/hex/bytes.go#L95, mod/primitives/pkg/encoding/ssz/schema/definitions.go, payload/pkg/cache/payload_id.go#L69, execution/pkg/client/engine.go#L85, execution/pkg/client/ethclient/rpc/client.go#L46, execution/pkg/client/ethclient/rpc/client.go#L53, execution/pkg/client/ethclient/rpc/client.go#L118, execution/pkg/client/client.go#L101, execution/pkg/client/client.go#L109, consensus/pkg/cometbft/service/middleware/abci.go#L292, consensus/pkg/cometbft/service/middleware/abci.go#L311, chain-spec/pkg/chain/data.go#L58, beacon/validator/types.go#L202, consensus/pkg/cometbft/service/middleware/abci.go#L86, consensus/pkg/cometbft/service/types.go#L44, node-api/engines/echo/vaildator.go

Description: This issue will contain a list of all of the misspellings and outdated comments. This is a WIP and will get edited throughout the audit.

Recommendation:

- mod/payload/pkg/attributes/factory.go#L61: `CreateAttributes` → `BuildPayloadAttributes`.
- mod/payload/pkg/cache/payload_id.go#L120: `Prune` → `prunePrior`.
- mod/consensus-types/pkg/types/block.go#L196: `GetSlot retrieves the slot` → `GetProposerIndex retrieves the proposer index`.
- mod/primitives/pkg/constraints/basic.go#L39: `EmptyWithForkVersion` → `EmptyWithVersion`.
- mod/primitives/pkg/constraints/basic.go#L50: `IsNil` → `Nillable`.
- mod/primitives/pkg/encoding/hex/bytes.go#L85: `MustFromHex` *rightarrow* `MustToBytes`.
- mod/primitives/pkg/encoding/hex/bytes.go#L95: `FromHex` → `ToBytes`.
- mod/primitives/pkg/encoding/ssz/schema/definitions.go: Whole file has many incorrect godocs and quite a few missing.
- payload/pkg/cache/payload_id.go#L69: Incomplete godoc.
- execution/pkg/client/engine.go#L85: `engine_forkchoiceUpdatedV1` → `engine_forkchoiceUpdatedVX`.
- execution/pkg/client/ethclient/rpc/client.go#L46: `jwtRefershInterval` → `jwtRefreshInterval`.
- execution/pkg/client/ethclient/rpc/client.go#L53: `New` → `NewClient`.
- execution/pkg/client/ethclient/rpc/client.go#L118: `Call` → `CallRaw`.
- execution/pkg/client/client.go#L101: `Clien` → `Client`.
- execution/pkg/client/client.go#L109: `connection connection` → `connection`.
- consensus/pkg/cometbft/service/middleware/abci.go#L86: `prepareProposal` → `PrepareProposal`.
- consensus/pkg/cometbft/service/middleware/abci.go#L292: `createResponse` → `createProcessProposalResponse`.
- consensus/pkg/cometbft/service/middleware/abci.go#L311: `EndBlock` → `FinalizeBlock`.
- chain-spec/pkg/chain/data.go#L58: `DomainDomainTypeProposerProposer` → `DomainTypeProposer`.
- beacon/validator/types.go#L202: `ProcessSlot` → `ProcessSlots`.
- consensus/pkg/cometbft/service/types.go#L44: `GetValidatorIndexByCometBFTAddress returns the validator index by the` → `ValidatorIndexByCometBFTAddress returns the validator index by the cometBFT address`.
- node-api/engines/echo/vaildator.go: filename `vaildator.go` → `validator.go`.

Berachain: Acknowledged. Some of these fixed during progress, some left out. No big deal.

Spearbit: Acknowledged.

5.5.30 Resolve all TODOs for production readiness

Severity: Informational

Context: geth-primitives/pkg/deposit/contract.go#L23-L25, node-api/handlers/beacon/types/response.go#L75, consensus-types/pkg/types/withdrawal_credentials.go#L62

Description: Multiple TODOs across the entire repository under review must be addressed, and all TODO comments must be removed/fixed. This will improve code quality, reduce technical debt, and implement all pending tasks correctly.

1. Remove irrelevant comments in geth-primitives/pkg/deposit/contract.go#L23-L25.
2. The `CommitteeData` struct in node-api/handlers/beacon/types/response.go#L75 file is unused and can be removed.
3. consensus-types/pkg/types/withdrawal_credentials.go#L62: TODOs can be removed from `UnmarshalJSON`, `String`, `MarshalText`, and `UnmarshalText`. `WithdrawalCredentials` is a different type than `common.Bytes32`, so it has to explicitly define its own `MarshalJSON` and `UnmarshalJSON` methods. This is unlike `common.Bytes32` which does not define its own `MarshalJSON` and `UnmarshalJSON` because it is not its own type, instead it is a type alias of `B32`.

Recommendation: Please ensure all pending tasks are properly tracked and implemented.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.5.31 Mislabeled constants `bytesPer64Bits` and `bytesPer256Bits`

Severity: Informational

Context: mod/primitives/pkg/encoding/hex/const.go#L30-L31

Description: The constants `bytesPer64Bits` and `bytesPer256Bits` are used in practice as "nibbles" instead of bytes. This is misleading when reading through the parsing code.

Recommendation: The two constants should be changed as such:

```
- bytesPer64Bits  = 16 // 64/8
- bytesPer256Bits = 64 // 256/8
+ nibblesPer64Bits  = 16 // 64/4
+ nibblesPer256Bits = 64 // 256/4
```

Berachain: Fixed in commit f6aab0ed.

Spearbit: Fix verified.